Using Captum to Explain Generative Language Models

Vivek Miglani*, Aobo Yang*, Aram H. Markosyan, Diego Garcia-Olano, Narine Kokhlikyan

Meta AI
{vivekm, aoboyang, amarkos, diegoolano, narine}@meta.com

Abstract

Captum is a comprehensive library for model explainability in PyTorch, offering a range of methods from the interpretability literature to enhance users' understanding of PyTorch models. In this paper, we introduce new features in Captum[1] that are specifically designed to analyze the behavior of generative language models. We provide an overview of the available functionalities and example applications of their potential for understanding learned associations within generative language models.

1 Introduction

Model interpretability and explainability have become significantly more important as machine learning models are used in critical domains such as healthcare and law. It is insufficient to simply make a prediction through a black-box model and important to better understand why the model made a particular decision.

Interest in Large Language Models (LLMs) has also grown exponentially in the past few years with the release of increasingly large and more powerful models such as GPT-4 (OpenAI, 2023). A lack of explainability continues to exist despite larger models, and with the use of these models expanding to more and more use-cases, it is increasingly important to have access to tooling providing model explanations.

Captum is an open-source model explainability library for PyTorch providing a variety of generic interpretability methods proposed in recent literature such as Integrated Gradients, LIME, DeepLift, TracIn, TCAV and more (Kokhlikyan et al., 2020).

In this work, we discuss newly open-sourced functionalities in Captum v0.7 to easily apply explainability methods to large generative language models, such as GPT-3.

*Denotes equal contribution
[1]https://captum.ai

2 Attribution Methods

One important class of explainability methods is attribution or feature importance methods, which output a score corresponding to each input feature's contribution or impact to a model's final output.

Formally, given a function $f : \mathbb{R}^d \to \mathbb{R}$, where $f \in \mathbb{F}$ and $X \in \mathbb{R}^d$ is a single input vector consisting of d dimensions or features, an attribution method is defined as a function $\phi : \mathbb{F} \times \mathbb{R}^d \to \mathbb{R}^d$. Each element in the attribution output corresponds to a score of the contribution of corresponding feature $i \in D$, where D denotes the set of all feature indices $D = \{1, 2, ..., d\}$.

Many attribution methods also require a baseline or reference input $B \in \mathbb{R}^d$ defining a comparison input point to measure feature importance with respect to.

We utilize the notation X_S to denote selecting the feature values with indices from the set $S \subseteq D$ and the remaining indices from B. Formally, the value of feature i in X_S is $(X_S)_i = I_{i\in S}X_i + I_{i\notin S}B_i$, where I is the indicator function.

In this section, we provide background context on attribution methods available in Captum. These methods can be categorized broadly into (i) perturbation-based methods, which utilize repeated evaluations of a black-box model on perturbed inputs to estimate attribution scores, and (ii) gradient-based methods, which utilize backpropagated gradient information to estimate attribution scores. Perturbation-based methods do not require access to model weights, while gradient-based models do.

2.1 Perturbation-Based Methods

2.1.1 Feature Ablation

The most straightforward attribution is feature ablation, where each feature is substituted with the corresponding element of the baseline feature vector to estimate the corresponding importance.

Formally, this method is defined as

$$\phi_i(f, X) = f(X) - f(X_{D\setminus\{i\}}) \tag{1}$$

Feature Ablation has clear advantages as a simple and straightforward method, but the resulting attributions may not fully capture the impacts of feature interactions since features are ablated individually.

2.1.2 Shapley Value Sampling

Shapley Values originated from cooperative game theory as an approach to distribute payouts fairly among players in a cooperative game. Analogously, in the attribution setting, Shapley Values assign scores to input features, with payouts corresponding to a feature's contribution to the model output. Shapley Values satisfy a variety of theoretical properties including efficiency, symmetry and linearity. Formally, this method is defined as

$$\phi_i(f, X) = \sum_{S \subseteq D\setminus\{i\}} \left[\frac{|S|!(|D| - |S| - 1)!}{|D|!} \right. \tag{2}$$
$$\left. f(X_{S\cup\{i\}}) - f(X_S) \right]$$

While computing this quantity exactly requires an exponential number of evaluations in the number of features, we can estimate this quantity using a sampling approach (Castro et al., 2009). The approach involves selecting a permutation of the d features and adding the features one-by-one to the original baseline. The output change as a result of adding each feature accounts for its contribution, and averaging this over sampled perturbations results in an unbiased estimate of Shapley Values.

2.1.3 LIME

LIME or Locally Interpretable Model Explanations proposes a generic approach to sample points in the neighborhood of the input point X and train an interpretable model (such as a linear model) based on the results of the local evaluations (Ribeiro et al., 2016).

This method proposes reparametrizing the input space to construct interpretable features such as super-pixels in images and then evaluating the original model on a variety of perturbations of the interpretable features. The method can be utilized with any perturbation sampling and weighting approaches and interpretable model / regularization parameters. The interpretable model can then be used as an explanation of the model's behavior in the local region surrounding the target input point. For a linear model, the coefficients of this model can be considered as attribution scores for the corresponding feature.

2.1.4 Kernel SHAP

Kernel SHAP is a special case of the LIME framework, which sets the sampling approach, intepretable model, and regularization in a specific way such that the results theoretically approximate Shapley Values (Lundberg and Lee, 2017).

2.2 Gradient Based Methods

2.2.1 Saliency

Saliency is a simple gradient-based approach, utilizing the model's gradient at the input point as the corresponding feature attribution (Simonyan et al., 2013). This method can be understood as taking a first order approximation of the function, in which the gradients would serve as the coefficients of each feature in the model.

$$\phi_i(f, X) = f'(X) \tag{3}$$

2.2.2 Integrated Gradients

Integrated Gradients estimates attribution by computing the path integral of model gradients between the baseline point and input point (Sundararajan et al., 2017). This approach has been shown to satisfy desirable theoretical properties including sensitivity and implementation invariance. Formally, the method can be defined as

$$\phi_i(f, X) = (X_i - B_i)$$
$$\times \int_{\alpha=0}^{1} \frac{f'(B + (X - B)\alpha)}{\partial x_i} d\alpha \tag{4}$$

2.2.3 Other Gradient-Based Methods

Other popular gradient-based attribution methods include DeepLift and Layerwise Relevance Propogation (LRP) (Shrikumar et al., 2017; Bach et al., 2015). These methods both require a backward pass of the model on the original inputs but customize the backward propagation of specific functions, instead of using their default gradient functions.

3 Language Model Attribution

In Captum v0.7, we propose new functionalities to apply the attribution methods within Captum to analyze the behaviors of LLMs. Users can choose any interested tokens or segments from the input

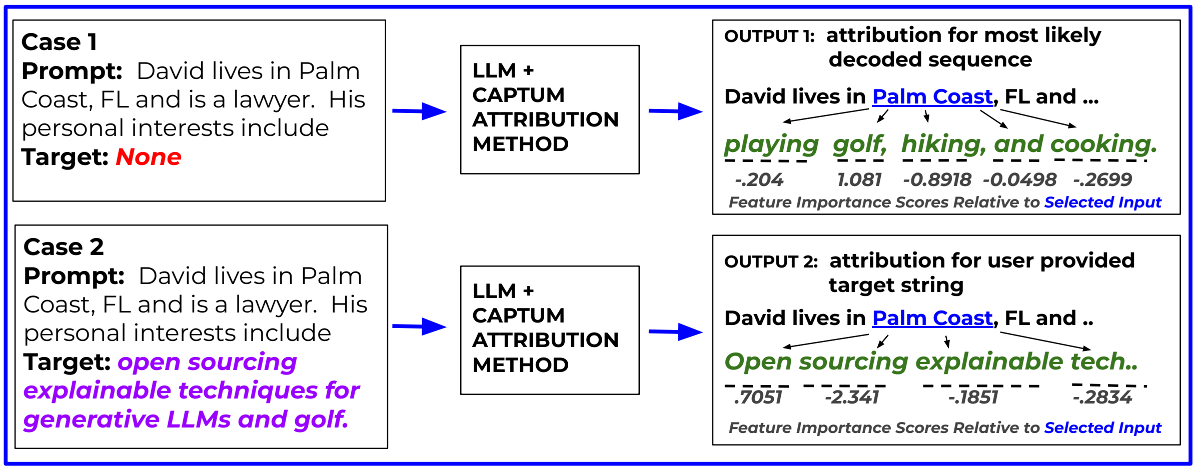
Case 1
Prompt: David lives in Palm Coast, FL and is a lawyer. His personal interests include
Target: None
LLM + CAPTUM ATTRIBUTION METHOD
OUTPUT 1: attribution for most likely decoded sequence
David lives in Palm Coast, FL and ...
playing golf, hiking, and cooking.
-.204 1.081 -0.8918 -0.0498 -.2699
Feature Importance Scores Relative to Selected Input
Case 2
Prompt: David lives in Palm Coast, FL and is a lawyer. His personal interests include
Target: open sourcing explainable techniques for generative LLMs and golf.
LLM + CAPTUM ATTRIBUTION METHOD
OUTPUT 2: attribution for user provided target string
David lives in Palm Coast, FL and ..
Open sourcing explainable tech..
.7051 -2.341 -.1851 -.2834
Feature Importance Scores Relative to Selected Input

Figure 1: Example of applying Captum attribution methods to analyze the input prompt's effect on the output of LLMs, showing two types of target strings accepted by Captum attribution API and token level attribution outputs for both with respect to the input "Palm Coast". In Case 1, no Target string is provided, so attributions are provided for the most likely decoded sequence. In Case 2, attributions are provided for the chosen target output.

prompt as features, e.g., "Palm Coast" in the example shown in Figure 1, and use attribution methods to quantify their impacts to the generation targets, which can be either a specified output sequence or a likely generation from the model.

3.1 Perturbation-Based Methods

We introduce simple APIs to experiment with perturbation-based attribution methods including Feature Ablation, Lime, Kernel SHAP and Shapley Value Sampling.

We prioritize ease-of-use and flexibility, allowing users to customize the chosen features for attribution, mask features into groups as necessary, and define appropriate baselines to ensure perturbed inputs remain within the natural data distribution.

In Figure 2, we demonstrate an example usage of the LLMAttribution API for the simple prompt *"Dave lives in Palm Coast, FL and is a lawyer. His personal interests include"*. Providing this input prompt to a language model to generate the most likely subsequent tokens, we can apply Captum to understand the impact of different parts of the prompt string on the model generation. Figure 3 presents a more customized usage where we use the same function to understand a specific output we are interested in (*"playing golf, hiking, and cooking."*).

3.1.1 Defining Features

Users are able to define and customize 'features' for attribution in the prompt text. The simplest approach would be defining the features as individual tokens.

Unfortunately, in many cases, it doesn't make sense to perturb individual tokens, since this may no longer form a valid sentence in the natural distribution of potential input sequences. For example, perturbing the token "Palm" in the above sentence would result in a city name that is not in the natural distribution of potential cities in Florida, which may lead to unexpected impacts on the perturbed model output. Moreover, tokenizers used in modern LLMs may further break a single word in many cases. For example, the tokenizer can break the word *"spending"* into *"_sp"* and *"ending"*.

The API provides flexibility to define units of attribution as custom interpretable features which could be individual words, tokens, phrases, or even full sentences. For example, in Figure 2, we select the relevant features to be the name, city, state, occupation, and pronoun in the sentence prompt and desire to determine the relative contribution of these contextual features on the model's predicted sentence completion.

Users can define the units for attribution as a list or dictionary of features and provide a format string or function to define a mapping from the attribution units to the full input prompt as shown in Figure 3.

3.1.2 Baselines

The baseline choice is particularly important for computing attribution for text features, as it serves as the reference value used when perturbing the chosen feature. The perturbation-based feature API allows defining custom baselines corresponding to

```
from captum.attr import FeatureAblation, LLMAttribution, TextTemplateFeature

fa = FeatureAblation(model)
llm_attr = LLMAttribution(fa, tokenizer)

inp = TextTemplateFeature(
    # the text template
    "{} lives in {}, {} and is a {}. {} personal interests include",
    # the values of the features
    ["Dave", "Palm Coast", "FL", "lawyer", "His"],
    # the reference baseline values of the features
    baselines=["Sarah", "Seattle", "WA", "doctor", "Her"],
)
llm_attr.attribute(inp)
```

Figure 2: Example of applying Captum with a list of features in a text template

```
inp = TextTemplateFeature(
    "{name} lives in {city}, {state} and is a {occupation}. {pronoun} personal
                                            interests include",
   {"name":"Dave", "city": "Palm Coast", "state": "FL", "occupation":"lawyer", "
                                           pronoun":"His"},
   baselines={"name":"Sarah", "city": "Seattle", "state": "WA", "occupation":"doctor
                                           ", "pronoun":"Her"}
)
attr_result = llm_attr.attribute(inp, target="playing golf, hiking, and cooking.")
attr_result.plot_token_attr()
```

Figure 3: Example of applying Captum with a dictionary of features in a text template and a specific target, and visualize the token attribution

each input feature.

It is recommended to select a baseline which fits the context of the original text and remains within the natural data distribution. For example, replacing the name of a city with another city ensures the sentence remains naturally coherent, but allows measuring the contribution of the particular city selected.

In addition to a single baseline, the Captum API also supports providing a distribution of baselines, provided as either a list or function to sample a replacement option. For example, in the example above, the name "Dave" could be replaced with a sample from the distribution of common first names to measure any impact of the name "Dave" in comparison to the chosen random distribution as shown in Figure 6.

3.1.3 Masking Features

Similar to the underlying Captum attribution methods, we support feature masking, which enables grouping features together to perturb as a single unit and obtain a combined, single attribution score. This functionality may be utilized for highly correlated features in the text input, where it often makes sense to ablate these features together, rather than independently.

For example, in Figure 2, the feature pairs (city, state) and (name, pronoun) are often highly correlated, and thus it may make sense to group them and consider them as a single feature.

3.1.4 Target Selection

For any attribution method, it is also necessary to select the target function output for which attribution outputs are computed. Since language models typically output a probability distribution over the space of tokens for each subsequently generated token, there are numerous choices for the appropriate target.

By default, when no target is provided, the target function behavior is for the attribution method to return attributions with respect to the most likely decoded token sequence.

When a target string is provided, the target function is the log probability of the output sequence from the language model, given the input prompt. For a sequence with multiple tokens, this is numerically computed through the sum of the log probabilities of each token in the target string. Figure 1 shows these two input use cases and shows token level attribution relative to an input subsequence for both.

We also support providing a custom function on the output logit distribution, which allows attribution with respect to an alternate quantity such as the entropy of the output token distribution.

3.2 Gradient-Based Methods

Captum 0.7 also provides a simple API to apply gradient-based methods to LLMs. Applying these methods to language models is typically more challenging than for models with dense feature inputs, since embedding lookups in LLMs are typically non-differentiable functions, and gradient-based attributions need to be obtained with respect to embedding outputs. Captum allows these attributions to be aggregated across embedding dimensions to obtain per-token attribution scores. Figure 4 demonstrates an example of applying Integrated Gradients on a sample input prompt.

3.3 Visualization

We also open source utilities for easily visualizing the attribution outputs from language models. Figure 3 shows how to use the utilities to visualize the attribution result, and Figure 5 demonstrates the heatmap plotted with the prompt along the top, the target string along the left side and feature importance scores in each cell.

(Feature) Value	Golfing	Hiking	Cooking
(Name) Dave	0.4660	-0.2640	-0.4515
(City) Palm Coast	1.0810	-0.8762	-0.2699
(State) FL	0.6070	-0.3620	-0.3513
(Occupation) lawyer	0.7584	-0.1966	0.0331
(Pronoun) His	0.2217	-0.0650	-0.2577

Table 1: Associations between input and generated text features

4 Applications

In this section, we discuss two applications of the attribution methods described above in different contexts. These applications provide additional transparency as well as contribute to a better understanding of a model's learned associations and robustness.

4.1 Understanding Model Associations

This perturbation-based tooling can be particularly useful for improved understanding of learned associations in LLMs.

Consider the example prompt:

"David lives in Palm Coast, FL and is a lawyer. His personal interests include "

We can define features including gender, city, state and occupation. Obtaining attributions on these features against the subsequent target

"playing golf, hiking, and cooking. "

allows us to better understand why the model predicted these personal interests and how each feature correlates with each of these interests. The model might be associating this activity as a common hobby for residents in the specific city or as an activity common to lawyers. Through this choice of features, we can obtain a better understanding of why the model predicted these particular hobbies and how this associates with the context provided in the prompt.

We apply Shapley Value Sampling to better understand how each of the features contributed to the prediction. The corresponding code snippet is shown in the Appendix in Figure 6. Table 1 presents the effects of each feature on the LLM's probability of outputting the corresponding interest, with positive and negative values indicating increases and decreases of the probability respectively. We can therefore identify some interesting and even potentially biased associations. For example, the male name and pronoun, i.e., "Dave" and "His", have positive attribution to "golfing" but negative attribution to "cooking".

4.2 Evaluating Effectiveness of Few-Shot Prompts

Significant prior literature has demonstrated the ability of LLMs to serve as few-shot learners (Brown et al., 2020). We utilize Captum's attribution functionality to better understand the impact and contributions of few-shot examples to model predictions. Table 2 demonstrates four example few shot prompts and corresponding attribution scores when predicting positive sentiment for "I really liked the movie, it had a captivating plot!" movie review.

Here we aim to understand the impact of each example prompt on the *Positive* sentiment prediction. The LLM is asked to predict positive or negative sentiment using the following prompt:

"Decide if the following movie review enclosed in quotes is Positive or Negative. Output only either Positive or Negative:

```
from captum.attr import LayerIntegratedGradients, TextTokenFeature

ig = LayerIntegratedGradients(model, "model.embed_tokens")
llm_attr = LLMGradientAttribution(ig, tokenizer)

inp = TextTokenFeature("Dave lives in Palm Coast, FL and is a lawyer. His personal
                        interests include", tokenizer)
llm_attr.attribute(inp)
```

Figure 4: Example of applying Captum with a gradient-based approach

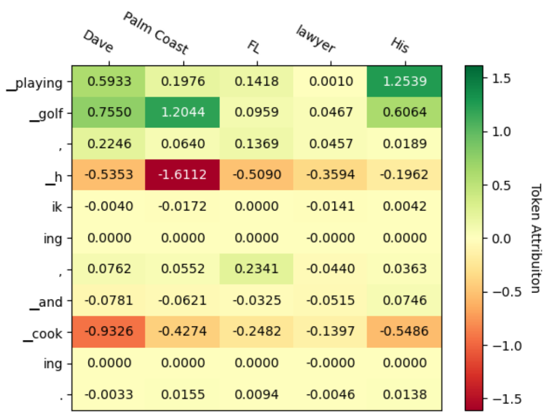

Figure 5: Text Attribution Visualization Example

‘I really liked the movie, it had a captivating plot!’
”

We consider each of the provided example prompts as features and we utilize zero-shot prompt as a baseline in the attribution algorithm. The detailed implementation can be found in Appendix in Figure 7.

We obtain results as shown in Table 2 by applying Shapley Values. Surprisingly, the results suggest that all the provided examples actually reduced confidence in the prediction of "Positive".

Example	Shapley Value
’The movie was ok, the actors weren’t great’ -> Negative	-0.0413
’I loved it, it was an amazing story!’ -> Positive	-0.2751
’Total waste of time!!’ -> Negative	-0.2085
’Won’t recommend’ -> Negative	-0.0399

Table 2: Example prompts’ contribution to model response "Positive."

5 Related Work

Numerous prior works have developed and investigated attribution methods with a variety of properties, but few efforts have been made to develop open-source interpretability tools providing a variety of available methods, particularly for the text domain. Captum was initially developed to fill this gap and provide a centralized resource for recent interpretability methods proposed in literature (Kokhlikyan et al., 2020).

Ecco and inseq are two libraries that have provided attribution functionalities for text / language models (Sarti et al., 2023; Alammar, 2021), and both libraries are built on top of the attribution methods available in Captum. These libraries primarily focus on gradient-based attribution methods, which provide token-level attribution based on gradient information.

In contrast, our main contribution in this work has been a focus on perturbation-based methods and providing flexibility on aspects such as feature definition, baseline choice and masking. We do not necessarily expect that these attribution methods provide a score for each token individually, which is typically the case for gradient-based methods. This shift in structure allows us to generalize to a variety of cases and allows the users to define features for attribution as it fits best.

Some prior work on attribution methods have also demonstrated limitations and counterexamples of these methods in explaining a model’s reliance on input features, particularly with gradient-based attribution methods (Adebayo et al., 2018).

Perturbation-based methods generally have an advantage of being justifiable through the model’s output on counterfactual perturbed inputs. But perturbing features by removing individual tokens or replacing them with pad or other baseline tokens may result in inputs outside of the natural data distribution, and thus, model outputs in this region may not be accurate. The tools developed have been designed to make it easier for developers to select features, baselines, and masks which can ensure perturbations remain within the natural data distribution in order to obtain more reliable feature attribution results.

Recent advances in data augmentation (Pluščec and Šnajder, 2023) for natural language processing have led to the development of a number of open-source libraries (Wang et al., 2021; Papakipos and Bitton, 2022; Zeng et al., 2021; Morris et al., 2020; Ma, 2019; Dhole et al., 2022; Wu et al., 2021). Among many functionalities, these libraries provide a rich set of text perturbations. Some libraries have specific focus, e.g. perturbing demographic references (Qian et al., 2022). An interesting direction of future work will be the extension of our present API to provide fully automated feature and baseline selections, allowing users to simply provide an input string and automatically identify appropriate text features and corresponding baselines for attribution.

6 Conclusion

In this work, we introduce new features in the open source library Captum that are specifically designed to analyze the behavior of generative LLMs. We provide an overview of the available functionalities and example applications of their potential in understanding learned associations and evaluating effectiveness of few-shot prompts within generative LLMs. We demonstrate examples for using perturbation and gradient-based attribution methods with Captum which highlight the API's flexibility on aspects such as feature definition, baseline choice and masking. This flexibility in structure allows users to generalize to a variety of cases, simplifying their ability to conduct explainability experiments on generative LLMs.

In the future, we plan to expand our API for additional automation in baseline and feature selection as well as incorporate other categories of interpretability techniques for language models. Runtime performance optimization of attribution algorithms is another area of research that could be beneficial for the OSS community.

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. 2018. Sanity checks for saliency maps.

J Alammar. 2021. Ecco: An open source library for the explainability of transformer language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Javier Castro, Daniel Gómez, and Juan Tejada. 2009. Polynomial calculation of the shapley value based on sampling. *Computers & Operations Research*, 36(5):1726–1730.

Kaustubh D. Dhole, Varun Gangal, Sebastian Gehrmann, Aadesh Gupta, Zhenhao Li, Saad Mahamood, Abinaya Mahendiran, Simon Mille, Ashish Shrivastava, Samson Tan, Tongshuang Wu, Jascha Sohl-Dickstein, Jinho D. Choi, Eduard Hovy, Ondrej Dusek, Sebastian Ruder, Sajant Anand, Nagender Aneja, Rabin Banjade, Lisa Barthe, Hanna Behnke, Ian Berlot-Attwell, Connor Boyle, Caroline Brun, Marco Antonio Sobrevilla Cabezudo, Samuel Cahyawijaya, Emile Chapuis, Wanxiang Che, Mukund Choudhary, Christian Clauss, Pierre Colombo, Filip Cornell, Gautier Dagan, Mayukh Das, Tanay Dixit, Thomas Dopierre, Paul-Alexis Dray, Suchitra Dubey, Tatiana Ekeinhor, Marco Di Giovanni, Tanya Goyal, Rishabh Gupta, Rishabh Gupta, Louanes Hamla, Sang Han, Fabrice Harel-Canada, Antoine Honore, Ishan Jindal, Przemysław K. Joniak, Denis Kleyko, Venelin Kovatchev, Kalpesh Krishna, Ashutosh Kumar, Stefan Langer, Seungjae Ryan Lee, Corey James Levinson, Hualou Liang, Kaizhao Liang, Zhexiong Liu, Andrey Lukyanenko, Vukosi Marivate, Gerard de Melo, Simon Meoni, Maxime Meyer, Afnan Mir, Nafise Sadat Moosavi, Niklas Muennighoff, Timothy Sum Hon Mun, Kenton Murray, Marcin Namysl, Maria Obedkova, Priti Oli, Nivranshu Pasricha, Jan Pfister, Richard Plant, Vinay Prabhu, Vasile Pais, Libo Qin, Shahab Raji, Pawan Kumar Rajpoot, Vikas Raunak, Roy Rinberg, Nicolas Roberts, Juan Diego Rodriguez, Claude Roux, Vasconcellos P. H. S., Ananya B. Sai, Robin M. Schmidt, Thomas Scialom, Tshephisho Sefara, Saqib N. Shamsi, Xudong Shen, Haoyue Shi, Yiwen Shi, Anna Shvets, Nick Siegel, Damien Sileo, Jamie Simon, Chandan Singh, Roman Sitelew, Priyank Soni, Taylor Sorensen, William Soto, Aman Srivastava, KV Aditya Srivatsa, Tony Sun, Mukund Varma T, A Tabassum, Fiona Anting Tan, Ryan Teehan, Mo Tiwari, Marie Tolkiehn, Athena Wang, Zijian Wang, Gloria Wang, Zijie J. Wang, Fuxuan Wei, Bryan Wilie, Genta Indra Winata, Xinyi Wu, Witold Wydmański, Tianbao Xie, Usama Yaseen, Michael A. Yee, Jing Zhang, and Yue Zhang. 2022. Nl-augmenter: A framework for task-sensitive natural language augmentation.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for pytorch.

Scott Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions.

Edward Ma. 2019. Nlp augmentation. https://github.com/makcedward/nlpaug.

John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp.

OpenAI. 2023. Gpt-4 technical report.

Zoe Papakipos and Joanna Bitton. 2022. Augly: Data augmentations for robustness.

Domagoj Pluščec and Jan Šnajder. 2023. Data augmentation for neural nlp.

Rebecca Qian, Candace Ross, Jude Fernandes, Eric Michael Smith, Douwe Kiela, and Adina Williams. 2022. Perturbation augmentation for fairer NLP. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9496–9521, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier.

Gabriele Sarti, Nils Feldhus, Ludwig Sickert, Oskar van der Wal, Malvina Nissim, and Arianna Bisazza. 2023. Inseq: An interpretability toolkit for sequence generation models. *ArXiv*, abs/2302.13942.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMLR.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Xiao Wang, Qin Liu, Tao Gui, Qi Zhang, Yicheng Zou, Xin Zhou, Jiacheng Ye, Yongxin Zhang, Rui Zheng, Zexiong Pang, Qinzhuo Wu, Zhengyan Li, Chong Zhang, Ruotian Ma, Zichu Fei, Ruijian Cai, Jun Zhao, Xingwu Hu, Zhiheng Yan, Yiding Tan, Yuan Hu, Qiyuan Bian, Zhihua Liu, Shan Qin, Bolin Zhu, Xiaoyu Xing, Jinlan Fu, Yue Zhang, Minlong Peng, Xiaoqing Zheng, Yaqian Zhou, Zhongyu Wei, Xipeng Qiu, and Xuanjing Huang. 2021. TextFlint: Unified multilingual robustness evaluation toolkit for natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 347–355, Online. Association for Computational Linguistics.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S Weld. 2021. Polyjuice: Generating counterfactuals for explaining, evaluating, and improving models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6707–6723.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Zixian Ma, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. OpenAttack: An open-source textual adversarial attack toolkit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

A Appendix

```python
from captum.attr import ShapleyValueSampling, LLMAttribution, TextTemplateFeature,
                                          ProductBaselines

svs = ShapleyValueSampling(model)
baselines = ProductBaselines(
    {
        ("name", "pronoun"): [("Sarah", "Her"), ("John", "His")],
        "city": ["Seattle", "Boston"],
        "state": ["WA", "MA"],
        "occupation": ["doctor", "engineer", "teacher", "technician", "plumber"],
    }
)

llm_attr = LLMAttribution(svs, tokenizer)

inp = TextTemplateFeature(
    "{name} lives in {city}, {state} and is a {occupation}. {pronoun} personal
                                              interests include",
    {"name":"Dave", "city": "Palm Coast", "state": "FL", "occupation":"lawyer", "
                                              pronoun":"His"},
    baselines=baselines,
)

attr_result = llm_attr.attribute(inp, target="playing golf, hiking, and cooking.")
```

Figure 6: Applying Captum for the model associations example

```python
from captum.attr import ShapleyValues, LLMAttribution, TextTemplateFeature

sv = ShapleyValues(model)
llm_attr = LLMAttribution(sv, tokenizer)

def prompt_fn(*examples):
    main_prompt = "Decide if the following movie review enclosed in quotes is
                                              Positive or Negative:\n'I really liked
                                               the Avengers, it had a captivating
                                              plot!'\nReply only Positive or
                                              Negative."
    subset = [elem for elem in examples if elem]
    if not subset:
        prompt = main_prompt
    else:
        prefix = "Here are some examples of movie reviews and classification of
                                                  whether they were Positive or
                                                  Negative:\n"
        prompt = prefix + "\n".join(subset) + "\n" + main_prompt
    return "[INST] " + prompt + "[/INST]"

input_examples = [
    "'The movie was ok, the actors weren't great' -> Negative",
    "'I loved it, it was an amazing story!' -> Positive",
    "'Total waste of time!!' -> Negative",
    "'Won't recommend' -> Negative",
]
inp = TextTemplateFeature(prompt_fn, input_examples)

attr_result = llm_attr.attribute(inp)
```

Figure 7: Applying Captum for the few-shot prompt example