University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Attribution Methods for Explaining Transformers

Pilsen, 2023

Vojtěch Bartička

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Vojtěch BARTIČKA**
Osobní číslo:	**A21N0038P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Atribuční metody pro Transformer modely**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte post-hoc metody pro vysvětlování rozhodnutí modelu strojového učení v oblasti zpracování přirozeného jazyka, které lze aplikovat na Transformer modely.
2. Prostudujte způsoby evaluace metod z bodu 1 a dostupné evaluační sady.
3. Vytvořte vlastní evaluační sadu na úloze klasifikace zpravodajských textů v českém jazyce.
4. Vybrané metody implementujte a jejich funkcionalitu demonstrujte na několika úlohách zpracování přirozeného jazyka.
5. Změřte úspěšnost implementovaných metod na vybraných evaluačních sadách a kriticky zhodnoťte výsledky.

Rozsah diplomové práce:	**doporuč. 50 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná/elektronická**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce:	**Ing. Ondřej Pražák** Nové technologie pro informační společnost
Datum zadání diplomové práce:	**9. září 2022**
Termín odevzdání diplomové práce:	**18. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D. děkan	**Doc. Ing. Přemysl Brada, MSc., Ph.D.** vedoucí katedry

V Plzni dne 11. října 2022

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, 16th May 2023

Vojtěch Bartička

Acknowledgements

I would like to express my sincere gratitude to my advisor, Ing. Ondřej Pražák, for his invaluable advice and patience.

Computational resources were provided by the e-INFRA CZ project, supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Abstract

In this thesis, we evaluate multiple attribution methods applicable to Transformer models using the SST and CTDC datasets. We extend the CTDC dataset by adding ground-truth annotations based on keywords and pointwise mutual information, creating a ground-truth evaluation benchmark for the Czech language. We fine-tune seven models of various sizes and architectures with five instances each, allowing us to measure the effect of random initialization and model size. We also evaluate a distilled multilingual model on the CTDC dataset, showing that it makes rational decisions even when used with a language less represented in the pre-training process. We test attribution methods with different baseline references and sample counts, providing valuable insight for practical applications. We show that overfitting negatively affects gradient-based attribution methods, while KernelSHAP sees little performance degradation.

Abstrakt

Tato práce zkoumá atrubuční metody aplikovatelné na Transformer modely pomocí datových sad SST a CTDC. Do datové sady CTDC přidáváme anotace založené na klíčových slovech a bodové vzájemné informaci, čímž umožňujeme evaluaci atribučních metod na české datové sadě. Používáme sedm modelů různých velikostí a architektur, každý s pěti instancemi, což nám umožňuje měřit vliv náhodné inicializace a velikosti modelu. Používáme také destilovaný vícejazyčný model na datové sadě CTDC a ukazujeme, že se rozhoduje racionálně i při použití s jazykem méně frekventovaným v předtrénování. Testujeme atribuční metody s různými referenčními vstupy a počty vzorků, což poskytuje cenné poznatky pro praktické aplikace. Ukazujeme, že přeučení negativně ovlivňuje atribuční metody využívající gradient, zatímco u metody KernelSHAP, která gradient nevyužívá, dochází k velmi malému zhoršení.

Contents

1 Introduction

Explainable Artificial Intelligence (XAI) is an area of active research that seeks to provide insight into the decision-making process of machine learning models. Machine learning models gradually became more complex and opaque to the user. While simple machine learning models can be self-explanatory (e.g., linear regression models (Ribeiro et al., 2016)) and we understand how they behave, neural networks and, recently, large models such as Transformers pose a new challenge.

AI models have succeeded in many fields, such as medicine or law. In some areas of application, it does not suffice that the model makes the right decision, but the reason it made such a decision is also crucial. Knowing how the decision is made can be as important as the decision itself. XAI provides methods to make complex models more transparent and trustworthy (Gohel et al., 2021). This thesis defines these methods as *attribution methods* and the explanations as *attributions*.

Natural Language Processing (NLP) is an established field of Artificial Intelligence research. It has seen success in many applications - text classification, translation, and text generation, among many others. NLP has seen a renaissance in recent years with the introduction of the Transformer architecture that has redefined state of the art and has been the architecture of choice since then (Wolf et al., 2020).

This thesis extends *Evaluating Attribution Methods for Explainable NLP with Transformers*, a published article from Bartička et al. (2022). We focus on methods for explaining model decisions in the context of NLP. Specifically, we focus on Transformer models. We provide a new Czech dataset for evaluating attributions. We use multiple models of different sizes to provide more insight into how the model size affects the decisions and how the attribution methods behave. We also use multiple architectures. For some of the evaluated attribution methods, we experiment with different hyperparameters.

In the first part of this thesis, we focus on the theoretical background of the Transformer architecture, attribution methods, and evaluation of attribution methods. In the second part, we describe the new dataset and the evaluation methodology and discuss the results.

2 Transformer Networks

2.1 Pre-trained Models

The idea of transfer learning came from the challenge of obtaining large high-quality datasets to train models. While some tasks (e.g., image classification) have large datasets available, such datasets are not available for all tasks. Creating datasets from scratch can be costly, especially when the labeling has to be done by an expert (Han et al., 2021).

Instead of training a model from scratch, sharing acquired knowledge between different tasks is possible. There are two main transfer learning approaches - feature transfer and parameter transfer. Feature transfer encodes knowledge into features injected into the target task. Parameter transfer assumes that the source task and target task can share the same parameters. Fine-tuning the transferred parameters on the target task then realizes the knowledge transfer (Han et al., 2021).

2.1.1 Pre-trained Word Embeddings

One of the first applications of transfer learning in NLP was semantic word representations such as Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). Using pre-trained embedding vectors to train a model on an end task significantly improved performance compared to random initialization (Han et al., 2021).

The next step was sequence-level embeddings. While Word2Vec and GloVe were word-level semantic representations, the semantic meaning of a word depends on its context. ELMo used a pre-trained bi-directional Long Short-Term Memory (LSTM) network to produce sequence-level embeddings and improved state of the art in multiple NLP tasks (Peters et al., 2018).

2.1.2 Pre-trained Language Models

The introduction of Transformers (Vaswani et al., 2017) created a new trend in the field of NLP. After the Transformer, pre-trained language models, such as BERT (Devlin et al., 2018) or GPT (Radford et al., 2018), were proposed. Unsupervised training in combination with a language modeling task (e.g., masked language modeling (Devlin et al., 2018)) produced pre-trained models, which could be used as a starting point for many different

NLP tasks, creating new state-of-the-art results (Devlin et al., 2018).

Since then, Transformer-based pre-trained language models have become the standard approach to solve NLP tasks (Han et al., 2021).

2.2 Attention

The attention mechanism computes alignment scores between elements from two inputs - a key and a query. The alignment scores represent how relevant a key is to the query. We want more important keys to have higher alignment scores. The attention mechanism allows the network to observe specific input elements, assigning high weights to the important ones.

Formally, attention computes attention weights a for a query q and key-value pairs k and v using a compatibility function f (Equation 2.1). Attention weights a are then processed with softmax to form probabilities p, which are then used to create a weighted average of v (Figure 2.1).

$$a = [f(k_i, q)]_{i=1}^{n} \tag{2.1}$$

In the context of Recurrent Neural Network-based (RNN-based) encoder-decoder architectures, the keys and values can be the hidden states of the encoder, and the query can be the hidden state of the decoder. In this case, attention weights represent the dependencies between the current decoder output and the input sequence elements.

There are many variants of the compatibility function f. The most commonly used are additive and multiplicative attention (Shen et al., 2017).

Additive Attention

Additive attention uses one dense layer to calculate the alignment score. The compatibility function is shown in Equation 2.2. The w, $W^{(1)}$ and $W^{(2)}$ are learnable parameters (Shen et al., 2017).

$$f(k_i, q) = w^T \sigma(W^{(1)} k_i + W^{(2)} q) \tag{2.2}$$

Multiplicative Attention

Multiplicative attention (also dot-product attention) uses an inner product or cosine similarity to compute alignment between a query and a key (Equation 2.3). The $W^{(1)}$ and $W^{(2)}$ are learnable parameters. Multiplicative attention can be performed efficiently using matrix multiplication (Shen et al., 2017).

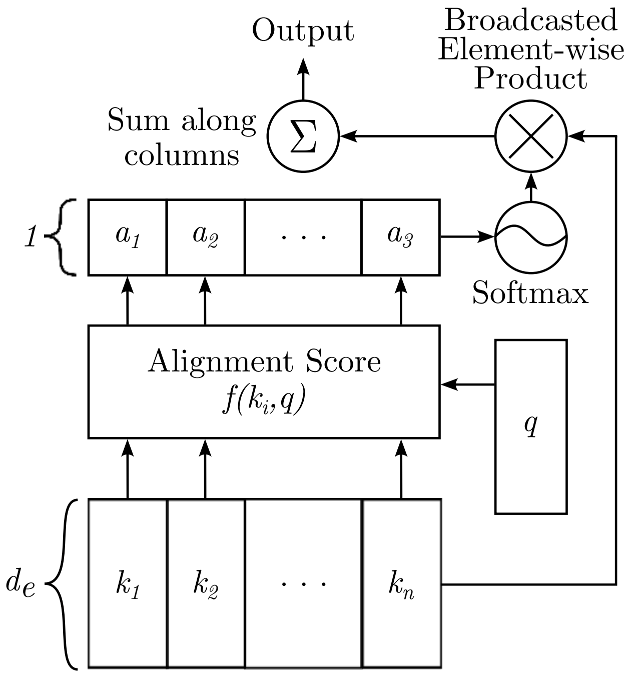


Figure 2.1: A schema of the attention mechanism. Figure from Shen et al. (2017).

$$f(k_i, q) =< W^{(1)}k_i, W^{(2)}q > \tag{2.3}$$

A special case of the attention mechanism is self-attention. Self-attention replaces the query q with another input element. In other words, keys k and queries q originate from the same input sequence, which enables the network to analyze dependencies between the input elements. Self-attention thus substitutes the long-range dependency modeling usually performed by RNNs and the local-context modeling performed by Convolutional Neural Networks (CNNs) (Shen et al., 2017).

2.3 Transformer

The Transformer is an encoder-decoder architecture introduced by Vaswani et al. (2017). At the time, RNNs and their variants, such as LSTM networks or Gated Recurrent Units (GRU), were state-of-the-art in language modeling and sequence-to-sequence tasks. RNNs, however, are sequential. RNNs create a sequence of hidden states, where hidden state h_t depends on the previous state h_{t-1}. These dependencies make parallelization across input positions hard (Vaswani et al., 2017).

The Transformer architecture solves this problem. It does not rely on recurrence, which makes it possible to process input positions in parallel. Other architectures before the Transformer aimed to reduce the - at the time - serial nature of sequence processing. They all used CNNs (Vaswani et al., 2017). These architectures had issues modeling long-range dependencies, as the number of operations needed to relate two different positions was not constant. The Transformer instead uses the attention mechanism (specifically self-attention) to form relations between different input positions, forgoing convolution and recurrence (Vaswani et al., 2017).

The Transformer is not the first architecture to use attention. RNN-based encoder-decoder architectures used attention to improve the performance by allowing the decoder to look at any of the hidden states of the encoder instead of just the output of the encoder. According to the authors, the Transformer was the first architecture to use only self-attention.

2.3.1 Encoder and Decoder

The encoder creates a contextual representation of the input elements. This representation is then passed to the decoder, which generates the output sequence. The Transformer is autoregressive, meaning the elements of the output sequence are generated one by one by feeding the previous output back into the decoder. See Figure 2.2 for an overview of the encoder and decoder blocks.

Encoder

The Transformer's encoder comprises six architecturally identical layers. Each layer has two sub-layers - a multi-head attention sub-layer and a dense position-wise feed-forward sub-layer. Each sub-layer has a residual connection and a layer normalization. As such, the output of each sub-layer corresponds to $LayerNorm(x + SubLayer(x))$, where x is the layer input and $SubLayer$ is the function performed by the sub-layer. Each sub-layer produces an output of d_{model} dimensions, which allows for the sum of the residual connection and the sub-layer output.

We first embed the input sequence into d_{model}-dimensional vectors. Then we add positional embeddings. We use these embeddings as the input to the first layer of the encoder.

Each of the six layers in the encoder first passes the embeddings to the attention sub-layer. There, multi-head attention (see Section 2.3.2) enriches the token embeddings with information from other input positions. We pass

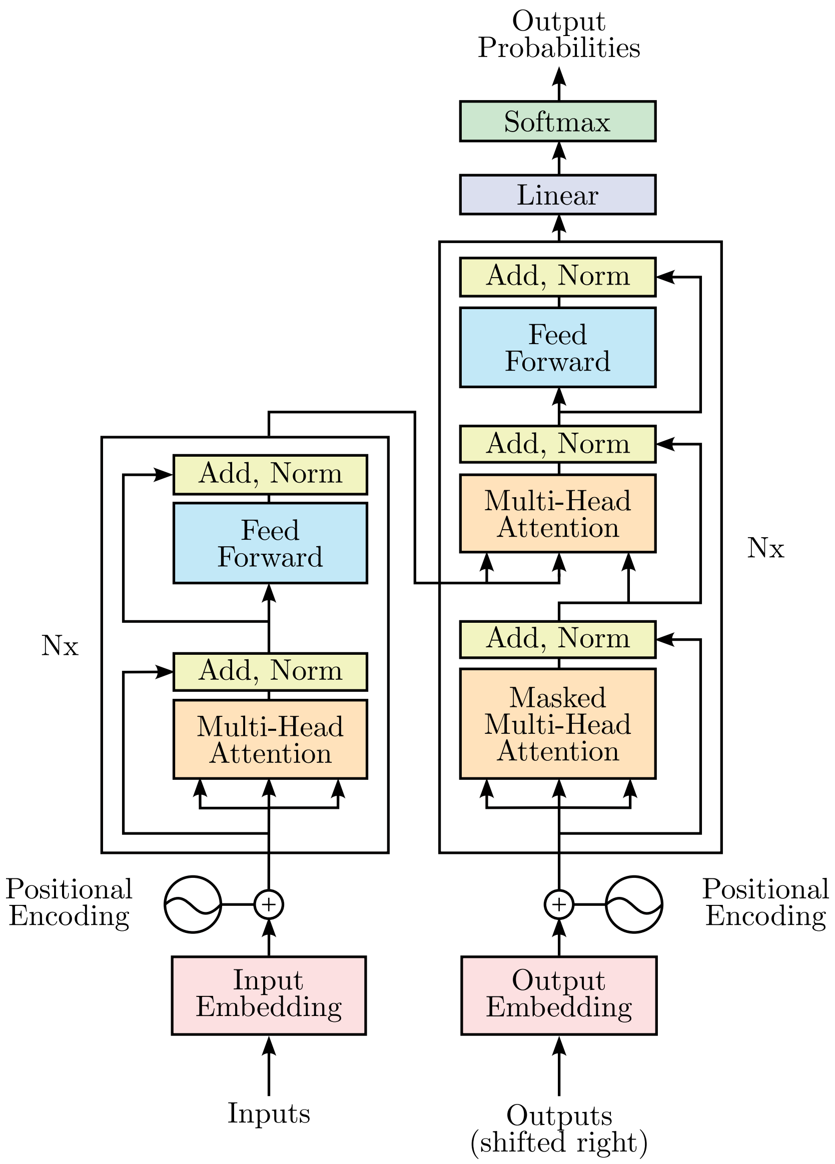


Figure 2.2: The Transformer architecture. On the left is a single encoder block, stacked N times. On the right is a single decoder block, stacked N times. Figure from Vaswani et al. (2017).

the output of this sub-layer to the dense feed-forward sub-layer, which performs a position-wise transformation of each input element (Vaswani et al., 2017).

Decoder

Like the encoder, the decoder comprises six architecturally identical layers. Each layer contains three sub-layers - a multi-head attention sub-layer, a multi-head cross-attention sub-layer, and a dense position-wise feed-forward sub-layer. Like the encoder, each sub-layer has a residual connection followed by layer normalization.

Decoder generates the output sequence in an autoregressive manner. We use the output of the decoder as input to the decoder. For a given sequence, the decoder produces probabilities of tokens using a dense feed-forward layer followed by softmax. When we return the resulting token index as the input, we embed it into a continuous vector-space representation (in the same manner as in the encoder) and add positional embeddings.

The first attention sub-layer performs self-attention on the decoder block inputs. This attention sub-layer is different from the encoder. As the decoder is autoregressive, it is necessary to prevent the decoder from conditioning on future tokens. We achieve this by applying a mask to the attention scores, which sets the attention scores related to future tokens to $-\infty$. Applying softmax on the masked attention scores then yields a probability of zero.

The second attention sub-layer performs cross-attention between the encoder output and the decoder input. We use the encoder output as the queries and keys, while the values are outputs of the decoder block's first attention sub-layer. Cross-attention allows the decoder to consider the contextual representation formed by the encoder. We then process the output of the second attention sub-layer with a dense position-wise feed-forward network (Vaswani et al., 2017).

Positional embeddings

As the model is not recurrent, passing the positional information to the model is necessary. We achieve this through positional embeddings. Positional embeddings can be fixed or learned. The authors of Transformer experimented with both approaches and found the results very similar. They opted to use fixed embeddings based on the assumption that the model could handle sequence lengths not seen in the training data more effectively.

The authors used cosine and sine functions to generate the positional embeddings (Equation 2.4). The positional embeddings are then summed

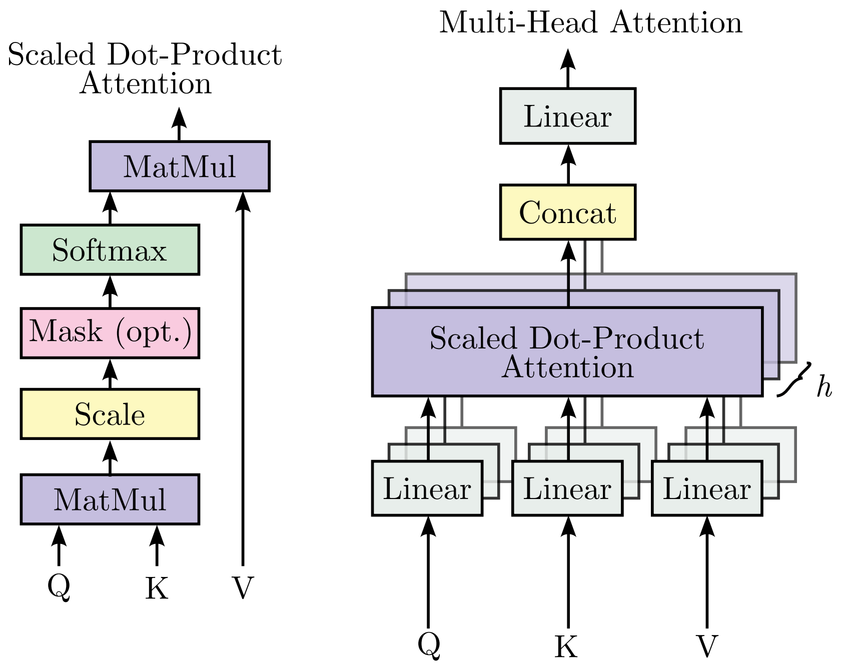


Figure 2.3: A schema of the scaled dot-product attention on the left. A schema of the multi-head attention on the right. Figure from Vaswani et al. (2017).

with the input embeddings and passed to the encoder layers (Vaswani et al., 2017).

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.4}$$

2.3.2 Attention

Scaled Dot-Product Attention

The Transformer uses scaled dot-product attention (Equation 2.5, Figure 2.3), a variant of multiplicative attention. The authors used dot-product instead of additive attention to leverage highly optimized matrix multiplication and a smaller memory footprint. The authors found that the additive and dot-product attention perform similarly for small vector dimensions. However, when the dimensionality increases, the unscaled dot-product attention performs worse than the additive attention. The authors argue that the dot products grow very large for larger vector sizes, which results in small gradients. The scaling factor mitigates this issue (Vaswani et al., 2017).

$$A = \frac{Q * K}{\sqrt{d_{model}}} * V \tag{2.5}$$

Multi-Head Attention

One of the main features of the Transformer architecture is the multi-head attention. Instead of using a single attention head on the keys, queries, and values, the authors propose to use h attention heads, where each head has three separate projection layers for queries, keys, and values.

The projection layers allow different attention heads to focus on different objectives. They also control the dimensionality of the queries, keys, and values. Authors project the d_{model}-sized vectors into d_k dimensions, where $d_k = d_{model}/h$. The attention heads produce h representations for each input element, which are then concatenated and passed through a dense position-wise feed-forward layer (Equation 2.6 and Figure 2.3) (Vaswani et al., 2017).

$$\begin{gathered} MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^o, \\ where\ head_i = Attention(QW_i^q, KW_i^k, VW_i^v) \end{gathered} \tag{2.6}$$

2.4 BERT

BERT is an acronym for Bidirectional Encoder Representations from Transformers and was introduced by Devlin et al. (2018). It addresses one of the limitations of previous pre-trained Transformer-based models. Models like GPT prevent attention to future tokens. The input sequence is processed left-to-right. The authors argue that this approach is not desirable for token-level and sentence-level tasks, where the context to the right of a token is important.

2.4.1 Masked Language Modeling and Next Sentence Prediction

BERT uses two language modeling tasks for pre-training - Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

MLM does not restrict the model to left-to-right processing. The core idea of MLM is to mask a percentage of input tokens and train the model to predict the token ids. Masked tokens are replaced by a special `[MASK]` token. Using the `[MASK]` token during pre-training creates discrepancies between pre-training and fine-tuning tasks. As mitigation, some masked tokens are left unchanged, and some are replaced with random tokens.

Relationships between sequences are essential for question answering (QA) and other similar tasks. Because MLM does not model these relationships, the authors added the NSP task. In NSP, the model learns to predict whether or not sentence B follows sentence A. The authors experimented with removing NSP as a pre-training objective and observed worse performance on QA tasks (Devlin et al., 2018).

2.4.2 Architecture

BERT only uses an encoder, but the architecture of the encoder blocks is the same as in the original Transformer (see Section 2.3).

The NSP task requires the input to be modified. Each sequence starts with a `[CLS]` token. Each sequence for the NSP task ends with a `[SEP]` token. Segment embeddings are added to the token embeddings and position embeddings, which allows the model to differentiate between the first and the second sequence. The segment embeddings are learned in pre-training.

The authors trained two models - a *base* model with 110M parameters and a *large* model with 340M parameters.

2.4.3 Distilled BERT Models

The inference speed of a large language model may be a limiting factor in real-world applications. We can use a large model as a teacher for a smaller student model. This arrangement is commonly referred to as *teacher-student learning* or *knowledge distillation*. Knowledge distillation often uses soft labels (the output distribution of a model), as opposed to hard labels (e.g., one-hot vectors), to allow the student to mimic the behavior of the teacher model. Different knowledge distillation techniques were applied to BERT models.

Sun et al. (2019) proposed *patient knowledge distillation*, which incorporates information from different layers of the teacher BERT model into the distillation process. While vanilla knowledge distillation uses only the hidden states of the last layer to compute soft labels, their approach also uses intermediate layers. The student is then taught to imitate the teacher's behavior in the intermediate layers, not only the last layer. Their approach first transfers layers from the teacher model to the student model and then distills knowledge from a teacher fine-tuned on a target task.

Sanh et al. (2019) introduced DistilBERT. Their distillation process includes transferring teacher model layers to the student model and then performing language modeling pre-training assisted by the teacher.

Turc et al. (2019) proposed *pre-trained distillation*, which first pre-trains the student model on a language modeling task without a teacher. Then the knowledge from the teacher is distilled using soft labels. Lastly, the student model can be fine-tuned with or without a teacher.

Sun et al. (2019) and Sanh et al. (2019) reduced the number of layers in the student model compared to the teacher model. As they transferred layers from the teacher model, their approaches only allowed limited configuration changes of the encoder layers. As Turc et al. (2019) do not transfer layers from the teacher to the student, they have more freedom to change the configuration of the student model.

They, however, all show that distilled models can be significantly smaller, faster, and still perform well compared to their teacher models.

2.4.4 RoBERTa

Liu et al. (2019) examined the training procedure used in the original BERT model. Their findings resulted in a modified training procedure, improving performance on downstream tasks.

They increased the amount of training data, the training batch size from 256 in the original BERT model to 8K, and the number of training steps. They removed the NSP pre-training objective and modified the sampling of the text. While BERT uses two text segments for the NSP task, RoBERTa samples sentences until the limit of 512 tokens. The 512-token input may cross document boundaries.

Conneau et al. (2019) used the RoBERTa pre-training approach to train multilingual models. Wang et al. (2020) later distilled these models using a novel knowledge distillation process.

2.4.5 ELECTRA

ELECTRA (Clark et al., 2020) is a BERT-like architecture that (similar to RoBERTa) modifies the pre-training process. The authors question the efficiency of the MLM pre-training objective and propose a more sample-efficient alternative.

The MLM objective masks out a small part of the input tokens with `[MASK]` tokens. The model then learns to predict the original word. The authors of ELECTRA argue that this approach is sample-inefficient, as the task is defined only over the masked tokens (15% in BERT pre-training (Devlin et al., 2018)).

The proposed method is called *replaced token detection*. Instead of replacing the input tokens with a masking token, the input tokens are replaced by plausible alternatives supplied by a generative network. The discriminative model then learns to distinguish between a replaced token and an original token. This approach utilizes the entire input, not just the replaced tokens. It also solves the discrepancy between BERT pre-training and fine-tuning, as the discriminative model does not see any masking tokens.

The authors show that the proposed pre-training objective outperforms BERT MLM+NSP when used on similarly-sized models while requiring less compute. It also performs well with small models (Clark et al., 2020).

3 Attribution Methods

Attribution methods are processes that give us insight into the decision-making process of a model. As the complexity of machine learning models increased, attribution methods have gained significance. They have become a valuable tool for machine learning developers, who can use them to analyze model decisions to improve them. Attribution methods were also used to improve the learning process (Erion et al., 2021).

There are also legal motivations to research attribution methods. The General Data Protection Regulation (GDPR) contains articles that restrict automatic decisions and also contain text about the right to "meaningful information about the logic involved" (Confalonieri et al., 2021; Selbst and Powles, 2017). The rapid adoption of automated systems raises concerns about discrimination, bias, and unfairness. Attribution methods (and broadly Explainable AI) help solve these problems (Confalonieri et al., 2021).

3.1 Taxonomy

The field of Explainable AI is complex and does not have a clear taxonomy of attribution methods. Different surveys of the Explainable AI field have classified the methods based on different properties.

Angelov et al. (2021) differentiate between opaque and transparent models and focus on post-hoc attribution methods. They then classify these methods as either model-agnostic or model-specific. The classification is then broken down further based on the specific approach implemented - feature-oriented, local, global, simplification, and human-centric methods.

Barredo Arrieta et al. (2020) provide a complex and detailed overview of the Explainable AI field and a taxonomy of attribution methods. The taxonomy is more fine-grained compared to Angelov et al. but, on a high level, captures the same classification. It differentiates between transparent models and opaque models (post-hoc attributions). For post-hoc attribution methods, it differentiates between model-agnostic and model-specific methods. These two classes are then divided based on the approach the methods use, and in the case of model-specific methods, it also deals with specific architectures.

Linardatos et al. (2021) (Figure 3.1) identify four main properties of attribution methods - the purpose of use, model specificity, locality of the method, and the data type concerned (e.g., text or image).

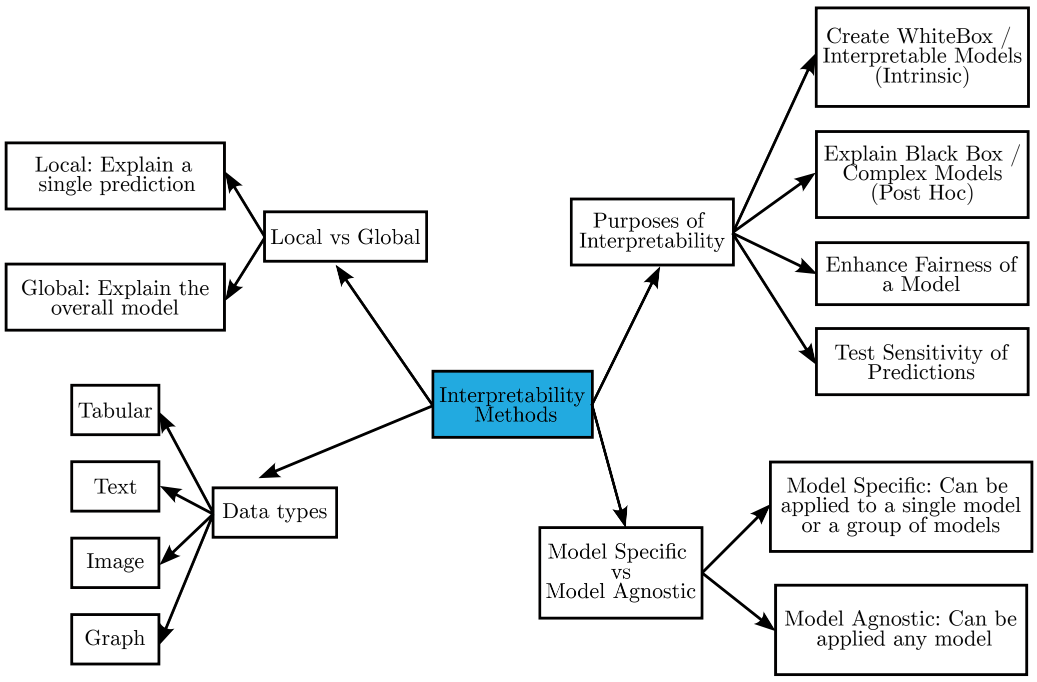

Interpretability Methods
Local vs Global
Local: Explain a single prediction
Global: Explain the overall model
Purposes of Interpretability
Create WhiteBox / Interpretable Models (Intrinsic)
Explain Black Box / Complex Models (Post Hoc)
Enhance Fairness of a Model
Test Sensitivity of Predictions
Data types
Tabular
Text
Image
Graph
Model Specific vs Model Agnostic
Model Specific: Can be applied to a single model or a group of models
Model Agnostic: Can be applied any model

Figure 3.1: An overview of the taxonomy proposed by Linardatos et al. Image from Linardatos et al. (2021).

For this thesis's theoretical analysis of attribution methods, we broadly follow the taxonomy proposed by Linardatos et al. (2021). It is less complex and does not impose model-based restrictions like Barredo Arrieta et al. (2020) does. Moreover, Barredo Arrieta et al. (2020) do not consider Transformer-based architectures in their taxonomy, which is limiting given the focus of this thesis.

In this thesis, we segment the methods based on their implementation. However, we provide the classification based on the Linardatos et al. (2021) taxonomy for each method.

Additionally, we provide a brief overview of the Linardatos et al. taxonomy and place it in the context of this thesis.

Purpose of use Linardatos et al. (2021) distinguish between four purposes of use - to explain a black-box model (post-hoc), to create a white-box model (intrinsic or ad-hoc), to enhance the predictions of a model, and to test the sensitivity of a model. This thesis focuses on post-hoc attributions, and as such, the other categories are not relevant.

Model specificity Model-specific methods require the model to fulfill some requirements in order for them to be applicable. An example of such a requirement is differentiability. Model-agnostic methods can be applied to any model, regardless of its internals. This thesis considers both model-specific and model-agnostic methods.

Data type Linardatos et al. (2021) define multiple data types, out of which only text is relevant in the context of this thesis.

Locality of the method Linardatos et al. (2021) describe local methods as providing attributions for only a specific input and global methods as explaining the entirety of the model. In the Linardatos et al. taxonomy context, this thesis considers only local methods.

3.2 Gradient-based Methods

3.2.1 Gradients and Gradients x Inputs

Gradients (Vanilla Gradients or Sensitivity Maps) are model-specific attribution methods. The intuition behind gradients (and other gradient-based methods) is that calculating gradients of the output w.r.t. the input features produces values, which tell us how big of an impact changing an input feature will have on the prediction.

A variant of this method, which uses absolute values of the gradients, is called Saliency Map (Simonyan et al., 2014).

Multiplying the gradients with input features (Gradients x Input) can improve the quality of attributions. This behavior was studied multiple times (Smilkov et al., 2017; Sundararajan et al., 2017). While the benefit was clear, the theoretical justification for it was not. Ancona et al. (2017) then proposed the concept of *local* and *global* attributions as a reason for the behavior. When we describe a method as *local* or *global*, we refer to Ancona et al. (2017) and not the Linardatos et al. (2021) taxonomy.

Local attributions *Local* attributions explain how a small change in the input will affect the output (intuitively, gradients). In other words, *local* attributions provide information about which feature has to be changed to obtain the desired output.

Global attributions *Global* attributions describe the marginal effect a change in the input will have with respect to a baseline. In other words, *global* attributions provide information about which feature had the most impact on the output. Gradients x Inputs is an example of a *global* attribution method, where the baseline is implicit and all-zero.

Ancona et al. (2017) provide a great and intuitive example using a linear model and an investment analogy.

3.2.2 SmoothGRAD

SmoothGRAD is a model-specific local (Ancona et al., 2017) attribution method proposed by Smilkov et al. (2017). They examine the performance of Vanilla Gradients and provide insight into why it produces noisy attributions.

They show that Vanilla Gradients fluctuate significantly in a small area around the input. This fact makes Vanilla Gradients unsound as an attribution method since a minor change to a feature should intuitively not result in a significant change in attributions. The authors include a visual example.

Based on these findings, they propose a new method based on Vanilla Gradients. This method creates n samples by adding Gaussian noise $\mathcal{N}$ to the input x, effectively smoothing the gradients with a Gaussian kernel. Then gradients M_c w.r.t. the input are calculated for each of the n samples and averaged, producing the final attributions $\hat{M}_c$ (Equation 3.1).

$$\hat{M}_c = \frac{1}{n} \sum_{1}^{n} M_c(x + \mathcal{N}(0, \sigma^2)) \tag{3.1}$$

From Equation 3.1, we can see that the number of samples n and the standard deviation σ are hyperparameters. The authors experiment with different n ranging from 2 to 100. They also experiment with different noise levels, determining the standard deviation from the noise level according to Equation 3.2.

They found that noise levels between 0.1 and 0.2 produce sharp sensitivity maps while preserving the image's structure. The authors saw diminishing returns when using sample size $n > 50$.

$$\sigma = noise\_level * (x_{max} - x_{min}) \tag{3.2}$$

3.2.3 Integrated Gradients

Integrated Gradients are a global (Ancona et al., 2017) model-specific attribution method introduced by Sundararajan et al. (2017).

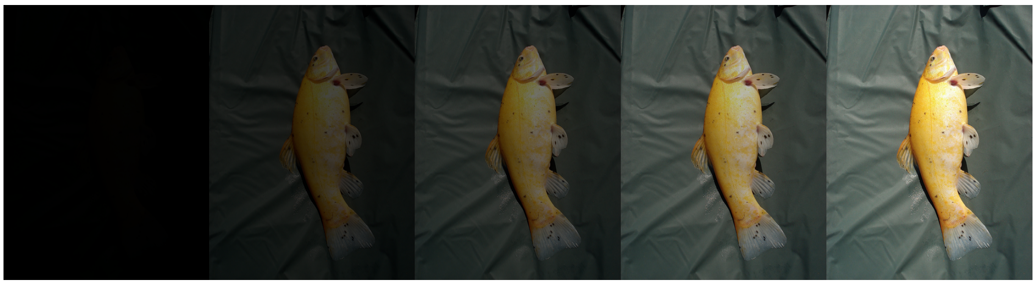


Figure 3.2: An example of linear interpolation using a black image as a baseline. The original image[1] is on the right.

The authors first examine two desirable axioms any attribution method should satisfy and find that many known attribution methods do not satisfy them.

The first axiom is *Sensitivity*. Let x be an input, x' be a baseline, $F(x)$ be the model prediction for the input, and $F(x')$ be the model prediction for the baseline such that $F(x) \neq F(x')$. A method satisfies the axiom of Sensitivity if, for every such input x and baseline x' that differ in one feature i, the feature i receives a non-zero attribution. Also, if a prediction does not depend on an input feature, the input feature should have zero attribution. Gradients break the axiom of Sensitivity.

The second axiom is *Implementation Invariance*. Networks are functionally equivalent if they produce the same output for every input regardless of implementation. Two functionally equivalent networks should have identical attributions if a method satisfies the Implementation Invariance axiom. The authors show that Layer-wise Relevance Propagation (LRP) and DeepLIFT break this axiom (Sundararajan et al., 2017).

The authors then propose a new attribution method called Integrated Gradients that satisfies both axioms.

Integrated Gradients require a baseline (reference) input. A baseline should be neutral or represent an absence of information. The authors suggest using a black image for image-related tasks or all-zero embeddings for text-related tasks. Note that the baseline is directly related to the axiom of Sensitivity.

Integrated Gradients also satisfy the axiom of *Completeness*. When a method satisfies this axiom, the total sum of attributions is equal to the difference between the output of a network F at the target input x and the baseline x'. This axiom is satisfied by other attribution methods such as LRP or DeepLIFT (Sundararajan et al., 2017).

The method proposed integrates the gradients while moving along the path from a baseline to the target input (Equation 3.3). In practice, this integral is approximated with linear interpolation (Figure 3.2) between the baseline and the target input in n steps. For each step, we calculate a gradient. We then average the gradients from all steps. We then multiply the averaged gradients by the difference between the target input and the baseline (Equation 3.4)(Sundararajan et al., 2017).

$$IG(x) = (x - x') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(x - x'))}{\partial x} d\alpha \tag{3.3}$$

$$IG^{approx}(x) = (x - x') \times \sum_{i=1}^{n} \frac{\partial F(x' + \frac{i}{n}(x - x'))}{\partial x} \times \frac{1}{n} \tag{3.4}$$

The n is a hyperparameter. The authors suggest that n between 20 and 300 should be sufficient. Additionally, we can use the Completeness axiom to check whether the n is high enough. The authors recommend raising the n until the sum of attributions is within 5% of the difference between the target and baseline outputs.

Choosing an appropriate baseline is also essential. For example, using a black image as a baseline when classifying whether or not an image contains a black circle against a white background will produce zero-only attributions for the circle pixels (Sundararajan et al., 2017). For text, the authors recommend all-zero embeddings as a baseline. Tan (2022) examine the choice of baselines for image-related tasks in detail, but text-related tasks have seen little research in this direction.

The effect of integrating the gradients between a baseline and the input becomes clear when we look at the sum of absolute gradients at different interpolation steps (Figure 3.3). The gradients saturate at the input. Saturation is a known issue when using gradients to interpret deep neural networks. Integrated Gradients circumvent this issue by integrating along the path between a baseline and the input (Sturmfels et al., 2020).

3.3 Attention-based Methods

Using attention weights to gain insight into the inner workings of a model is an intuitive and popular approach (Abnar and Zuidema, 2020). Whether or not this approach provides good explanations is a point of contention.

Jain and Wallace (2019) argue that attention weights are not good explanations when using Bidirectional LSTM networks. Wiegreffe and Pinter (2019) further examine those claims and suggest that attention weights can

[1] Image from https://github.com/EliSchwartz/imagenet-sample-images

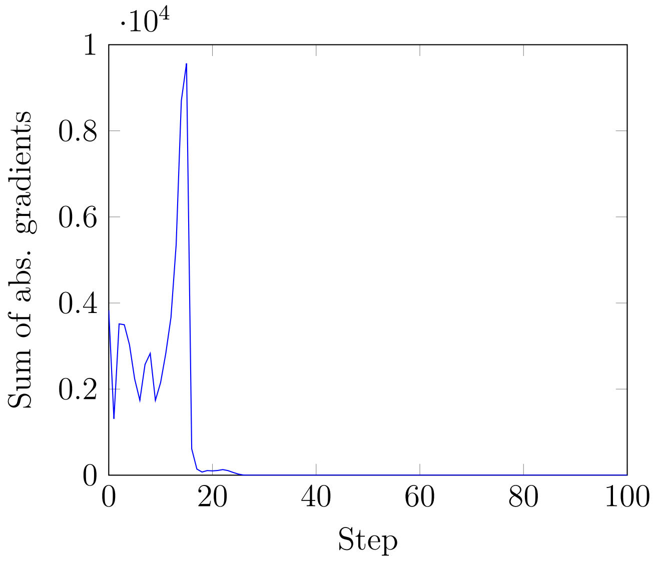


Figure 3.3: The sums of absolute gradients at different interpolation steps. Step 0 corresponds to baseline embeddings, and step 100 corresponds to target embeddings (the original input sequence). A *bert-base-cased* model fine-tuned on the SST dataset was used with all-zero embeddings as a baseline.

be meaningful explanations. Pruthi et al. (2020) analyze the attention mechanism in BERT by preventing it from focusing on words important to the task - *impermissible words*. They add a term that penalizes high attention weights on impermissible tokens to the loss function. Noting that it does not lead to significant performance reduction while still allowing the model to focus on important tokens, they call into question the use of attention weights as an auditing tool.

Nonetheless, attribution methods utilizing the attention mechanism, especially in the context of Transformer-based architectures, were proposed and performed well compared to other methods (Abnar and Zuidema, 2020).

3.3.1 Raw Attention

Raw Attention can be used to interpret model decisions (Abnar and Zuidema, 2020; Chefer et al., 2020). There are inherent problems with using attention weights as an attribution. These problems are discussed further in Section 3.3.2. Raw Attention performs worse than more complex methods (Abnar and Zuidema, 2020; Chefer et al., 2020) and provides unsigned attributions. We do not know if the input element contributed positively or negatively to the decision.

Combining Raw Attention with additional information such as gradients

(Serrano and Smith, 2019; Liu et al., 2021) or input norm (Kobayashi et al., 2020) can improve the performance. Transformer models provide attention weights from multiple layers. Approaches using a combination of layers and a single layer were tested (Kobayashi et al., 2020).

Liu et al. (2022) use faithfulness violation test to compare multiple attribution methods, including attention-based ones. They find that all attribution methods tested violate faithfulness to a certain extent and that Raw Attention performs worst. Using Raw Attention in combination with attention gradients does improve the results. The authors note that the cause of this improvement may be the signed attributions produced by the method, as opposed to unsigned attributions from Raw Attention.

3.3.2 Attention Rollout and Attention Flow

The self-attention in Transformer architectures enriches the token representations with other token embeddings, raising a question about how much a token embedding in higher layers corresponds to the same position in the input layer. Brunner et al. (2019) show that self-attention causes heavy mixing between tokens as they move to higher layers. While tokens mostly keep their identity across the model, the authors suggest that existing techniques utilizing attention can be improved by accounting for the mixing between tokens.

Abnar and Zuidema (2020) focus on the problems highlighted by Brunner et al. (2019). They propose two attribution methods - Attention Rollout and Attention Flow.

Both methods interpret the attentions between layers as a directed acyclic graph, where nodes are token embeddings, edges are attentions between tokens, and edge weights are attention weights. The authors augment this graph to account for residual connections by adding an identity matrix to the raw attention weights W_{att} (Equation 3.5). Both methods average the token attention over the attention heads.

$$A = 0.5W_{att} + 0.5I \tag{3.5}$$

Attention Rollout assumes that the attention combines linearly between layers. Tracing the attention from layer l_i to layer l_j, where $j < i$, it recursively multiplies the attention weight matrices (Equation 3.6).

$$A(l_i) = \begin{cases} A(l_i)\tilde{A}(l_{i-1}) & \text{if } i > j \\ A(l_i) & \text{if } i = j \end{cases} \tag{3.6}$$

Attention Flow treats the attention graph as a flow network, where the attention weights are edge capacities. Attention Flow can be calculated using any maximum flow algorithm.

The authors observe that Attention Flow performs better compared to Attention Rollout.

3.3.3 Chefer et al.

Chefer et al. (2020) propose an attribution method based on relevancy propagation designed for Transformer-based architectures. The authors note that other methods, such as Layer-wise Relevance Propagation, fail to deal with negative relevance and residual connections. Moreover, in practice, many methods produce class-agnostic attributions. The proposed method deals with these issues.

Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) is a model-specific (Linardatos et al., 2021) attribution method introduced by Bach et al. (2015). It creates attributions by propagating relevance backward through the network using specially designed local propagation rules. LRP conserves the total relevance when propagating between layers.

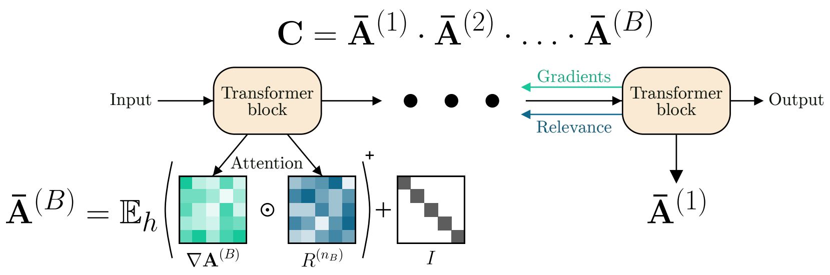
The local propagation between two neurons i and j in two consecutive layers is achieved by applying Equation 3.7, where z_{jk} represents the amount of relevance contributed by neuron j to neuron k. The calculation of z depends on the rule used. For example, the LRP-0 rule calculates R_j according to Equation 3.8 and is equivalent to Gradients x Inputs when applied throughout the whole network. Other rules, such as LRP-γ or LRP-ϵ, exist (Montavon et al., 2019). LRP, in combination with the LRP-ϵ rule, is a global attribution method (Ancona et al., 2017).

$$R_j = \sum_k \frac{z_{jk}}{\sum_j z_{jk}} \tag{3.7}$$

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} \tag{3.8}$$

Proposed Method

Chefer et al. (2020) modify the relevance propagation rule of LRP to consider only elements with positive weighted relevance. They define relevance



$\mathbf{C} = \bar{\mathbf{A}}^{(1)} \cdot \bar{\mathbf{A}}^{(2)} \cdot \ldots \cdot \bar{\mathbf{A}}^{(B)}$
Input
Transformer block
Gradients
Relevance
Transformer block
Output
Attention
$\bar{\mathbf{A}}^{(B)} = \mathbb{E}_h\left(\nabla\mathbf{A}^{(B)} \odot R^{(n_B)}\right)^+ + I$
$\bar{\mathbf{A}}^{(1)}$

Figure 3.4: A high-level schema of the method proposed by Chefer et al. Image from Chefer et al. (2020).

propagation rules for self-attention and residual connections, where the operands are two feature maps instead of a feature map and a weight vector. They propagate the relevance from the output toward the input with gradients. Refer to the original paper for more details about the relevance propagation rules.

More formally, for each Transformer block b, a weighed attention relevance matrix $\bar{A}^{(b)}$ is calculated according to Equation 3.9, where I facilitates the residual connection, $\mathbf{E}_h$ is a mean across the attention heads, $\nabla A^{(b)}$ are gradients of the target class w.r.t. the attention map $A^{(b)}$, and R^{n_b} is the relevance of the attention map $A^{(b)}$. They then calculate the output C of the method using Equation 3.10. Refer to Figure 3.4 for a visual overview of the process. C has $s \times s$ dimensions, where s is the input sequence length. Each row of C corresponds to a relevance map of a token given the other tokens - similar in structure to attention maps.

$$\bar{A}^{(b)} = I + \mathbf{E}_h(\nabla A^{(b)} \odot R^{n_b})^+ \tag{3.9}$$

$$C = \bar{A}^{(1)} \cdot \bar{A}^{(2)} \cdot \ldots \cdot \bar{A}^{(b)} \tag{3.10}$$

Chefer et al. (2020) compare the proposed method to multiple attribution methods (including Raw Attention and Attention Rollout), showing that the proposed method outperforms the others.

3.4 SHAP Methods

Lundberg and Lee (2017) introduce Shapley Additive Explanations (SHAP) as a unified framework for interpreting model predictions.

SHAP uses a game-theoretic approach. It is closely related to Shapley values (Shapley, 1951). This cooperative game theory concept aims to allocate credit fairly to a group of players in a game based on their contribution to the result. A player's Shapley value is the player's average marginal contribution to all possible coalitions, accounting for the different ways to form a coalition. In the context of attributions and interpretability, the credit is the model prediction, and the players are input features.

The authors first define *additive feature attribution methods*, a new class of attribution methods. An additive feature attribution method has to have an *explanation model* that is a linear function of binary variables. An explanation model is any model that is an interpretable approximation of the original model. Multiple existing methods fall into this class of attribution methods, including Local Interpretable Model Explanations (LIME), DeepLIFT, and LRP. The authors show that the three methods use equations from cooperative game theory to generate the explanations, which ties them to Shapley value estimation (Lundberg and Lee, 2017).

Additive feature attribution methods have three desirable properties. The authors propose *SHAP Values* as a measure of importance. SHAP values are based on Shapley values and are designed to adhere to the desirable properties, be compatible with the equations mentioned above, and allow connections to LIME, LRP, and DeepLIFT. Lastly, they propose modifications to existing methods, which prevent them from violating the desirable properties (Lundberg and Lee, 2017).

3.4.1 KernelSHAP

KernelSHAP is a modification of the LIME (Ribeiro et al., 2016) attribution method proposed by Lundberg and Lee (2017).

LIME

LIME is a model-agnostic attribution method. It approximates the behavior of a complex model in a small area around the target input by constructing an interpretable model (e.g., a linear regression model). LIME is defined with Equation 3.11, where $\xi(x)$ is an explanation for input x, g is an interpretable approximation of the complex model f, L is a measure of how faithful g is to f, $\pi_x(z)$ is a weighting function, and $\Omega(g)$ is a measure of complexity of g.

The weighting function π_x is used to define the locality of a sample z around the target input x. When a sample z is local to x, π_x assigns it a higher weight. The authors suggest using distant and local samples, noting that the weighting makes the method more resistant to noise. The faithfulness measure L can be understood as a loss function.

$$\xi(x) = \underset{g \in G}{argmin}\ L(f, g, \pi_x) + \Omega(g) \tag{3.11}$$

The authors then focus on sparse linear attributions using a linear regression model $g(x) = w_x x$. As L, they use a locally weighted square loss and an exponential weighting kernel as π_x.

A dataset is required to train the linear model. The dataset is constructed by creating perturbations z of the target input x and using $f(z)$ as labels. The nature of the perturbations depends on the data. Often, parts of the input are replaced. The perturbed samples are weighted with the weighting function π_x. After training the linear regression model, the weights w_x are the attributions.

KernelSHAP

Like the authors of LIME, Lundberg and Lee (2017) use a linear regression model. However, they change the loss function L (Equation 3.14), the weighting function π_x (Equation 3.13), and the complexity measure $\Omega(g)$ (Equation 3.12), where M is the number of simplified input features, $z \in \{0, 1\}^M$, and h_x is function that maps a simplified input z to the original input space. For example, given a vector $x' \in \{0, 1\}^M$ where 0 signifies the absence of a token and 1 the presence of a token, $h(x')$ converts this binary vector into a vector of token ids, where the absent tokens are replaced by padding tokens. The authors prove that with this choice of L, π_x, and $\Omega(g)$, LIME recovers Shapley values.

$$\Omega(g) = 0 \tag{3.12}$$

$$\pi_x(z) = \frac{(M-1)}{(M\ choose\ |z|)|z|(M - |z|)} \tag{3.13}$$

$$L(f, g, \pi) = \sum_{z \in Z} [f(h_x(z)) - g(z)]^2 \pi_x(z) \tag{3.14}$$

3.4.2 DeepSHAP

DeepSHAP is a modification of DeepLIFT, proposed by Lundberg and Lee (2017).

DeepLift

DeepLIFT is a global (Ancona et al., 2017) model-specific attribution method based on relevance propagation. Ancona et al. (2017) show that DeepLIFT is equivalent to LRP-ϵ if some conditions are satisfied.

DeepLIFT uses a neutral reference input to calculate a *difference-from-reference* for each neuron. The difference-from-reference for a neuron is defined as $\Delta t = t - t^0$, where t is the neuron's activation for a target input, and t^0 the activation for a reference input. Let $x_1, x_2, ..., x_n$ be a set of neurons necessary and sufficient to calculate t. Then, we define the contribution score $C_{\Delta x_i \Delta t}$ according to Equation 3.15. The $C_{\Delta x_i \Delta t}$ is the amount of difference-from-reference in t that can be attributed to the difference-from-reference of x_i.

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta t \tag{3.15}$$

The authors then define *multipliers*. Let x be an input neuron with a difference-from-reference Δx, and t be a target neuron with a difference-from-reference Δt. We define the multiplier $m_{\Delta x \Delta t}$ according to Equation 3.16. The multiplier can be thought of as the contribution of Δx to Δt divided by Δx.

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \tag{3.16}$$

With multipliers, a way to propagate them through the network can be defined. Let $x_1, x_2, ..., x_n$ be the neurons of an input layer, $y_1, y_2, ..., y_n$ be the neurons of a hidden layer, and t be an output neuron. If we have multipliers $m_{\Delta x_i \Delta y_j}$ and $m_{\Delta y_j \Delta t}$, we define $m_{\Delta x_i \Delta t}$ as Equation 3.17. The authors refer to this equation as chain rule for multipliers and note the similarity with backpropagation.

$$m_{\Delta x_i \Delta t} = m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta t} \tag{3.17}$$

Now, given a neuron and its immediate inputs, we need to calculate the contribution scores for each of the immediate inputs. Three rules for assigning contributions are presented - the Linear rule, the Rescale rule, and the RevealCancel rule. The authors also differentiate between positive and negative contributions, treating them separately.

The Linear rule applies to dense and convolutional layers. The Rescale rule applies to nonlinearities with a single input neuron, for example, activation functions. The RevealCancel rule makes use of separate negative and positive contributions, alleviating issues with positive and negative contributions canceling each other out.

DeepSHAP

Lundberg and Lee (2017) note that DeepLIFT approximates SHAP values, assuming the model is linear and the input features are independent. The rules defined by Shrikumar et al. (2016) linearize the non-linear components of the network. DeepSHAP modifies the DeepLIFT multipliers in a way that allows it to compute SHAP values for single components and combine them into SHAP values for the whole network.

4 Evaluating Attributions

As the number of proposed and actively used attribution methods increased, a need to compare their performance came. This increase led to the proposal of benchmarks to standardize the evaluation of attribution methods in specific domains (DeYoung et al., 2019). Today, a complex field focused on evaluating attributions and attribution methods exists.

Nauta et al. (2022) publish a comprehensive review of existing evaluation methods. They define a set of twelve *explanation quality properties*, called *Co-12*, that are further segmented based on their most significant dimension - content, presentation, and user. The authors differentiate between multiple explanation types, e.g., feature importance, heatmap, white-box model, or decision trees, and provide a comprehensive list of categories of evaluation methods applied to the twelve explanation quality properties.

In the context of this thesis, only evaluation methods applicable to feature importance attributions are relevant. These methods include perturbations to features identified as important to the decision to see if the model prediction changes appropriately (Single Deletion, Incremental Deletion), seeing how much the attributions change when applied to a different target or class (Target Sensitivity), or comparing the attributions to a 'ground truth' (Alignment with Domain Knowledge). These methods fall into the *correctness*, *output-completeness*, and *coherence* categories of Co-12.

Mohseni et al. (2018) publish a survey focusing on the design and evaluation of explainable systems. They distinguish between design goals and evaluation metrics. Additionally, they segment these according to the end user - AI novice, data expert, and AI expert. The authors note that evaluation metrics and design goals are not exclusive to their end-user group and are more of an organizational convenience.

They further divide evaluation measures into computational measures and human-grounded measures. Human-grounded measures are primarily based on human-AI interactions, such as helping end users understand AI models or measuring how useful and trustworthy the explanations are from the end user's view. Relying on users to evaluate attributions can lead to skewed results because users prefer simple explanations. Computational measures do not rely on users to perform the evaluations. They encompass a wide range of different approaches to evaluating attributions. They include comparisons to other state-of-the-art attribution methods, correlation with existing methods, comparisons to white-box models, or comparisons to a 'ground truth.'

In the context of this thesis, we do not consider human-grounded measures.

Moving forward, we focus on evaluation methods that fall into the categories of correctness, output-completeness, and coherence as defined by Nauta et al. (2022). We focus mainly on ground-truth evaluations and provide background for other evaluation methods.

The authors of ERASER (DeYoung et al., 2019), who focus on agreement with human rationales as well as the faithfulness of the model, took an analogous approach. Agreement with human rationales (annotations) corresponds to a ground-truth evaluation method. Faithfulness measures how much the attributed features influenced the decision, corresponding to the categories of correctness and output-completeness as defined by Nauta et al. (2022).

It is also notable that some methods relied primarily on anecdotal evidence (e.g., attribution visualizations) as a justification for their effectiveness (along with, for example, theoretical properties) (Smilkov et al., 2017; Sundararajan et al., 2017; Simonyan et al., 2014).

4.1 Ground-Truth Evaluation Methods

Ground-truth evaluation methods rely on annotated examples to compare the attributions. Higher agreement of the attributions with the annotated examples results in a better score.

4.1.1 Intersection Over Union

The authors of ERASER use Intersection Over Union (IOU) to evaluate attributions. IOU allows for partial matches to count toward the score. Given an annotation span a and an attribution span b, IOU is the size of the intersection of a and b over their union. The authors also use a threshold that defines how much overlap must a and b have to count as an intersection. Partial matches can then be used to calculate token-level F1 scores (DeYoung et al., 2019).

ERASER IOU requires 'hard rationales' - excerpts supporting the ground truth. As many attribution methods provide 'soft scores' (e.g., SHAP values), they must be converted into hard rationales to use the ERASER IOU.

Chefer et al. (2020) use part of ERASER to evaluate their proposed method. They use the ERASER token-level F1 score and only consider top-k attributed tokens as a part of the explanation. That deals with the problem of selecting which attributions support the ground truth. Chefer et al. also mention the possibility of thresholding, noting that it may put some attribution methods at a disadvantage.

4.1.2 Soft Scoring

The authors of ERASER also consider soft scores and use the Area Under the Precision-Recall Curve (AUPRC) metric to evaluate them. AUPRC is calculated by moving a threshold over the token attributions, but the authors do not provide a detailed description in their paper (DeYoung et al., 2019).

4.2 Faithfulness Evaluation Methods

An attribution method may provide attributions that align with ground-truth annotations, but that does not necessarily mean the model relied on the highly-attributed elements to make the decision. Faithfulness evaluation methods measure the reliance of the model on highly-attributed elements. These methods often perturb or remove the highly-attributed elements of the input and measure how much it affects the prediction (Bach et al., 2015; Chefer et al., 2020; Ancona et al., 2017; DeYoung et al., 2019; Liu et al., 2022, 2021; Tan, 2022; Erion et al., 2021).

4.2.1 Comprehensivness and Sufficiency

ERASER (DeYoung et al., 2019) defines a measure of comprehensiveness by using contrast examples. Given an example x_i, they construct a contrast example $\tilde{x}_i$ by removing the elements supporting the decision (highly-attributed) a_i from x_i. Comprehensiveness is then equal to the difference between the model prediction $m(x_i)$ and $m(\tilde{x}_i)$. Intuitively, if the highly-attributed elements are important to the decision, their removal should decrease confidence in the prediction.

ERASER also defines a measure of sufficiency. Sufficiency measures how sufficient are the highly attributed elements for the model to make a decision. Sufficiency is the difference between the model prediction $m(x_i)$ and $m(a_i)$.

Both of these measures apply to soft scores through top-k_d discretization, where k_d is specific to a dataset and is equal to the average length of the ground-truth annotations. The top-k_d attributions are then used to calculate comprehensiveness and sufficiency.

Area Over the Pertubation Curve

The authors of ERASER (DeYoung et al., 2019) also define the Area Over the Pertubation Curve (AOPC) by using comprehensiveness and sufficiency as a function of k. ERASER has multiple datasets with varying document and annotation lengths. To make the scores comparable, they use bins. The bins define how many tokens get deleted. The bins correspond to the top 1%, 5%, 10%, 20% and 50% of highest-attributed tokens (meaning $k = 5$). For comprehensiveness, the aggregate measure is defined as Equation 4.1, and the same measure for sufficiency is defined analogously. $\tilde{x}_{ik}$ is a contrasting example with tokens up to and including bin k removed.

$$\frac{1}{|B|+1}\sum_{k=0}^{|B|} m(x_i) - m(\tilde{x}_{ik}) \tag{4.1}$$

Decision Flip

Serrano and Smith (2019) investigate how much of an impact zeroing attention values has on the prediction using a 'decision flip' measure. The core idea is that zeroing the highest attention value should have a larger impact on the prediction, while zeroing a random attention value should have a smaller impact on the prediction.

Area Under Curve on Threshold Performance

Liu et al. (2022) use the Area Under Curve on Threshold Performance (AUC-TP) alongside the ERASER comprehensiveness and sufficiency metrics. AUC-TP calculates an AUC score based on the change in model performance using feature replacement of top-k% attributed tokens.

Incremental Deletion

Ancona et al. (2017) evaluate multiple methods by incrementally removing the highest-attributed pixel. If the attributions are faithful, then the model performance should decrease. They also perform the same test but remove the lowest-attributed pixel, which allows them to verify that the attributions have the correct sign.

4.3 ERASER Benchmark

ERASER (DeYoung et al., 2019) is an NLP-focused explainability benchmark for evaluating attribution methods. It measures alignment with human annotations as well as the faithfulness of the attributions.

The benchmark comprises multiple datasets and tasks from document classification to fact extraction. The datasets vary in size. Alongside the datasets and annotations, the authors also propose multiple measures.

The benchmark is suited for evaluating soft attributions (e.g., gradients) and hard attributions (e.g., text spans). Chefer et al. (2020) use the Movie Reviews dataset from ERASER to evaluate the performance of their proposed method. They note that other datasets from ERASER contain documents too long to be processed by a BERT model in a single forward pass. The authors of ERASER use BERT models to create contextual embeddings, which are then processed using a recurrent network.

5 Experimental Setup

We select multiple attribution methods and two datasets for evaluation. In this section, we describe the datasets, attribution methods we use, hyper-parameters of the methods, and models we use and justify our choices.

5.1 Datasets

We choose to use two publicly available datasets to evaluate the attribution methods. We considered using ERASER but ultimately decided against using it. As noted by Chefer et al., the potentially long documents pose a problem for processing them with standard BERT-like models, which usually have a circa 512-token limit (Chefer et al., 2020; DeYoung et al., 2019). Processing longer documents would mean deciding how to split the documents (e.g., overlap or no overlap) and how to process the attributions. The authors of ERASER solve this problem by using a BERT model to produce context embeddings that they concatenate for longer documents and process with an LSTM network instead of a standard classification head. They start encoding a new sequence each time they reach the 512-token limit, hoping the LSTM network learns to compensate for it. For the BoolQ and Evidence Inference datasets, the authors of ERASER use GloVe embeddings instead of BERT embeddings (DeYoung et al., 2019).

Given the document length issues, the only ERASER dataset viable for us is the Movie Reviews dataset, a sentiment classification task. Instead, we decide to use the Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013). The SST dataset has the advantage of having more test samples (circa 2000 compared to 200). The training set of the SST dataset is more extensive, allowing for more thorough training. The SST dataset has out-of-the-box annotations we can use as ground truth. Additionally, we can use similar metrics for both of our datasets. Our second dataset does not have annotated text spans that the ERASER Movie Reviews dataset has and would thus require a different metric.

5.1.1 Stanford Sentiment Treebank

Stanford Sentiment Treebank (SST) (Socher et al., 2013) is a sentiment classification dataset. It consists of fine-grained parse trees with human-annotated sentiments (See Figure 5.1). Sentiments are represented as a

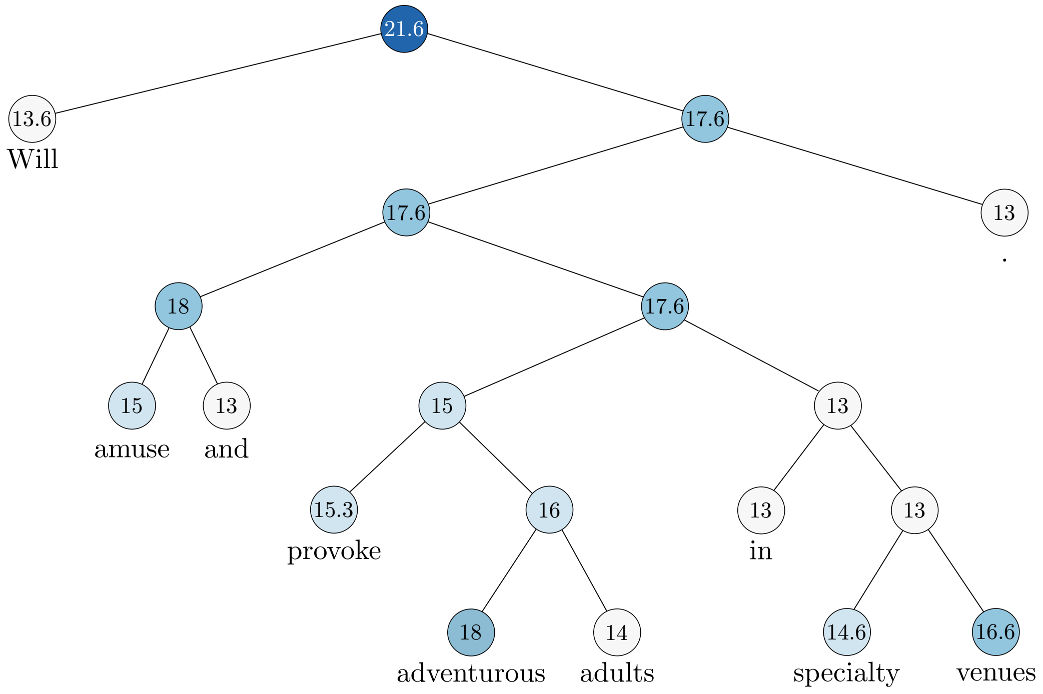


Figure 5.1: An example of an annotated sentence from the SST dataset. This is sentence number 222. The visualization is available on the dataset official website[1]. Note that the decimals are truncated. The 2/3 and 1/3 from the original visualization were replaced by .6 and .3 respectively.

continuous scale ranging from 0 to 1, where values closer to 0 indicate negative sentiment and those closer to 1 indicate positive sentiment. The dataset contains sentences from Rotten Tomatoes movie reviews and features train/dev/test splits.

Common variations of this dataset are SST-2, a binary classification dataset with only positive and negative labels, and SST-5, with five sentiment classes. SST-2 is also part of the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018).

We use the train and dev splits to train and evaluate our models. We then evaluate the attributions on the test split. The statistics of our SST dataset are in Table 5.1.

Preprocessing

We first remove all occurrences of neutral sentences from all three splits. We consider a sentence neutral if its sentiment is between 0.4 and 0.6, exclusive.

[1]https://nlp.stanford.edu/sentiment/treebank.html

Split	Samples	Positive samples	Negative samples	Tokens
Train	61 647	31 865	29 782	843 691
Dev	872	444	428	17 046
Test	1 821	909	912	35 023

Table 5.1: Statistics of different splits of our SST dataset.

Split	Samples	Class instances	Tokens
Train	11 955	30 524	2 882 120
Dev	2 538	5 249	708 897

Table 5.2: The statistics of the different splits in the CTDC dataset. Only the 37 classes we use for classification are included in the *Class instances* column.

We also replace the `-LRB-` and `-RRB-` character sequences with left and right round brackets.

We expand the training set from its original approximately 12 000 sentences using the phrase dictionary. SST includes a list of phrases (subsentence units) and their respective sentiment values. We add all phrases that are not neutral, not present in any test or dev split sentences, and not already present in the train split. GLUE benchmark applies a similar approach to their SST-2 dataset, which has a significantly larger train split than the original.

5.1.2 Czech Text Document Corpus

Czech Text Document Corpus (CTDC) (Kral and Lenc, 2018) is a multi-label document classification dataset in the Czech language. It consists of news documents from the Czech News Agency (CTK). We use the 2.0 version.

The dataset contains train and dev splits. The authors employ five-fold cross-validation on the train split, meaning no test split is necessary.

The documents vary in length, with the longest documents being more than 7 000 words long and the shortest documents being less than 50 words long. There are 60 classes in total. The authors use the 37 most frequent classes for classification. The authors do not specify which 37 classes they use, and they do not specify how they count the class frequencies. We select the 37 classes most frequently present in the train split. The statistics of our CTDC dataset are in Table 5.2.

	bert-base-cased	bert-medium	bert-small	bert-mini
Acc. (dev)	0.931 ± 0.004	0.909 ± 0.003	0.888 ± 0.003	0.856 ± 0.003
Acc. (test)	0.939 ± 0.002	0.917 ± 0.002	0.908 ± 0.001	0.865 ± 0.003

Table 5.3: The dev and test split accuracies of the models on the SST dataset. We report the mean and standard deviation for each model.

	bert-base-cased	bert-medium	bert-small	bert-mini
Parameters	110M	41.7M	29.1M	11.3M
Layers	12	8	4	4
Hidden size	768	512	512	256
Intermediate size	3072	2048	2048	1024
Attention heads	12	8	8	4

Table 5.4: A comparison between the different sizes of BERT models we fine-tune on the SST dataset.

5.2 Models

For each model, we train five instances. Unless stated otherwise, all metrics we report are a mean of the five instances and a standard deviation.

5.2.1 SST

We use the HuggingFace GLUE training script[1] to fine-tune our SST models. We run the training script with the default settings, excluding the learning rate. We fine-tune all models for three epochs with a learning rate of 1e-5.

For the models, we fine-tune the *bert-base-cased* model from HuggingFace, and smaller distilled BERT models. The distilled BERT models were trained by Turc et al. (2019) and ported to HuggingFace by Bhargava et al. (2021). We use the *medium*, *small*, and *mini* variants. The distilled models are uncased.

The accuracies of the models on the test and train splits are in Table 5.3. The comparison of model sizes is in Table 5.4.

[1]https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue.py

	Czert-B-base-cased	small-e-czech	mMiniLMv2-L6-H384
F1 (train)	0.874 ± 0.002	0.815 ± 0.003	0.750 ± 0.034
F1 (dev)	0.847 ± 0.004	0.790 ± 0.003	0.733 ± 0.028

Table 5.5: The micro F1 score of models fine-tuned on the CTDC dataset. F1 (train) is the aggregate micro F1 score obtained through five-fold cross-validation. F1 (dev) is the micro F1 score on the dev split. We report the mean and standard deviation for each model.

5.2.2 CTDC

For the CTDC dataset, we use three models. First, we use *Czert-B-base-cased* (Sido et al., 2021), a BERT model pre-trained on Czech data. While the authors report the performance on version 1.0 of the CTDC dataset, we use version 2.0, so the results are not comparable.

To evaluate small models, we use *small-e-czech* (Kocián et al., 2021), an ELECTRA model pre-trained on Czech data.

We also use a multilingual *MiniLMv2* (Wang et al., 2020) model, specifically the smaller variant with six layers and a hidden size of 384 (L6xH384). The model is distilled from a multilingual *XLM-Roberta-Large* (Conneau et al., 2019) model, which supports Czech.

We train each model with a different number of epochs and different learning rates. The hyperparameters are in Table 5.6 along with the sizes of the models.

Our training and validation procedure follows Kral and Lenc (2018). We perform five-fold cross-validation, first training the model on the four training folds and saving the predictions on the hold-out fold. After processing each fold as a hold-out fold, we calculate a micro F1 score on all the predictions (the whole training split). We then train the model on the entire training split. After tuning the hyperparameters, we also evaluate the models on the dev split. The results are in Table 5.5.

5.3 Evaluation

We evaluate only correct predictions. Because we use ground-truth evaluation methods as opposed to faithfulness evaluation methods, evaluating the attributions when the model makes the wrong decision could introduce noise to the overall results and could unfairly punish the attribution methods for the model's inaccuracy. We also do not evaluate predictions with a low level of certainty. We define a low level of certainty as a prediction with

	Czert-B-base-cased	small-e-czech	mMiniLMv2-L6-H384
Parameters	110M	14M	107M
Epochs	10	12	14
Learning rate	1e-5	1e-4	1e-4

Table 5.6: The training hyperparameters and sizes of the models that we fine-tune on the CTDC dataset. The *MiniLMv2* model is multilingual, and its word embeddings are large compared to the monolingual *Czert* and *small-e-czech*.

a probability < 0.6.

As each instance of a model will correctly predict different examples, different model instances are evaluated on slightly different sets of examples.

5.3.1 SST

The SST dataset comes with phrase-level sentiment annotations. Each sentence consists of word-level phrases. The SST dataset includes a list of word-level phrases for each sentence. For each of these phrases, we have a sentiment annotation. We thus treat the sentence as a sequence of word-level phrases with their respective sentiment annotations. The word-level phrase annotations are our ground truth, as we can infer how the individual words contribute to the overall sentiment of the sentence. We assume that the sentiment of the individual words is independent of the sentence. That is an obvious simplification that ignores linguistic structures such as negation. We believe that this simplification does not significantly impact the results. As the sentiment annotations are continuous, we can also rank the annotated words according to their sentiment.

We do not evaluate the methods on sentences with less than ten words because the words tend to have a neutral sentiment. Eliminating short sentences gives us more confidence that a high-enough number of words will have non-neutral sentiment.

We always center the annotations around 0 by subtracting 0.5 from them. How we interpret the annotations depends on the sentiment of the sentence. For example, when a sentence has a negative sentiment, the attributions contributing to the decision have positive values, but the negative sentiment annotations have negative values. As we need to align the attributions and the annotations, we multiply the annotations by -1. Then, a positive attribution will correspond to a positive annotation. This process is necessary only when the sentiment of the sentence is negative.

Top-K-Top-K Intersection

The metric we choose to use utilizes the ranking of attributions and the ranking of the ground-truth annotations. Given a sentence with token-level annotations t, token-level attributions a, and a function $top(k, x)$ that returns the indices of the k highest values from x, we define our metric according to Equation 5.1. We use $k = 1$, 2, and 5. As defined, this metric evaluates only one sentence. We evaluate each model on the entire test split of sentences and report the mean across all of the n sentences (Equation 5.2).

$$s_k = \frac{top(k, a) \cap top(k, t)}{k} \tag{5.1}$$

$$S_k = \frac{1}{n} \sum_{i=1}^{n} \frac{top(k, a_i) \cap top(k, t_i)}{k} \tag{5.2}$$

5.3.2 CTDC

The CTDC dataset does not contain information we can use as ground-truth annotations. It does, however, contain the classes for each document. These classes correspond to the categories of Czech News Agency articles. Each article is associated with a list of manually added keywords. We could use these keywords as ground-truth annotations, but we do not know how related a keyword is to a class. It may be the case that a keyword is similarly common across all classes.

We use *pointwise mutual information (PMI)* to determine how much a keyword is tied to a class. Given a class c and a keyword w, PMI (Equation 5.3) expresses how much more likely it is for class c and keyword w to appear together compared to if they were independent. A higher PMI means the keyword is more indicative of the class.

$$pmi(c; k) = \log_2 \frac{p(c, k)}{p(c)p(k)} = \log_2 \frac{p(c|k)}{p(c)} = \log_2 \frac{p(k|c)}{p(k)} \tag{5.3}$$

Since each document can belong to multiple classes, it can be associated with multiple sets of keywords. We treat each correctly predicted class of a document as a document-class pair. We call a document-class pair a *document instance*. As one document can belong to multiple classes, and each class has its own set of keywords, it can be evaluated for each class.

Because the dataset contains very long documents, we discard any documents that do not fit into the 512-token limit of our models. We also

Perc.	50	55	60	65	70	75	80	85	90	95
PMI	1.28	1.43	1.59	1.76	1.95	2.15	2.32	2.70	3.26	4.00

Table 5.7: The keywords PMI percentiles, from the 50th percentile. We ignore any keywords not associated with one of the 37 classes we use for classification. We also ignore keywords that consist of multiple words. The keywords are lowercased, stemmed, and duplicates removed.

discard documents shorter than 30 tokens. When a document instance does not contain any keywords specific to the evaluated class, we ignore it.

The keyword PMI distribution is in Table 5.7. We ignore any keywords not associated with one of the 37 classes we use for classification. We also ignore keywords that consist of multiple words (discarding about 16% of the keywords). We lowercase and stem[1] the keywords to account for inflections. The documents are lowercased and stemmed as well to search for keywords.

We set a minimum PMI of 2.0, meaning we only use keywords with an equal or higher PMI during the evaluation. A PMI of 2.0 is between the 70th and 75th percentile. Setting the minimum PMI to the 75th percentile would remove all keywords from a large class that we use for evaluation. We manually examined the keywords for that class, decided the keywords were of a high-enough quality, and lowered the minimum PMI to 2.0. A lower PMI threshold would introduce low-quality keywords (e.g., dates).

That leaves us with 6 730 class-keyword pairs with an average of 181.9 keywords per class. A graph with the number of associated keywords for each class can be seen in Figure 5.2. A histogram of unique keyword occurrences in document instances can be seen in Figure 5.3. On average, a document instance contains 6.08 unique keywords, and only 96 (2%) document instances do not contain any keywords.

Top-K-All Intersection

We treat all keywords equally. No distinction is made based on the PMI. We do this because while we are confident that the keywords are indicative of the class, we are not confident in the PMI being directly proportional to the importance of the keyword as a ground truth.

Negative attributions are not considered (set to zero). While the two classes in the SST dataset are mutually exclusive and evidence for one class serves as counter-evidence for the other, the same is not the case for the

[1]https://github.com/UFAL-DSG/alex/blob/master/alex/utils/czech_stemmer.py

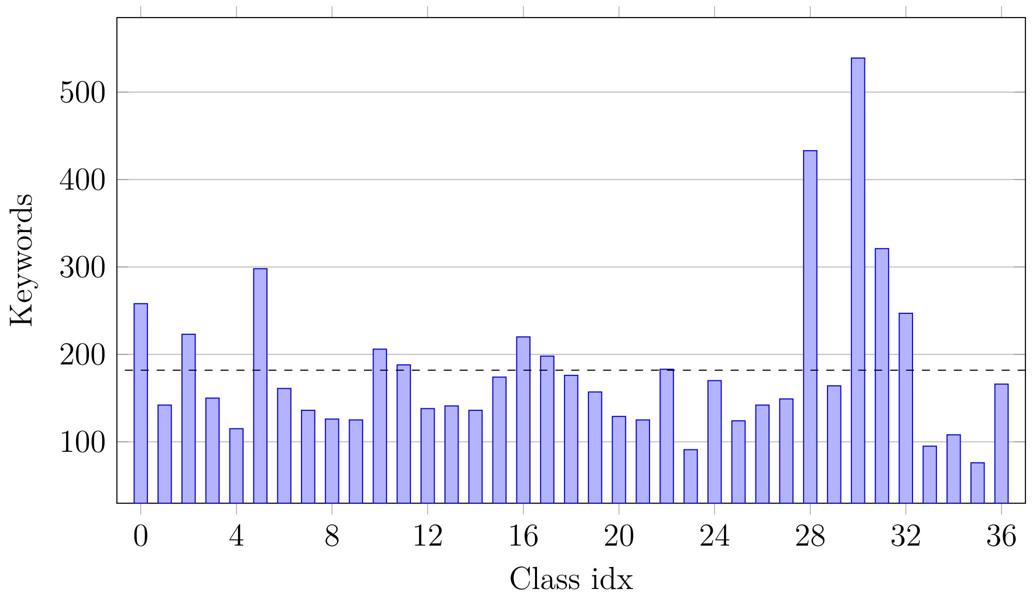


Figure 5.2: The number of associated keywords for each of the 37 classes. The dashed horizontal line represents the average.

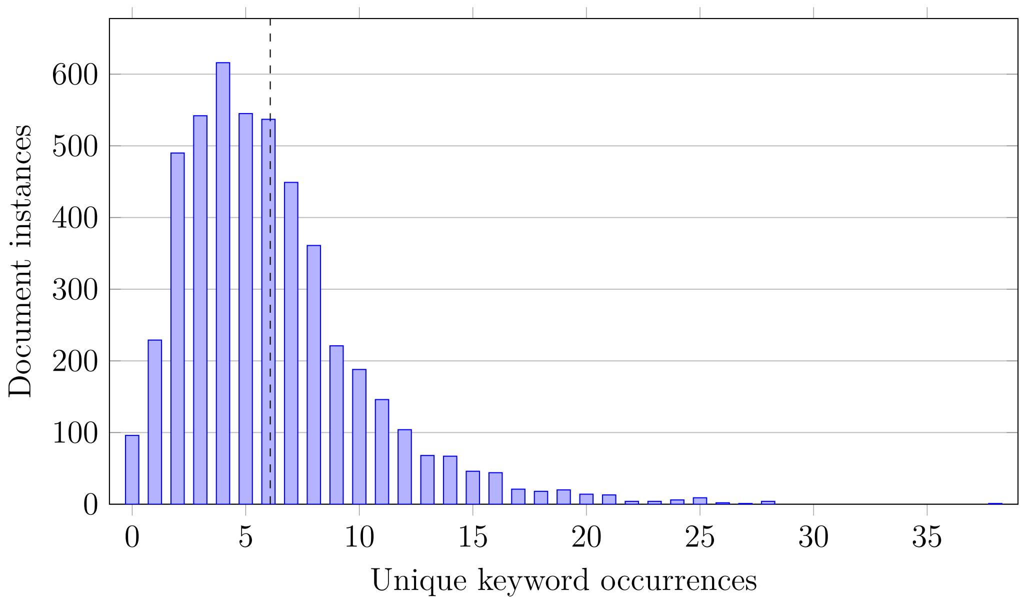


Figure 5.3: The histogram of unique keyword occurrences in document instances. The dashed vertical line represents the average.

CTDC dataset, which is a multi-label classification task. As such, we can ignore the negative attributions.

For the CTDC dataset, we adopt a relaxed metric. Given attributions a, a function $top(k, x)$ that returns the indices of the k highest values in x, and a list of positions of keywords in the document $keywords$, we define the metric according to Equation 5.4. We report the mean over all n valid document instances (Equation 5.5) and use $k = 5$, 10, and 15.

$$c_k = \frac{|top(k, a) \cap keywords|}{|keywords|} \tag{5.4}$$

$$C_k = \frac{1}{n} \sum_{i=1}^{n} \frac{|top(k, a_i) \cap keywords_i|}{|keywords_i|} \tag{5.5}$$

5.4 Attribution Methods

We choose multiple gradient-based methods, a Transformer-specific method from Chefer et al. (2020), and a SHAP-based method. Every method we use creates class-specific attributions. The generated attributions can be matrices of different shapes. The exact process for extracting the attributions is described separately for each method.

5.4.1 Gradients and Gradients x Input

We take the gradients of the target output with the softmax or sigmoid functions applied w.r.t. the word embeddings. The gradients are an $s \times e$ matrix, where s is the sequence length and e is the embedding dimension size. We take an average over the embedding dimension e, giving us a vector of length s containing an attribution value for each token. We then remove the first and last value, eliminating the `[CLS]` and `[SEP]` tokens.

5.4.2 SmoothGRAD

We perform SmoothGRAD experiments with $n = 20$, 50, and 100 samples. We obtain the gradients and process the attributions as described in Section 5.4.1. We follow Equations 3.1 and 3.2. We determine suitable noise levels for every model in Section 5.6.

5.4.3 Integrated Gradients

As with SmoothGRAD, we run Integrated Gradients experiments with steps $n = 20$, 50, and 100. To approximate the integral, we use the Trapezoidal rule. We obtain the gradients and process the attributions as described in Section 5.4.1. We determine suitable baselines for each model in Section 5.5.

5.4.4 Chefer et al.

To represent Transformer-specific methods, we choose the method proposed by Chefer et al. (2020). Chefer et al. compare their method to the Raw Attention and Attention Rollout. Their proposed method outperforms both the Raw Attention and Attention Rollout. Therefore, we do not include Raw Attention or any of its variants or Attention Rollout.

The method produces an $s \times s$ matrix of attributions, where s is the sequence length. We employ the same procedure as the authors and use the relevancies for the `[CLS]` token as attributions, which gives us a vector of s positive values, from which we remove values that correspond to the `[CLS]` and `[SEP]` tokens.

We use the reference implementation from the authors' GitHub repository[1]. Because the implementation supports only HuggingFace BERT models, we do not evaluate the method in combination with ELECTRA and RoBERTa models.

5.4.5 KernelSHAP

There are no clear guidelines about how many samples are appropriate. The authors of LIME, on which KernelSHAP is based, use $n = 15\ 000$ for their experiments. They, however, use computationally cheap methods such as Nearest Neighbours or Support Vector Machines (Ribeiro et al., 2016). Using 15 000 samples in our experiments is computationally too expensive for our networks.

We have found existing work that applies KernelSHAP on Transformer models and textual data (Remmer, 2022). They use the number of samples $n = 500$. We use $n = 100$, 200, and 500, using their choice of n as our upper limit. The perturbed samples are created by replacing the input features with a specified baseline. The choice of a baseline replacement for removed tokens is further discussed in Section 5.7. We make sure that the `[CLS]` and `[SEP]` tokens are not subject to replacement.

We use the Captum implementation of KernelSHAP[2] with the default settings, changing only the number of samples.

[1] https://github.com/hila-chefer/Transformer-Explainability

[2] https://captum.ai/api/kernel_shap.html

BERT-base			
method	top1	top3	top5
ig 50 zero	.293 ± .012	.356 ± .010	.444 ± .007
ig 50 pad	.255 ± .056	.314 ± .035	.396 ± .014
ig 50 avg	.186 ± .023	.300 ± .023	.411 ± .024
ig 50 custom	.104 ± .000	.237 ± .002	.360 ± .003
random	.056 ± .005	.195 ± .005	.329 ± .001

BERT-medium			
method	top1	top3	top5
ig 50 zero	.381 ± .012	.407 ± .009	.452 ± .008
ig 50 pad	.338 ± .007	.403 ± .012	.463 ± .018
ig 50 avg	.329 ± .038	.395 ± .023	.455 ± .013
ig 50 custom	.200 ± .011	.312 ± .007	.411 ± .004
random	.064 ± .005	.194 ± .005	.333 ± .003

Table 5.8: The Integrated Gradients baseline evaluation metrics for the *bert-base-cased* and *bert-medium* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

5.5 Integrated Gradients Baselines

The authors of Integrated Gradients recommend using all-zero embeddings as a baseline. According to the authors, an all-zero baseline works well because unimportant words tend to have embeddings with a small norm (Sundararajan et al., 2017). Their GitHub[1] repository also suggests using padding tokens as a baseline. For a binary classification problem, the authors' GitHub repository mentions using a sample with a sigmoid score of 0.5 as a neutral baseline.

We test the three baselines mentioned above and a baseline made of average embedding vectors on three trained instances of each model we use.

5.5.1 SST

For the SST dataset, we use an all-zero baseline, a baseline made of only padding tokens, a baseline made of average embedding vectors, and an artificially constructed input with probabilities around 0.5 ± 0.025. We then use $n = 50$ interpolation steps and evaluate the method on the SST dataset, as described in Section 5.3.1.

The results are in Tables 5.8 and 5.9. For each model, we underscore the baseline we use.

Based on the results, the all-zero baseline is a good choice for all tested models. The only case where the all-zero baseline is outperformed is the *bert-small* model, where the average vector baseline slightly outperforms it.

5.5.2 CTDC

For the CTDC dataset, we consider the same baselines as the SST dataset above (all-zero, average embedding, padding tokens, and custom). Because

[1]https://github.com/ankurtaly/Integrated-Gradients

BERT-small			
method	top1	top3	top5
ig 50 zero	.382 ± .004	.443 ± .009	.489 ± .010
ig 50 pad	.363 ± .006	.432 ± .002	.499 ± .004
ig 50 avg	.383 ± .010	.447 ± .006	.501 ± .009
ig 50 custom	.234 ± .008	.329 ± .007	.426 ± .004
random	.059 ± .002	.192 ± .002	.332 ± .002

BERT-mini			
method	top1	top3	top5
ig 50 zero	.416 ± .005	.482 ± .013	.520 ± .014
ig 50 pad	.407 ± .009	.457 ± .017	.506 ± .020
ig 50 avg	.365 ± .024	.407 ± .032	.468 ± .027
ig 50 custom	.292 ± .005	.378 ± .007	.459 ± .006
random	.065 ± .007	.202 ± .003	.339 ± .001

Table 5.9: The Integrated Gradients baseline evaluation metrics for the *bert-small* and *bert-mini* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

Czert			
method	top5	top10	top15
ig 50 zero	.251 ± .011	.352 ± .014	.424 ± .012
ig 50 pad	.233 ± .005	.335 ± .015	.409 ± .016
ig 50 avg	.235 ± .011	.335 ± .011	.405 ± .010
ig 50 custom	.238 ± .006	.347 ± .007	.421 ± .007
random	.058 ± .001	.125 ± .000	.189 ± .002

MiniLMv2			
method	top5	top10	top15
ig50 zero	.272 ± .007	.373 ± .007	.444 ± .007
ig50 pad	.297 ± .016	.404 ± .018	.475 ± .018
ig50 avg	.271 ± .016	.369 ± .017	.439 ± .019
ig50 custom	.260 ± .013	.364 ± .013	.437 ± .010
random	.102 ± .002	.182 ± .004	.250 ± .003

Table 5.10: The Integrated Gradients baseline evaluation metrics for the *Czert-B-base-cased* and *mMiniLMv2-L6-H384* models on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

the CTDC dataset is a muti-label classification problem, we construct the custom baseline differently. We artificially create samples with a very low probability ($< 1e - 2$) for each class.

The results are in Tables 5.10 and 5.11. For each model, we underscore the baseline we use.

As with the SST dataset, we observe that even though an all-zero baseline is not always the best choice, it is not significantly worse than the best choice.

small-e-czech			
method	top5	top10	top15
ig 50 zero	.269 ± .004	.366 ± .007	.431 ± .010
ig 50 pad	.286 ± .005	.382 ± .011	.445 ± .011
ig 50 avg	.254 ± .003	.352 ± .001	.421 ± .001
ig 50 custom	.229 ± .006	.321 ± .006	.382 ± .006
random	.054 ± .002	.119 ± .002	.179 ± .001

Table 5.11: The Integrated Gradients baseline evaluation metrics for the *small-e-czech* model on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

BERT-base			
method	top1	top3	top5
sg 50 0.05	.123 ± .004	.240 ± .004	.355 ± .001
sg 50 0.15	.058 ± .002	.204 ± .002	.332 ± .003
sg 50 0.25	.058 ± .001	.196 ± .004	.336 ± .006
sg 50 0.05 x I	.343 ± .010	.399 ± .005	.459 ± .003
sg 50 0.15 x I	.116 ± .007	.249 ± .015	.372 ± .011
sg 50 0.25 x I	.091 ± .015	.234 ± .012	.361 ± .012
random	.056 ± .005	.195 ± .005	.329 ± .001

BERT-medium			
method	top1	top3	top5
sg 50 0.05	.143 ± .006	.255 ± .001	.361 ± .002
sg 50 0.15	.070 ± .004	.209 ± .003	.340 ± .005
sg 50 0.25	.065 ± .003	.199 ± .001	.340 ± .001
sg 50 0.05 x I	.410 ± .010	.440 ± .005	.477 ± .003
sg 50 0.15 x I	.324 ± .007	.423 ± .007	.489 ± .004
sg 50 0.25 x I	.277 ± .003	.400 ± .007	.478 ± .005
random	.064 ± .005	.194 ± .005	.333 ± .003

Table 5.12: The SmoothGRAD noise level evaluation metrics for the *bert-base-cased* and *bert-medium* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

5.6 SmoothGRAD Noise Levels

To select the appropriate noise level for our SmoothGRAD experiments, we use the original SmoothGRAD article as a reference (Smilkov et al., 2017). The authors compare multiple noise levels on image-related tasks. They observe that a noise level between 0.1 and 0.2 produces the best results. Based on their observations, we experiment with noise levels $nl = 0.05$, 0.15, and 0.25. As with Integrated Gradients, we choose the number of samples $n = 50$.

We test the three noise levels on three trained instances of each model we use. We evaluate SmoothGRAD and SmoothGRAD x Input as two separate attribution methods.

5.6.1 SST

The results for the SST dataset are in Tables 5.12 and 5.13. For each model, we underscore the noise level we use.

We can see that high noise levels perform worse. A noise level of 0.05 provides the best performance, except for the *bert-mini* model, where SmoothGRAD x Input with a noise level of 0.15 offers better performance than the 0.05 noise level.

5.6.2 CTDC

The results for the CTDC dataset are in Tables 5.14 and 5.15. For each model, we underscore the noise level we use.

We can see that high noise levels perform worse. A noise level of 0.05 provides the best performance, except for the multilingual *MiniLMv2* and

BERT-small			
method	top1	top3	top5
sg 50 0.05	.127 ± .007	.245 ± .002	.362 ± .004
sg 50 0.15	.073 ± .006	.206 ± .003	.343 ± .003
sg 50 0.25	.064 ± .004	.208 ± .002	.338 ± .002
sg 50 0.05 x I	.424 ± .004	.472 ± .005	.512 ± .000
sg 50 0.15 x I	.354 ± .008	.443 ± .000	.502 ± .003
sg 50 0.25 x I	.299 ± .002	.406 ± .002	.480 ± .003
random	.059 ± .002	.192 ± .002	.332 ± .002

BERT-mini			
method	top1	top3	top5
sg 50 0.05	.152 ± .003	.277 ± .001	.386 ± .001
sg 50 0.15	.105 ± .009	.234 ± .002	.359 ± .001
sg 50 0.25	.077 ± .007	.217 ± .001	.351 ± .002
sg 50 0.05 x I	.406 ± .019	.439 ± .022	.480 ± .019
sg 50 0.15 x I	.425 ± .012	.483 ± .014	.521 ± .014
sg 50 0.25 x I	.385 ± .016	.462 ± .016	.512 ± .014
random	.065 ± .007	.202 ± .003	.339 ± .001

Table 5.13: The SmoothGRAD noise level evaluation metrics for the *bert-small* and *bert-mini* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

Czert			
method	top5	top10	top15
sg 50 0.05	.155 ± .008	.240 ± .009	.304 ± .008
sg 50 0.15	.077 ± .010	.147 ± .016	.211 ± .015
sg 50 0.25	.059 ± .005	.127 ± .011	.189 ± .010
sg 50 x I 0.05	.264 ± .004	.366 ± .001	.432 ± .003
sg 50 x I 0.15	.206 ± .017	.329 ± .020	.416 ± .017
sg 50 x I 0.25	.182 ± .016	.300 ± .021	.386 ± .023
random	.058 ± .001	.125 ± .000	.189 ± .002

MiniLMv2			
method	top5	top10	top15
sg50 0.05	.188 ± .004	.275 ± .006	.341 ± .003
sg50 0.15	.144 ± .003	.227 ± .003	.299 ± .002
sg50 0.25	.118 ± .007	.201 ± .004	.268 ± .004
sg50 0.05 x I	.295 ± .010	.398 ± .011	.466 ± .010
sg50 0.15 x I	.301 ± .005	.404 ± .004	.477 ± .003
sg50 0.25 x I	.263 ± .004	.364 ± .006	.438 ± .007
random	.102 ± .002	.182 ± .004	.250 ± .003

Table 5.14: The SmoothGRAD noise level evaluation metrics for the *Czert-B-base-cased* and *mMiniLMv2-L6-H384* models on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

small-e-czech, which benefit from a higher noise level when multiplying SmoothGRAD attributions with input.

5.7 KernelSHAP Baselines

KernelSHAP perturbs the input by replacing input features with baseline features. In our experiments, we use the input tokens as input features. Once again, there are no clear guidelines about an appropriate baseline choice, and we refer to existing work (Remmer, 2022). They use the `[MASK]` token as a baseline but mention experimenting with `[UNK]` and `[PAD]` tokens.

We evaluate all three options with $n = 200$ samples on three trained instances of each model we use.

small-e-czech			
method	top1	top3	top5
sg 50 0.05	.174 ± .000	.249 ± .004	.305 ± .004
sg 50 0.15	.100 ± .003	.171 ± .007	.232 ± .006
sg 50 0.25	.066 ± .006	.133 ± .004	.194 ± .003
sg 50 0.05 x I	.215 ± .008	.308 ± .011	.375 ± .011
sg50 0.15 x I	.228 ± .006	.332 ± .002	.406 ± .002
sg 50 0.25 x I	.178 ± .008	.281 ± .009	.357 ± .011
random	.054 ± .002	.119 ± .002	.179 ± .001

Table 5.15: The SmoothGRAD noise level evaluation metrics for the *small-e-czech* model on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

BERT-base			
method	top1	top3	top5
ks 200 pad	.373 ± .005	.386 ± .003	.448 ± .001
ks 200 mask	.357 ± .011	.392 ± .011	.449 ± .008
ks 200 unk	.337 ± .007	.413 ± .006	.475 ± .003
random	.056 ± .005	.195 ± .005	.329 ± .001

BERT-medium			
method	top1	top3	top5
ks 200 pad	.337 ± .003	.375 ± .007	.440 ± .006
ks 200 mask	.374 ± .008	.427 ± .002	.468 ± .001
ks 200 unk	.360 ± .009	.423 ± .000	.477 ± .004
random	.064 ± .005	.194 ± .005	.333 ± .003

Table 5.16: The KernelSHAP baseline evaluation metrics for the *bert-base-cased* and *bert-medium* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

5.7.1 SST

The results for the SST dataset are in Tables 5.16 and 5.17. We do not see any discernible pattern in the results. It seems that different models require different baselines. We can, however, observe that the `[MASK]` token performs reasonably well in all cases, even if sometimes noticeably worse than the best baseline. For each model, we underscore the baseline we use.

BERT-small			
method	top1	top3	top5
ks 200 pad	.343 ± .010	.422 ± .004	.480 ± .004
ks 200 mask	.341 ± .014	.441 ± .005	.496 ± .001
ks 200 unk	.363 ± .004	.440 ± .008	.490 ± .005
random	.059 ± .002	.192 ± .002	.332 ± .002

BERT-mini			
method	top1	top3	top5
ks 200 pad	.413 ± .016	.482 ± .007	.514 ± .010
ks 200 mask	.398 ± .006	.480 ± .005	.517 ± .001
ks 200 unk	.409 ± .009	.449 ± .008	.491 ± .007
random	.065 ± .007	.202 ± .003	.339 ± .001

Table 5.17: The KernelSHAP baseline evaluation metrics for the *bert-small* and *bert-mini* models on the test split of the SST dataset. The metrics are averages from three models and a standard deviation.

Czert			
method	top5	top10	top15
ks200 pad	0.105 ± 0.003	0.178 ± 0.005	0.241 ± 0.006
ks200 unk	0.091 ± 0.004	0.168 ± 0.002	0.230 ± 0.002
ks200 mask	0.115 ± 0.004	0.191 ± 0.005	0.254 ± 0.007
random	0.058 ± 0.001	0.125 ± 0.000	0.189 ± 0.002

MiniLMv2			
method	top5	top10	top15
ks200 pad	0.142 ± 0.004	0.222 ± 0.005	0.290 ± 0.004
ks200 unk	0.133 ± 0.006	0.211 ± 0.004	0.280 ± 0.004
ks200 mask	0.154 ± 0.008	0.232 ± 0.007	0.298 ± 0.002
random	0.102 ± 0.002	0.182 ± 0.004	0.250 ± 0.003

Table 5.18: The KernelSHAP baseline evaluation metrics for the *Czert-B-base-cased* and *mMiniLMv2-L6-H384* models on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

small-e-czech			
method	top5	top10	top15
ks200 pad	0.111 ± 0.007	0.181 ± 0.003	0.238 ± 0.002
ks200 unk	0.096 ± 0.008	0.165 ± 0.005	0.226 ± 0.007
ks200 mask	0.109 ± 0.004	0.179 ± 0.004	0.238 ± 0.002
random	0.054 ± 0.002	0.119 ± 0.002	0.179 ± 0.001

Table 5.19: The KernelSHAP baseline evaluation metrics for the *small-e-czech* model on the dev split of the CTDC dataset. The metrics are averages from three models and a standard deviation.

5.7.2 CTDC

The results for the CTDC dataset are in Tables 5.18 and 5.11. Once again, we observe no discernible patterns in the results. The `[UNK]` token is the worst-performing baseline. The `[MASK]` token outperforms the `[PAD]` token in two out of three cases. The results are very close when it does not outperform the `[PAD]` token. For each model, we underscore the baseline we use.

6 Achieved Results

The evaluation metrics of the models fine-tuned on the SST dataset are in Tables 6.1 and 6.2. The evaluation metrics of the models fine-tuned on the CTDC dataset are in Tables 6.3 and 6.4. We discuss the results in Section 6.1.

The models we fine-tuned on the SST dataset made few low-confidence predictions. The models we fine-tuned on the CTDC dataset made more low-confidence predictions than the SST dataset models. On average, the *Czert* models made low-confidence predictions in 19% of cases, the *MiniLMv2* models in 32% of cases, and the *small-e-czech* models in 23% of cases.

6.1 Discussion

6.1.1 SmoothGRAD

With SmoothGRAD, we see similar results with both our datasets. Pure SmoothGRAD shows a worse performance when compared to Vanilla Gradients. The cause is the design of our datasets and metrics. As the authors note (Smilkov et al., 2017), SmoothGRAD effectively applies a Gaussian kernel on the gradients, smoothing them out. Smoothing removes the sharp gradients and evens out their distribution. Our metrics, however, reward sharp and accurate attributions, which causes the lackluster performance of SmoothGRAD.

Multiplying the SmoothGRAD attributions with input brings a significant improvement. We believe the cause of this improvement is the word embeddings of important words having larger norms Sundararajan et al. (2017). When we multiply the smoothed-out gradients with the input embeddings, we effectively sharpen the attributions.

On the SST dataset, increasing the number of samples from 20 to 50 brings a considerable improvement. In contrast, the increase from 50 to 100 brings a less significant improvement (Tables 6.1 and 6.2). The authors of ShoothGRAD observe similar behavior (Smilkov et al., 2017), where increasing the number of samples beyond 50 brings diminishing returns. We can see that with the small- and mini-size BERT models, SmoothGRAD x Input is outperformed by Integrated Gradients when using 20 samples, but increasing the number of samples improves the performance of SmoothGRAD x Input. Integrated Gradients, however, see no further improvement with an

BERT-base			
method	top1	top3	top5
grads	.145 ± .006	.267 ± .004	.369 ± .003
grads x I	.118 ± .013	.236 ± .011	.353 ± .008
ig 20	.251 ± .022	.333 ± .008	.427 ± .012
ig 50	.282 ± .018	.353 ± .009	.438 ± .008
ig 100	.290 ± .018	.359 ± .008	.443 ± .009
sg 20	.125 ± .005	.236 ± .006	.354 ± .002
sg 50	.118 ± .006	.237 ± .005	.355 ± .003
sg 100	.127 ± .005	.239 ± .003	.355 ± .004
sg 20 x I	.330 ± .015	.384 ± .007	.450 ± .003
sg 50 x I	.341 ± .012	.395 ± .008	.459 ± .004
sg 100 x I	.362 ± .010	.405 ± .004	.468 ± .005
ks 100	.327 ± .012	.354 ± .010	.423 ± .007
ks 200	.367 ± .010	.388 ± .011	.451 ± .007
ks 500	.403 ± .011	.427 ± .010	.473 ± .007
Chefer et al.	.265 ± .021	.360 ± .008	.450 ± .007
random	.057 ± .004	.194 ± .004	.328 ± .002

BERT-medium			
method	top1	top3	top5
grads	.171 ± .006	.278 ± .004	.378 ± .004
grads x I	.160 ± .009	.266 ± .005	.371 ± .005
ig 20	.384 ± .016	.412 ± .009	.455 ± .008
ig 50	.386 ± .015	.413 ± .011	.455 ± .008
ig 100	.387 ± .015	.413 ± .010	.456 ± .008
sg 20	.137 ± .009	.251 ± .002	.363 ± .002
sg 50	.140 ± .006	.253 ± .003	.363 ± .003
sg 100	.143 ± .009	.257 ± .004	.364 ± .002
sg 20 x I	.393 ± .008	.428 ± .003	.467 ± .003
sg 50 x I	.411 ± .010	.444 ± .006	.478 ± .004
sg 100 x I	.422 ± .007	.455 ± .004	.485 ± .004
ks 100	.334 ± .009	.384 ± .004	.442 ± .004
ks 200	.367 ± .006	.424 ± .004	.471 ± .003
ks 500	.407 ± .008	.468 ± .006	.498 ± .004
Chefer et al.	.339 ± .023	.391 ± .016	.457 ± .011
random	.066 ± .005	.193 ± .004	.331 ± .003

Table 6.1: The evaluation metrics for the *bert-base-cased* and *bert-medium* models on the test split of the SST dataset. The metrics are averages from five models and a standard deviation.

BERT-small			
method	top1	top3	top5
grads	.166 ± .004	.275 ± .004	.380 ± .002
grads x I	.210 ± .021	.310 ± .013	.399 ± .010
ig 20	.380 ± .010	.445 ± .006	.500 ± .008
ig 50	.380 ± .010	.445 ± .006	.500 ± .008
ig 100	.380 ± .010	.445 ± .006	.500 ± .008
sg 20	.119 ± .004	.242 ± .003	.362 ± .002
sg 50	.125 ± .004	.241 ± .003	.359 ± .006
sg 100	.128 ± .001	.245 ± .006	.363 ± .004
sg 20 x I	.396 ± .007	.445 ± .003	.491 ± .005
sg 50 x I	.428 ± .003	.470 ± .003	.506 ± .002
sg 100 x I	.439 ± .007	.483 ± .005	.518 ± .003
ks 100	.322 ± .009	.393 ± .007	.457 ± .003
ks 200	.364 ± .010	.439 ± .007	.488 ± .005
ks 500	.394 ± .007	.470 ± .011	.514 ± .006
Chefer et al.	.378 ± .012	.435 ± .006	.489 ± .006
random	.060 ± .003	.193 ± .002	.333 ± .002

BERT-mini			
method	top1	top3	top5
grads	.164 ± .012	.284 ± .003	.385 ± .004
grads x I	.196 ± .025	.290 ± .011	.385 ± .010
ig 20	.414 ± .005	.478 ± .011	.518 ± .011
ig 50	.414 ± .005	.478 ± .011	.518 ± .011
ig 100	.414 ± .005	.478 ± .011	.518 ± .012
sg 20	.154 ± .009	.274 ± .007	.383 ± .007
sg 50	.158 ± .007	.276 ± .003	.386 ± .003
sg 100	.165 ± .008	.276 ± .003	.383 ± .000
sg 20 x I	.399 ± .013	.464 ± .014	.513 ± .012
sg 50 x I	.429 ± .015	.488 ± .010	.524 ± .011
sg 100 x I	.436 ± .009	.494 ± .015	.533 ± .013
ks 100	.365 ± .019	.440 ± .009	.488 ± .008
ks 200	.402 ± .020	.477 ± .010	.515 ± .011
ks 500	.423 ± .015	.509 ± .008	.540 ± .011
Chefer et al.	.283 ± .025	.384 ± .008	.464 ± .006
random	.063 ± .006	.201 ± .002	.337 ± .004

Table 6.2: The evaluation metrics for the *bert-base-cased* and *bert-medium* models on the test split of the SST dataset. The metrics are averages from five models and a standard deviation.

Czert			
method	top5	top10	top15
grads	.169 ± .007	.249 ± .008	.312 ± .009
grads x I	.125 ± .013	.201 ± .014	.266 ± .015
ig 20	.255 ± .013	.356 ± .012	.429 ± .010
ig 50	.256 ± .013	.355 ± .013	.428 ± .011
ig 100	.256 ± .013	.356 ± .013	.428 ± .010
sg 20	.161 ± .012	.243 ± .015	.306 ± .016
sg 50	.162 ± .011	.245 ± .014	.309 ± .015
sg 100	.162 ± .013	.247 ± .014	.312 ± .016
sg 20 x I	.271 ± .014	.376 ± .018	.442 ± .017
sg 50 x I	.275 ± .014	.381 ± .017	.448 ± .018
sg 100 x I	.276 ± .013	.383 ± .018	.450 ± .017
ks 100	.117 ± .004	.193 ± .006	.257 ± .004
ks 200	.119 ± .004	.195 ± .004	.257 ± .003
ks 500	.162 ± .006	.249 ± .005	.313 ± .005
Chefer et al.	.217 ± .047	.331 ± .046	.410 ± .042
random	.057 ± .001	.125 ± .000	.190 ± .002

MiniLMv2			
method	top5	top10	top15
grads	.191 ± .007	.284 ± .007	.351 ± .009
grads x I	.186 ± .015	.272 ± .017	.340 ± .020
ig20	.295 ± .013	.400 ± .015	.474 ± .015
ig50	.295 ± .013	.401 ± .015	.475 ± .014
ig100	.295 ± .013	.401 ± .015	.474 ± .013
sg20	.189 ± .005	.276 ± .004	.344 ± .003
sg50	.188 ± .006	.274 ± .007	.343 ± .007
sg100	.188 ± .005	.275 ± .005	.343 ± .003
sg20 x I	.285 ± .018	.379 ± .020	.447 ± .022
sg50 x I	.292 ± .019	.387 ± .021	.454 ± .021
sg100 x I	.293 ± .019	.388 ± .023	.458 ± .025
ks100	.153 ± .007	.240 ± .008	.310 ± .007
ks200	.152 ± .005	.234 ± .004	.301 ± .004
ks500	.190 ± .013	.278 ± .013	.346 ± .012
Chefer et al.	-	-	-
random	.104 ± .002	.183 ± .004	.251 ± .003

Table 6.3: The evaluation metrics for the *Czert-B-base-cased* and *mMiniLMv2-L6-H384* models on the dev split of the CTDC dataset. The metrics are averages from five models and a standard deviation.

small-e-czech			*certain*
method	top5	top10	top15
grads	.164 ± .003	.243 ± .001	.303 ± .002
grads x I	.150 ± .008	.234 ± .009	.301 ± .009
ig 20	.286 ± .005	.382 ± .009	.445 ± .010
ig 50	.286 ± .005	.382 ± .010	.445 ± .010
ig 100	.286 ± .005	.382 ± .010	.445 ± .010
sg 20	.171 ± .004	.246 ± .003	.301 ± .002
sg 50	.174 ± .007	.250 ± .005	.307 ± .005
sg 100	.174 ± .004	.251 ± .004	.305 ± .005
sg 20 x I	.211 ± .009	.304 ± .015	.367 ± .015
sg 50 x I	.217 ± .009	.310 ± .016	.375 ± .015
sg 100 x I	.220 ± .009	.314 ± .014	.378 ± .014
ks 100	.105 ± .004	.176 ± .006	.236 ± .005
ks 200	.110 ± .009	.178 ± .007	.236 ± .005
ks 500	.155 ± .009	.229 ± .009	.286 ± .007
Chefer et al.	-	-	-
random	.054 ± .001	.118 ± .002	.178 ± .003

Table 6.4: The evaluation metrics for the *small-e-czech* model on the dev split of the CTDC dataset. The metrics are averages from five models and a standard deviation.

increased number of steps.

On the CTDC dataset, the performance saturates at 20 samples (Tables 6.3 and 6.4). With both *MiniLMv2* and *Czert* models, SmoothGRAD x Input performs similarly to Integrated Gradients, but with the *small-e-czech* models, Integrated Gradients outperform SmoothGRAD x Input.

SmoothGRAD x Input is either the best or the second best-performing method. Pure SmoothGRAD does not perform well, but that seems to be caused by the design of our datasets and metrics. SmoothGRAD x Input performs well even with a small number of samples. SmoothGRAD x Input performs better with large models than Integrated Gradients, regardless of the number of samples. On smaller models, it outperforms or is on par with Integrated Gradients. The *small-e-czech* models are an exception, where Integrated Gradients surpass SmoothGRAD x Input. The method is easy to implement. It does, however, require some way to pass an embedded sequence instead of token ids.

6.1.2 Integrated Gradients

In most cases, there are better-performing methods than Integrated Gradients. Notable exceptions are the *small-e-czech* models fine-tuned on the CTDC dataset (Table 6.4). There, Integrated Gradients outperform the other methods.

When looking at the smaller models fine-tuned on the SST dataset, we can see a very fast saturation of performance (Tables 6.1 and 6.2). Increasing the number of interpolation steps does not improve performance. The base-size model is an exception where increasing the number of steps from 20 to 50 brings a measurable improvement. The standard deviations of the scores for the base- and medium-size models are high compared to other methods such as SmoothGRAD, which suggests that SmoothGRAD might be more stable than Integrated Gradients on larger models.

We do not observe the same behavior with the Czert models fine-tuned on the CTDC dataset (Table 6.3), where Integrated Gradients exhibit the same behavior as the medium, small, and mini-size models fine-tuned on the SST dataset.

In conclusion, Integrated Gradients do not outperform SmoothGRAD in most cases. Exceptions are the *small-e-czech* ELECTRA models, where Integrated Gradients perform well above SmoothGRAD. With smaller models and the *Czert* model, there is no measurable improvement when increasing the number of interpolation steps beyond 20. While the choice of a baseline is problematic, our experiments show that an all-zero baseline is appropriate.

The necessary number of interpolation steps can be checked experimentally via the axiom of Completeness.

6.1.3 Gradients and Gradients x Input

Vanilla Gradients do not perform very well compared to other methods. We see that multiplying them by input does not consistently improve performance. When we look at the results for the SST dataset (Tables 6.1 and 6.2), Gradients x Input show worse performance on the base- and medium-size models when compared to Vanilla Gradients. The *bert-medium* model shows only a minor difference between Vanilla Gradients and Gradients x Input, while the *bert-base* model shows a significant difference. The small- and mini-size models show Gradients x Input outperforming Vanilla Gradients.

Gradients x Input do not improve performance over Vanilla Gradients for any of the CTDC models (Tables 6.3 and 6.4). Similar to the SST *bert-base* models, the *Czert* models show Gradients x Input being significantly worse compared to Vanilla Gradients. The difference between Vanilla Gradients and Gradients x Input is smaller when we look at the *MiniLMv2* and *small-e-czech* models.

The *small-e-czech* models are roughly comparable in size to the *bert-mini* models we fine-tune on the SST dataset (see Tables 5.6 and 5.4 for model sizes). While Gradients x Inputs outperform Vanilla Gradients with the *bert-mini* models, they are slightly worse with the *small-e-czech* models. The models were fine-tuned on different tasks and are not directly comparable, but the fact that the *bert-mini* models were distilled from a larger model could be a contributing factor.

From the results, we see that, in general, Gradients x Input does not perform well on large models. On smaller models, Gradients x Input can improve the performance compared to Vanilla Gradients or at least not significantly worsen it. The main advantage of both Vanilla Gradients and Gradients x Input is their simple implementation and speed. Only a single forward and backward pass is required to compute the attributions. Both methods also have issues, such as local instability and saturation around the input. SmoothGRAD and Integrated Gradients address these issues. The authors of SmoothGRAD explicitly question (Smilkov et al., 2017) the suitability of Vanilla Gradients as an attribution method and demonstrate its flaws. Moreover, other attribution methods show significantly better performance.

6.1.4 Chefer et al.

Chefer et al. outperform Vanilla Gradients and Gradients x Input but mostly underperform compared to other methods. On the SST dataset, the performance is roughly equal to Integrated Gradients with the base-size BERT models. With the medium- and small-size models, Chefer et al. perform similarly to KernelSHAP with lower sample counts. The performance falls with the mini-size BERT models.

On the CTDC dataset, we only tested the method with the *Czert* models. There, Chefer et al. perform below Integrated Gradients.

The advantage of this method is the speed. It requires only a single forward propagation and two backward propagations. While other methods perform better, they also need multiple forward and backward propagations, potentially limiting their practical applicability. Chefer et al. produce only positive attributions and require a custom implementation. The authors provide an implementation for the BERT architecture, but applying the method to other architectures necessitates modifying the model implementation.

6.1.5 KernelSHAP

We observe a disparity between KernelSHAP performance on the SST and CTDC datasets. When applied to the models fine-tuned on the SST dataset, KernelSHAP is often the best-performing method. It outperforms SmoothGRAD on the base-size BERT models (Table 6.1). When looking at models fine-tuned on the CTDC dataset, KernelSHAP is the worst-performing method (Tables 6.3 and 6.4).

We can explain the difference by looking at how the method generates attributions. KernelSHAP fits a linear regression model to approximate the local behavior around the input. The SST dataset has short sequences (an average of 24 tokens). The CTDC dataset has longer sequences with an average of 292 tokens (using the Czert tokenizer, discarding sequences with more than 512 tokens and less than 30 tokens). The length of the input sequence defines the size of the linear regression model. With very short sequences in the SST dataset, fewer samples are necessary to train the model. With the CTDC models, we can see that while the performance of KernelSHAP is not good, the jump from 200 to 500 samples significantly improves it (Tables 6.3 and 6.4). That implies that an increase in the number of samples could improve further improve performance.

To test this hypothesis, we have done additional tests on one of our *small-e-czech* models using KernelSHAP with $n = 1\ 000$ and 2 000 (Table

small-e-czech			
method	top5	top10	top15
ks 100	0.106	0.177	0.233
ks 200	0.111	0.178	0.235
ks 500	0.155	0.231	0.288
ks 1 000	0.189	0.270	0.327
ks 2 000	0.217	0.294	0.348
random	0.052	0.120	0.178

Table 6.5: The evaluation metrics for a *small-e-czech* model on the dev split of the CTDC dataset. Metrics for KernelSHAP are shown. KernelSHAP was additionally tested with $n = 1\ 000$ and 2 000.

6.5). We observe an improvement in metrics, which puts KernelSHAP on par with SmoothGRAD x Input (Table 6.4). KernelSHAP, however, requires $n = 2\ 000$ to reach this level of performance. Due to time constraints, we did not test higher values of n.

From the results, we can see that when the input sequences are short, only a small number of samples is required to achieve good results. The performance advantage of KernelSHAP is apparent with larger models, where it outperforms the other methods. With smaller models, the advantage is still present. However, the other gradient-based methods require relatively few samples while generating on-par or superior attributions, requiring less compute. From our testing, KernelSHAP seems unsuitable when long sequences are processed. While it may perform well with more samples, then the computational requirements exceed the other methods.

6.1.6 The Impact of Overfitting

As many of the methods we test are primarily gradient-based or utilize gradients in some way, we wanted to examine the effect overfitting may have on the attributions. The models we fine-tune on the CTDC dataset do not suffer from overfitting (see Figure 6.2), but all of the SST models do (see Figure 6.1).

We focus on the base-size BERT model on the SST dataset. To examine the impact of overfitting, we fine-tune base-size BERT models for one, two, four, and five epochs. We then calculate and evaluate the attributions for the models (Table 6.6).

As expected, we see an overall degradation in the performance of most attribution methods with increased training time. A notable exception is

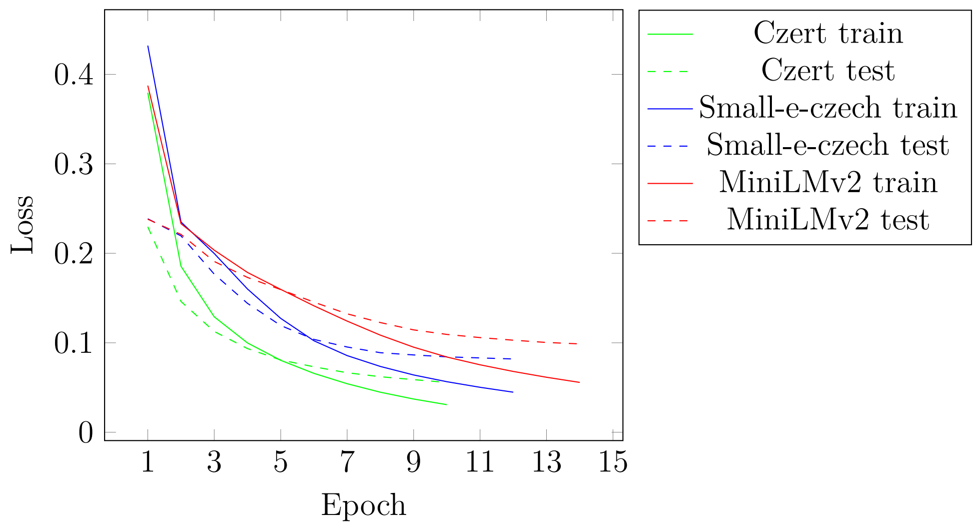


Figure 6.1: Train and test losses of the models fine-tuned on the CTDC dataset.

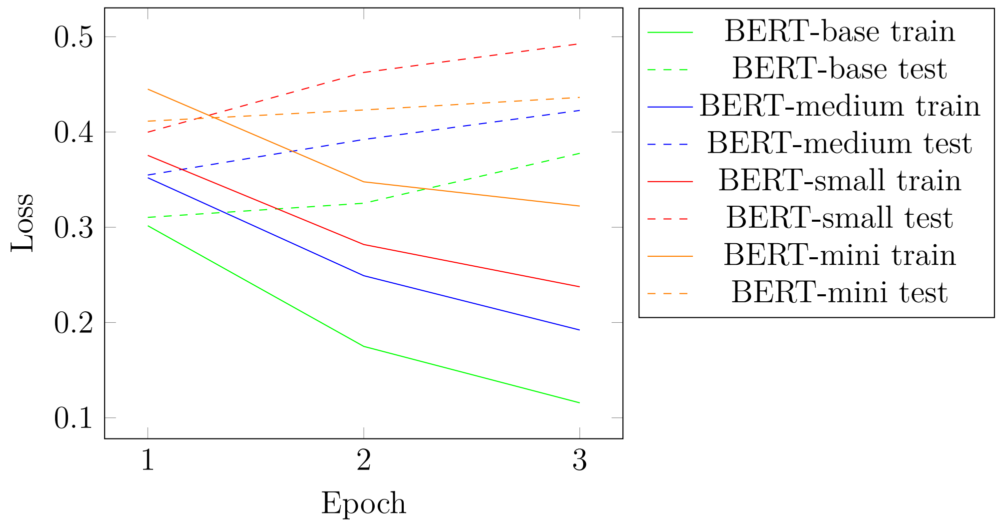


Figure 6.2: Train and test losses of models we fine-tuned on the SST dataset.

BERT-base					
epochs	1	2	3	4	5
grads	.158	.150	.155	.137	.145
grads x I	.135	.122	.107	.125	.097
ig 20	.295	.284	.275	.216	.205
ig 50	.338	.300	.306	.243	.216
ig 100	.340	.307	.314	.258	.242
sg 20	.127	.130	.118	.130	.112
sg 50	.135	.120	.115	.125	.120
sg 100	.129	.109	.121	.105	.121
sg 20 x I	.367	.331	.313	.315	.301
sg 50 x I	.384	.364	.322	.325	.321
sg 100 x I	.409	.360	.348	.341	.321
ks 100	.329	.322	.323	.315	.335
ks 200	.374	.384	.363	.361	.369
ks 500	.413	.408	.401	.400	.406
Chefer et al.	.339	.297	.293	.218	.258
random	.054	.050	.050	.056	.057

Table 6.6: The evaluation metrics for *bert-base-cased* models trained for different numbers of epochs on the test split of the SST dataset. We only report the *top1* metric.

KernelSHAP, where almost no degradation occurs. Because KernelSHAP does not rely on gradients to calculate the attributions, overfitting does not have a significant effect.

Integrated Gradients produce significantly worse attributions in the fourth and fifth epochs. We also see that with the increase in training time, the performance of Integrated Gradients stops saturating at 50 interpolation steps. In the fifth epoch, we see a jump in performance when comparing 50 and 100 interpolation steps. The jump is the effect of overfitting, where the model behaves reasonably at inputs present in the training dataset but does not generalize well. As part of the interpolation, Integrated Gradients create artificial inputs that may not be realistic. The gradient function of the loss w.r.t. the input embeddings can then oscillate, making it harder to integrate. That causes the improvement in the Integrated Gradients attributions at later epochs when increasing the number of interpolation steps.

Sanyal and Ren (2021) discuss a similar issue and propose Discretized Integrated Gradients (DIG). DIG is a modification of Integrated Gradients. DIG does not use linear interpolation between a baseline and the input. Instead, the interpolation path consists of points close to existing token embeddings in the embedding space. The authors argue that the model is trained to operate with existing token embeddings, and as such, the gradients at unrealistic points in the embedding space may be unfaithful.

SmoothGRAD x Input also shows degraded performance at later epochs, but to a lesser extent when compared to Integrated Gradients. The difference in performance between SmoothGRAD x Input and Integrated Gradients at the fifth epoch and $n = 20$ is significant. For the base-size BERT models, we used a noise size of 0.05, meaning the generated artificial samples are closer to the original input when compared to a linear interpolation between an all-zero baseline and the input.

The method proposed by Chefer et al. is also affected in the fourth and fifth epochs. That is expected, as it uses gradients of the loss w.r.t. the attention maps to scale the relevance and identify negative contributions.

We can also see that the performance of Gradients x Input progressively falls with the increase in epochs. The exception is the fourth epoch, where we see an increase in performance compared to the third epoch.

7 Conclusion

In this thesis, we modify the CTDC dataset to evaluate attribution methods. We fine-tune multiple different models on the modified dataset and SST. We choose multiple attribution methods applicable to Transformer models and evaluate them on the datasets. We first determine suitable hyperparameters for methods that require them. Here, we use existing literature in combination with additional testing. We then evaluate the attribution methods on the two datasets using seven different models with five instances of each.

Our testing shows that SmoothGRAD x Input is the best-performing attribution method. On small models, SmoothGRAD is similar in performance to Integrated Gradients. SmoothGRAD x Input on the CTDC dataset saturates in performance, while on the SST dataset, the performance improves with additional samples. Vanilla Gradients and Gradients x Input do not perform well compared to other methods. The performance of Gradients x Input improves as the model size decreases. Integrated Gradients perform worse than SmoothGRAD x Input with one exception; increasing the number of interpolation steps beyond 20 does not improve the performance in all but one case. KernelSHAP performs well on the SST dataset but fails on the CTDC dataset. The cause is the design of the attribution method, where more samples are required to explain longer inputs. The method proposed by Chefer et al. performs between Integrated Gradients and Vanilla Gradients. It produces only positive attributions and requires modifications to the model, but it is fast compared to other methods that require multiple backpropagations.

We show that the models we fine-tune on the SST dataset are overfitted. We then examine how the attribution methods behave when a model is fine-tuned for different numbers of epochs. We observe an overall degradation of performance across all attribution methods except KernelSHAP. KernelSHAP is the only method we evaluate that does not use gradients. We notice a significant fall in the performance of Integrated Gradients with an increased number of epochs. SmoothGRAD x Input also suffers, but to a lesser extent when compared to Integrated Gradients.

8 List of Abbreviations

AI - Artificial Intelligence
XAI - Explainable Artificial Intelligence
NLP - Natural Language Processing
LSTM - Long Short-Term Memory
RNN - Recurrent Neural Network
CNN - Convolutional Neural Network
GRU - Gated Recurrent Network
MLM - Masked Language Modeling
NSP - Next Sentence Prediction
QA - Question Answering
IG - Integrated Gradients
SST - Stanford Sentiment Treebank
CTDC - Czech Text Document Corpus
AUC-TP - Area Under Curve on Threshold Performance
AOPC - Area Over the Pertubation Curve
AUPRC - Area Under the Precision-Recall Curve
IOU - Intersection Over Union
LRP - Layer-wise Relevance Propagation
PMI - Point-wise Mutual Information
DIG - Discretized Integrated Gradients
SHAP - Shapley Additive Explanations
GDPR - General Data Protection Regulation
LIME - Local Interpretable Model Explanations

Bibliography

Abnar, S. and Zuidema, W. H. (2020). Quantifying attention flow in transformers. *CoRR*, abs/2005.00928.

Ancona, M., Ceolini, E., Öztireli, A. C., and Gross, M. H. (2017). A unified view of gradient-based attribution methods for deep neural networks. *CoRR*, abs/1711.06104.

Angelov, P. P., Soares, E. A., Jiang, R., Arnold, N. I., and Atkinson, P. M. (2021). Explainable artificial intelligence: an analytical review. *WIREs Data Mining and Knowledge Discovery*, 11(5):e1424.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.

Bartička, V., Pražák, O., Konopík, M., and Sido, J. (2022). Evaluating attribution methods for explainable nlp with transformers. In Sojka, P., Horák, A., Kopeček, I., and Pala, K., editors, *Text, Speech, and Dialogue*, pages 3–15, Cham. Springer International Publishing.

Bhargava, P., Drozd, A., and Rogers, A. (2021). Generalization in NLI: ways (not) to go beyond simple heuristics. *CoRR*, abs/2110.01518.

Brunner, G., Liu, Y., Pascual, D., Richter, O., and Wattenhofer, R. (2019). On the validity of self-attention as explanation in transformer models. *CoRR*, abs/1908.04211.

Chefer, H., Gur, S., and Wolf, L. (2020). Transformer interpretability beyond attention visualization. *CoRR*, abs/2012.09838.

Clark, K., Luong, M., Le, Q. V., and Manning, C. D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Confalonieri, R., Coba, L., Wagner, B., and Besold, T. R. (2021). A historical perspective of explainable artificial intelligence. *WIREs Data Mining and Knowledge Discovery*, 11(1):e1391.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

DeYoung, J., Jain, S., Rajani, N. F., Lehman, E., Xiong, C., Socher, R., and Wallace, B. C. (2019). ERASER: A benchmark to evaluate rationalized NLP models. *CoRR*, abs/1911.03429.

Erion, G., Janizek, J. D., Sturmfels, P., Lundberg, S. M., and Lee, S.-I. (2021). Improving performance of deep learning models with axiomatic attribution priors and expected gradients. *Nature Machine Intelligence*, 3(7):620–631.

Gohel, P., Singh, P., and Mohanty, M. (2021). Explainable AI: current status and future directions. *CoRR*, abs/2107.07045.

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., Han, W., Huang, M., Jin, Q., Lan, Y., Liu, Y., Liu, Z., Lu, Z., Qiu, X., Song, R., Tang, J., Wen, J.-R., Yuan, J., Zhao, W. X., and Zhu, J. (2021). Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

Jain, S. and Wallace, B. C. (2019). Attention is not Explanation. In *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)*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

Kobayashi, G., Kuribayashi, T., Yokoi, S., and Inui, K. (2020). Attention is not only a weight: Analyzing transformers with vector norms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7057–7075, Online. Association for Computational Linguistics.

Kocián, M., Náplava, J., Stancl, D., and Kadlec, V. (2021). Siamese bert-based model for web search relevance ranking evaluated on a new czech dataset. *CoRR*, abs/2112.01810.

Kral, P. and Lenc, L. (2018). Czech text document corpus v 2.0. In chair), N. C. C., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Hasida, K., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., Piperidis, S., and Tokunaga, T., editors, *Proceedings of the Eleventh International Conference* *on Language Resources and Evaluation (LREC 2018)*, Paris, France. European Language Resources Association (ELRA).

Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2021). Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1).

Liu, S., Le, F., Chakraborty, S., and Abdelzaher, T. (2021). On exploring attention-based explanation for transformer models in text classification. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1193–1203.

Liu, Y., Li, H., Guo, Y., Kong, C., Li, J., and Wang, S. (2022). Rethinking attention-model explainability through faithfulness violation test. *CoRR*, abs/2201.12114.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Lundberg, S. M. and Lee, S. (2017). A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Mohseni, S., Zarei, N., and Ragan, E. D. (2018). A survey of evaluation methods and measures for interpretable machine learning. *CoRR*, abs/1811.11839.

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., and Müller, K.-R. (2019). *Layer-Wise Relevance Propagation: An Overview*, pages 193–209. Springer International Publishing, Cham.

Nauta, M., Trienes, J., Pathak, S., Nguyen, E., Peters, M., Schmitt, Y., Schlötterer, J., van Keulen, M., and Seifert, C. (2022). From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable AI. *CoRR*, abs/2201.08164.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Pruthi, D., Gupta, M., Dhingra, B., Neubig, G., and Lipton, Z. C. (2020). Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online. Association for Computational Linguistics.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Remmer, E. (2022). Explainability methods for transformer-based artificial neural networks:: a comparative analysis.

Ribeiro, M., Singh, S., and Guestrin, C. (2016). “why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California. Association for Computational Linguistics.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Sanyal, S. and Ren, X. (2021). Discretized integrated gradients for explaining language models. *CoRR*, abs/2108.13654.

Selbst, A. D. and Powles, J. (2017). Meaningful information and the right to explanation. *International Data Privacy Law*, 7(4):233–242.

Serrano, S. and Smith, N. A. (2019). Is attention interpretable? *CoRR*, abs/1906.03731.

Shapley, L. S. (1951). *Notes on the N-Person Game – II: The Value of an N-Person Game*. RAND Corporation, Santa Monica, CA.

Shen, T., Zhou, T., Long, G., Jiang, J., Pan, S., and Zhang, C. (2017). Disan: Directional self-attention network for rnn/cnn-free language understanding. *CoRR*, abs/1709.04696.

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. (2016). Not just a black box: Learning important features through propagating activation differences. *CoRR*, abs/1605.01713.

Sido, J., Prazák, O., Pribán, P., Pasek, J., Seják, M., and Konopík, M. (2021). Czert - czech bert-like model for language representation. *CoRR*, abs/2103.13031.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*.

Smilkov, D., Thorat, N., Kim, B., Viégas, F. B., and Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Sturmfels, P., Lundberg, S., and Lee, S.-I. (2020). Visualizing the impact of feature attribution baselines. *Distill*. https://distill.pub/2020/attribution-baselines.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. (2019). Patient knowledge distillation for BERT model compression. *CoRR*, abs/1908.09355.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365.

Tan, H. (2022). Maximum entropy baseline for integrated gradients.

Turc, I., Chang, M., Lee, K., and Toutanova, K. (2019). Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Wang, W., Bao, H., Huang, S., Dong, L., and Wei, F. (2020). Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *CoRR*, abs/2012.15828.

Wiegreffe, S. and Pinter, Y. (2019). Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20, Hong Kong, China. Association for Computational Linguistics.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

A Archive Contents

The archive, which is a part of this thesis, contains the following directories:

- `Application_and_libraries` contains Python scripts that are a part of this thesis. It also contains a `README.md` file with a detailed description of the contents, the installation of prerequisites, and a user manual.
- `Results` contains contains files with the created attributions and their metrics. It contains a `README.md` file with more information.
- `Text_thesis` contains the LaTeX files, images, CSV files, and the generated PDF of this thesis.
- `Poster` contains the .pub and PDF files of the poster.

Due to their size, the models used in this thesis are not included in the archive, and can be downloaded instead from `https://bit.ly/3oQUuoE`.

The CTDC and SST datasets, which would be in a `Input_data` folder, are not included in the archive. Instead, the `Application_and_libraries` folder contains scripts to download and preprocess these datasets. A detailed description of this process is in the user manual.

B User Manual

The application was tested with Python 3.8.9 and Windows 10 Pro (build 19045.2846). The scripts are in the `Application_and_libraries` folder of the archive (see Appendix A), which also contains a `requirements.txt` file with required libraries and their versions.

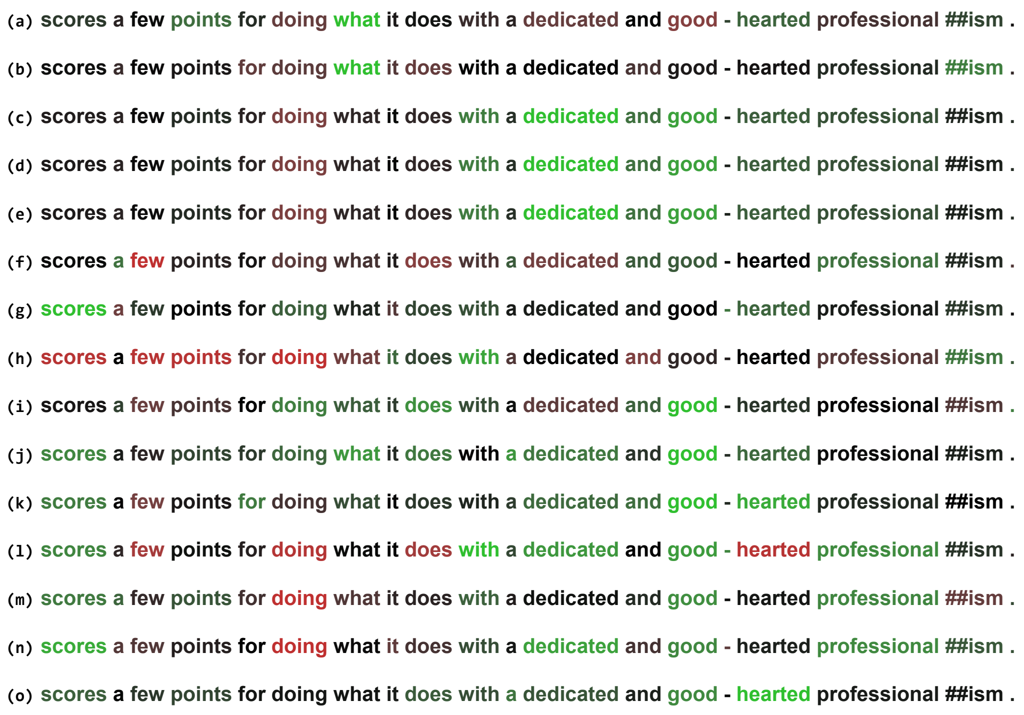
Our fine-tuned models and datasets are not included in the archive and must be downloaded separately. The `Application_and_libraries` folder contains a `README.md` file with instructions on how to download the models, download and preprocess the datasets, train new models, generate attributions, and evaluate the attributions. It also describes how to replicate our results.

As the user manual is extensive, it is not included here. Please refer to the `README.md` file for further information.

C Visualizations

We include visual representations of attributions for two positive and two negative samples from the SST dataset. We used a *bert-small* model to generate them. The model classified these samples correctly.

Green colored text signifies a contribution towards a positive sentiment. Red colored text signifies a contribution towards a negative sentiment. Gray text signifies no contribution.



(a) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(b) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(c) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(d) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(e) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(f) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(g) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(h) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(i) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(j) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(k) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(l) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(m) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(n) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

(o) scores a few points for doing what it does with a dedicated and good - hearted professional ##ism .

Figure C.1: A visual representation of attributions for a sample from the SST dataset test split. This sample has a **positive sentiment**. (a) Gradients, (b) Gradients x Input, (c) Integrated Gradients with $n = 20$, (d) Integrated Gradients with $n = 50$, (e) Integrated Gradients with $n = 100$, (f) SmoothGRAD with $n = 20$, (g) SmoothGRAD with $n = 50$, (h) SmoothGRAD with $n = 100$, (i) SmoothGRAD x Input with $n = 20$, (j) SmoothGRAD x Input with $n = 50$, (k) SmoothGRAD x Input with $n = 100$, (l) KernelSHAP with $n = 100$, (m) KernelSHAP with $n = 200$, (n) KernelSHAP with $n = 500$, (o) Chefer et al.

(a) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(b) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(c) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(d) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(e) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(f) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(g) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(h) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(i) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(j) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(k) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(l) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(m) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(n) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

(o) most of crush is a clever and capt ##ivating romantic comedy with a welcome pinch of tar ##tness .

Figure C.2: A visual representation of attributions for a sample from the SST dataset test split. This sample has a **positive sentiment**. (a) Gradients, (b) Gradients x Input, (c) Integrated Gradients with $n = 20$, (d) Integrated Gradients with $n = 50$, (e) Integrated Gradients with $n = 100$, (f) SmoothGRAD with $n = 20$, (g) SmoothGRAD with $n = 50$, (h) SmoothGRAD with $n = 100$, (i) SmoothGRAD x Input with $n = 20$, (j) SmoothGRAD x Input with $n = 50$, (k) SmoothGRAD x Input with $n = 100$, (l) KernelSHAP with $n = 100$, (m) KernelSHAP with $n = 200$, (n) KernelSHAP with $n = 500$, (o) Chefer et al.

(a) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(b) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(c) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(d) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(e) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(f) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(g) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(h) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(i) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(j) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(k) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(l) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(m) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(n) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

(o) the cr ##eak ##ing , rusty ship makes a fine backdrop , but the ghosts ' haunting is routine .

Figure C.3: A visual representation of attributions for a sample from the SST dataset test split. This sample has a **negative sentiment**. (a) Gradients, (b) Gradients x Input, (c) Integrated Gradients with $n = 20$, (d) Integrated Gradients with $n = 50$, (e) Integrated Gradients with $n = 100$, (f) SmoothGRAD with $n = 20$, (g) SmoothGRAD with $n = 50$, (h) SmoothGRAD with $n = 100$, (i) SmoothGRAD x Input with $n = 20$, (j) SmoothGRAD x Input with $n = 50$, (k) SmoothGRAD x Input with $n = 100$, (l) KernelSHAP with $n = 100$, (m) KernelSHAP with $n = 200$, (n) KernelSHAP with $n = 500$, (o) Chefer et al.

(a) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(b) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(c) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(d) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(e) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(f) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(g) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(h) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(i) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(j) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(k) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(l) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(m) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(n) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

(o) began life as a computer game , then mor ##ph ##ed into a movie - - a bad one , of course .

Figure C.4: A visual representation of attributions for a sample from the SST dataset test split. This sample has a **negative sentiment**. (a) Gradients, (b) Gradients x Input, (c) Integrated Gradients with $n = 20$, (d) Integrated Gradients with $n = 50$, (e) Integrated Gradients with $n = 100$, (f) SmoothGRAD with $n = 20$, (g) SmoothGRAD with $n = 50$, (h) SmoothGRAD with $n = 100$, (i) SmoothGRAD x Input with $n = 20$, (j) SmoothGRAD x Input with $n = 50$, (k) SmoothGRAD x Input with $n = 100$, (l) KernelSHAP with $n = 100$, (m) KernelSHAP with $n = 200$, (n) KernelSHAP with $n = 500$, (o) Chefer et al.