Explaining CodeBERT's prediction for Vulnerability Detection using Markov chain and Integrated Gradient

Md Mahbubur Rahman, Shaila Sharmin,

December 2022

Abstract

Transformer-based models show significant performance in Natural Languages Processing (NLP) and Computer Vision (CV). CodeBERT is a recent transformer-based model that achieved state-of-the-art source code vulnerability detection performance. Although CodeBERT can predict whether an input code is vulnerable or not, there is no way to explain which features of the input source code are responsible for that prediction. Different explainability tools are proposed to compute the relevance of input tokens for the model decisions. However, none of them considers the information flow across layers of the transformers and provides the class discriminative explanation. In this project, we propose an explanation approach that leverages the information flow across layers of the codeBERT model using the idea of the Markov chain and the integrated gradient. Our model outperforms the baseline model by at least 2.57% top-10 accuracy. Our code and dataset are available at https://github.com/mahbubcseju/TransformerExplainability.

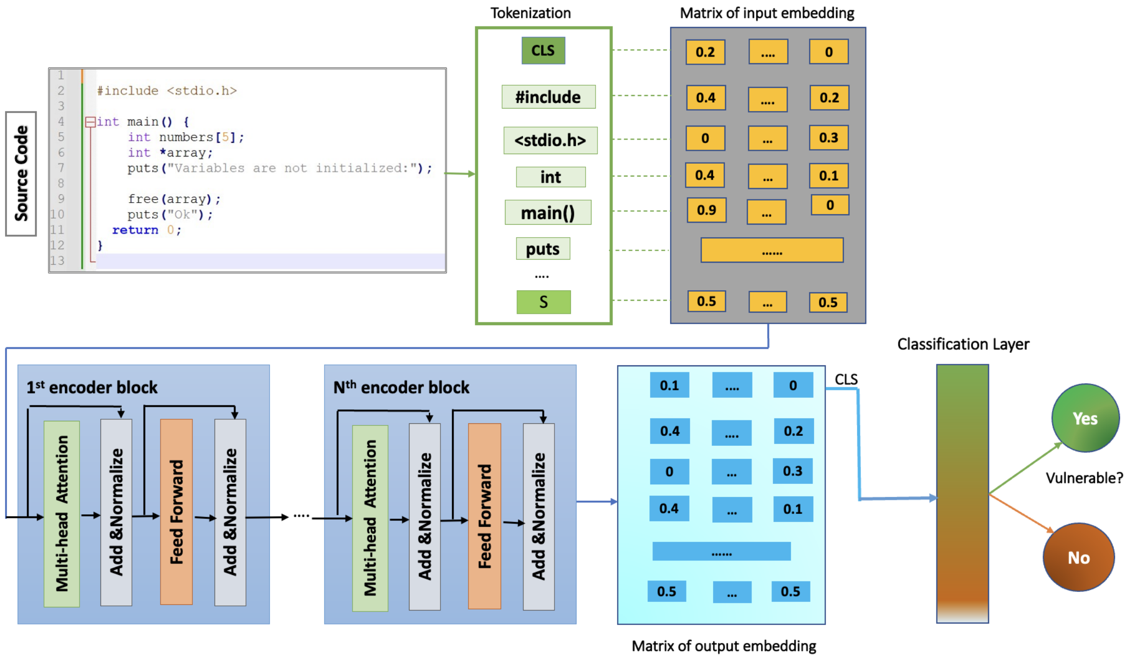
1 Introduction

Vulnerable software is a threat to any kind of software security. Vulnerability can be of different types ranging from privacy, and security to data leakage. Therefore, fixing vulnerable code chunks or identifying potential vulnerable portion of code is important for making any system secure. Deep learning has greatly enhanced software vulnerability detection in recent years. Some transformer-based models like CodeBERT [1] also achieved state-of-the-art performance in vulnerability detection. These models have a high level of classification accuracy, but their decision-making and prediction processes are unclear. It is impossible to determine whether the model use the proper features to make a prediction.

Explainability tools help us in this regard by giving us scores for each input token or feature and can explain the relevance of input features for the model decisions. The higher the score of a feature, the more important the feature is for that prediction. Based on this score, we can determine the score for each statement or line in the source code, which will enable us to identify the vulnerable lines in a function or section of code.

The most widely used method to explain Transformer-based method is to leverage the attention weights in the encoder blocks. The simplest approach to use attention weights for explainability is to use raw attention scores [2], [3] of the class token. However, Abnar and Zuidema [4] raised a concern that the contextual information from tokens becomes similar as going deeper into the model. This leads to an unreliable explanation of the information flow throughout the model if we use raw attention weights. To address this problem, the transition attention maps [5] method was proposed for Vision-Transformer that uses the information flow to explain which features the transformer focuses on to make the prediction. Though Vision-Transformers and CodeBERT use different architectures and APIs, the information flow process is same in both cases and the above concept can be used to explain the prediction of CodeBERT.

Following the lines of the above method, we continue to explore the information flow in CodeBERT. In this work, we propose a modified version of [5] that relates the information flow in the CodeBERT model to the Markov process, using the hidden states for tokens at each layer. These states start from an initial value and are recursively changing along with layers according to the CodeBERT's processing for tokens' representations. When a data sample passes through the attention blocks of the CodeBERT, traits are left in the computed attention scores. To track the traits and investigate the information flow, our approach considers these attention weights as transition matrices within the context of the Markov chain as they are naturally row-stochastic matrices, with each row summing to 1. Therefore, the proposed approach propagates the information from top to down and computes the relevance between high-level semantics and input features using transitions of states.



```c
#include <stdio.h>

int main() {
    int numbers[5];
    int *array;
    puts("Variables are not initialized:");

    free(array);
    puts("Ok");
  return 0;
}
```

Figure 1: Basics of CodeBERT encoder

Furthermore, to exhibit the class discriminative ability, the proposed approach combines the idea of attentions being transition matrices with Integrated Gradient [6] to assign the importance scores w.r.t. the predicted category to input features. Experiments result show that our approach outperform the baseline methods.

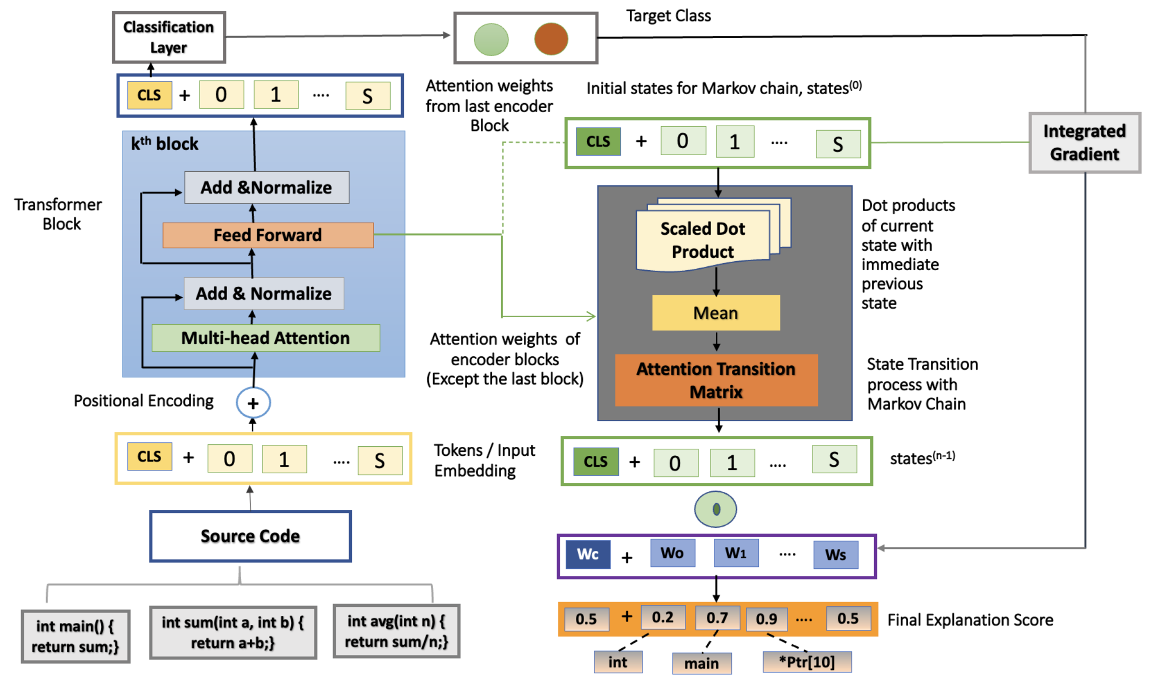
2 Background

2.1 CodeBERT for Vulnerability Detection

CodeBERT is a pre-trained model for multiple programming languages like Python, C/C++, Go, Java, etc. It is developed based on a transformer-based architecture named RoBERTa [7] and can be used for several down-streaming tasks such as clone detection, vulnerability detection, code search, etc.

For vulnerability detection, CodeBERT takes the whole input program as a flat input sequence and tokenizes the sequence. During tokenization, it adds two extra tokens; one is the [CLS] token which is added at the beginning of the sequence, and another is the [SEP] or [S] token, which is added at the end of the sequence. It then represents each token as a vector of *embed_dim* size where *embed_dim* is a hyper-parameter in CodeBERT. If the input sequence has T tokens, after converting each token into a vector of size *embed_dim*, it gets a matrix of size $(T \times embed\_dim)$. The token embedding then goes through N encoder blocks sequentially. The dimension of the input and output of each encoder block is the same, and the output of one encoder is the input of the next encoder. A specific encoder block is responsible for finding relationships between the input embedding. The initial block learns the basic relationship, but the more it goes through the encoder block, the more complex relation it learns.

The input is passed through a multi-head attention layer in each encoder block. The multi-head attention layer computes the attention multiple times in parallel with different weights and then concatenates these attentions together. The result of each of those parallel computations of attention is called a head. After normalizing the output of the multi-head attention, the result is sent to a feed-forward network. The normalized output is the output of the encoder block. After getting the output of the final encoder block, the output embedding of the [CLS] token is sent to a classification layer (multi-layer perceptron). Finally, the classification layer decides whether or not the input source code is vulnerable. The basics of codeBERT are described in the figure 1



Classification Layer
Target Class
CLS + 0 1 .... S
Attention weights from last encoder Block
Initial states for Markov chain, states(0)
Integrated Gradient
kth block
Transformer Block
Add &Normalize
Feed Forward
Add & Normalize
Multi-head Attention
Scaled Dot Product
Mean
Attention Transition Matrix
Dot products of current state with immediate previous state
State Transition process with Markov Chain
Attention weights of encoder blocks (Except the last block)
Positional Encoding
Tokens / Input Embedding
states(n-1)
Source Code
Wc + Wo W1 .... Ws
0.5 + 0.2 0.7 0.9 .... 0.5
Final Explanation Score
int
main
*Ptr[10]
int main() { return sum;}
int sum(int a, int b) { return a+b;}
int avg(int n) { return sum/n;}

Figure 2: Workflow of our proposed method

2.2 Explainability of Deep Learning

Deep learning models are like a black box, and it is difficult to say which features they emphasize on to make a prediction. Explainability tools make this thing transparent and help us to explain how much a token contributes to making the final decision. Several explainability techniques have been proposed in recent years, and among them, gradient-based methods are very popular. Some of them are discussed below:

- Gradient [8] is the first gradient-based explainability technique where the gradient of the output is calculated with respect to the input.
- Integrated Gradient [6] starts with a baseline that does not have any information about the sample. It then reaches to the input sample from the baseline by k steps by generating interpolated data in each step. During these steps, the interpolated data are passed through the classification model, and the output gradient is calculated with respect to the input. The summation of each of the token's gradients over k steps denotes the relevance/contribution score of the token.
- DeepLift [9] uses a reference input along with the actual input. It compares the activation of each neuron with the reference activation and propagates the difference (same as gradients) to learn the important features.

All the above approaches are model agnostic, and none of them focus on the transformers' internal mechanisms. Hence, this work proposes an approach that leverages the information flow inside transformers through the Markov chain. Our approach also leverages *Integrated Gradients* to remove noise introduced by the information flow.

3 Our Methodology

In this section, we briefly introduce our proposed methodology. Figure 2 demonstrates the workflow of our proposed method.

3.1 Markov Chain in CodeBERT

A markov chain is a stochastic process that describes a series of random variables where the probability of the current state depends solely on the previous state. On the other hand, we know that the output representation of each CodeBERT encoder block is input to the next block. As a result, we can connect the CodeBERT decision process with the Markov chain by considering the representation of output tokens at each encoder block as states. As the attention weights in each encoder block are in charge of computing the output representation from the input representation, we can construct the link between the input and output tokens by computing the transition of states using the attention weights. To sum up, we can use the representation of output tokens at each encoder block as states and compute the transition of states using the attention weights.

3.2 Transitions of States

The output embedding of the [CLS] token of the last encoder block is used for the classification task. Hence, some works use the attention of the [CLS] token from the last encoder block to provide explanations, which produces poor explainability. The reason could be the semantic gap between the lower and higher encoder blocks. Therefore, we can use the state transition process (mentioned in 3.1) to determine the relevance between the input tokens and model decisions.

To implement the idea, we follow the methods from [5] and initialize the states of the Markov Chain $states^{(0)}$ with the attention weights of the [class] tokens from the transformer's final block, as shown below:

$$states^{(0)} = E[AttentionWeights_{head}^{(B)}](CLS)$$

Here, E is the average across multiple heads in the attention layer and (CLS) is the index of the [CLS] token, and (B) is the index of the last block. The overall transition matrix is generated as follows:

$$states^{(i)} = \begin{cases} E[AttentionWeights_{head}^{(B)}](class), & \text{when } i = 0 \\ \frac{1}{2}states^{(i-1)} + \frac{1}{2}states^{(i-1)} \cdot E[AttentionWeights_{head}^{(B-i)}], & otherwise \end{cases} \tag{1}$$

We should mention that each encoder block has a corresponding $states^{(i)}$, which explains the information flow propagated to that block.

3.3 Integrated Gradient for Noise Removal

During the transition process, some irrelevant features or noises will be introduced, which need to be diminished by some method. So, we use integrated gradients to get the feature's relevance scores. The reason for using integrated gradients rather than only gradients is that the integration process effectively retains the relevant parts and reduces the gradient's self-induced noise [6]. Finally, by weighting the states of the tokens, $states^{(B)}$, with the integrated gradient's score, some noise that is created during the transition process will be reduced or eliminated [5]. By combining the state of the tokens with the integrated gradients, we also get the class discriminative explanations.

3.4 CodeBERT Explainability

In summary, we provide an explainability approach by leveraging the information flow inside CodeBERT. We consider the states of tokens at each encoder block as the information flow through that block and use that information flow as a Markov process. We compute the transition of states with the attention weights at each layer. Finally, to get the class discriminative explanation, we combine the integrated gradients with the states of the tokens. Thus, this approach exploits the information flow through the attention module and is able to explain the contributions of tokens with respect to the predictions. Algorithm 1 describes our approach.

Algorithm 1 Transformer Explainability

```
Input: Input function/program X
Output: Explainability score
1: $states \leftarrow E[AttentionWeights_{head}^{(B)}](CLS)$
2: for i = 1 to B do
3:     $AverageAttention \leftarrow E[AttentionWeights_{head}^{(B-i)}]$
4:     $states = \frac{1}{2}states \cdot AverageAttention + \frac{1}{2}states$
5: end for
6: $IntegratedGradientsScore = IntegratedGradient(X)$
7: $ExplanationScore \leftarrow IntegratedGradientsScore \bigodot states$
```

4 Experimental Setup

4.1 Dataset

We construct a new vulnerability dataset from Devign [10] and MSR [11]. In Devign and MSR datasets, functions are labeled as vulnerable or non-vulnerable. But for the vulnerable samples, they don't have any information about which lines are vulnerable within the function. To find out the vulnerable lines in the vulnerable functions, we follow the existing literature [12]. At first, we generate the `diff` information from the current commit and the next commit of the example functions. Then, we label a line as vulnerbale if it is removed or altered in the next commit (starting with '-' in `diff` files). If the `diff` information is empty for a sample, we don't consider that sample. To make our dataset balanced, we take around same amount of non-vulnerable data of vulnerable data. Finally, we divide the data into three different splits, train, test and validation by the ratio of 80:10:10. We use the train and validation data for fine-tuning the CodeBERT model and test data to evaluate our proposed explanation approach.

Class / split	Train	Test	Validation
Vulnerable	6927	865	865
Non Vulnerable	6869	859	859
Total	13796	1724	1724

Table 1: Data distribution over different splits.

4.2 Evaluation Process

We, at first, fine-tune CodeBERT with our training and validation data. During fine-tuning, we use CodeBERT's default settings. We don't perform any hyper-parameter tuning as our main focus is the evaluation of the explainability tools. After fine-tuning the CodeBERT, we run our method as well as some baseline methods using the test data. We compare our approach with strong and up-to-date baselines such as DeepLift, Gradients, and Integrated Gradients. These methods as well as our approach provide us the importance score for each token of the input sequence. It is difficult to evaluate any method based on the token's score. Therefore, following the existing literature [12], we calculate the score for each line of the input function/program from the token's score. The score of each line is the average of the scores of its tokens.

4.3 Evaluation Matrices

To evaluate our approach, we follow the existing literature [12] and use the following two matrices:

- **Top - 10 Accuracy:** It is the percentage of vulnerable functions where at least one vulnerable line appears in the top-10 rankings. The top-10 rankings are constructed by taking the top-10 scored lines from the sorted list of lines. Here, the lines are sorted by their scores in descending order.
- **Initial False Alarm (IFA):** It is the measure of average non-vulnerable lines that appears before the first vulnerable line in the sorted list of lines. Here, the lines are sorted by their scores in descending order.

We only use the vulnerable samples during using these matrices.

5 Experimental Results

Table 2 presents the evaluation results of our proposed method and the baselines. We can see that our proposed approach achieves 2.57% , 9.47% and 2.85% higher top-10 accuracy than Integrated Gradient, Gradients and DeepLift, respectively. Moreover, our approach improves significantly over the baseline methods regarding IFA score. To sum up, our approach outperforms all the three baseline methods.

Method	Top-10 Accuracy	Initial False Alarm
DeepLift	62.81%	9.01
Gradient	56.19%	9.43
Integrated Gradient	63.09%	8.98
Our Approach	**65.66%**	**8.25**

Table 2: The Top-10 Accuracy and IFA of our approach and three other methods. (↑) Higher Top10 Accuracy = Better, (↓) Lower IFA = Better.

6 Related Work

Deep learning models mainly depend on the important features to make a decision. Therefore, it is important to know how the information is changing/transmitting within the models to make a correct decision. Generally, methods based on gradients are applied to highlight the input parts relevant to the prediction. Smilkov et al. [13] proposed a method to remove the noisy features by adding noises to the input and then averaging the gradient. Other gradient-based models are DeepLIFT (Deep Learning Important FeaTures) [14], SHAP (Shapley Additive exPlanations) [15]. Several model-agnostic methods are also used for calculating important features like LIME [16] and Deep Taylor Decomposition [17] and Layer-Wise Relevance Propagation [18]. Although these gradient-based and model-agnostic methods can be applied in transformer models like CodeBERT, there are limited approaches specifically designed for the transformers. Hila et al. conducted [19] research on explainability for transformers using Taylor decomposition property. Tingyi et al. [20] proposed an approach to explain information flow inside vision transformers using the Markov chain. However, most transfer-based model interpretation was conducted in the computer vision field; there are few works done for NLP-based transformers. Besides, none of them considers the information flow across layers of the transformers. In this project, we propose to explore the information flow inside a transformer model named CodeBERT using the Markov chain and integrated gradient.

7 Concluding Remarks and Future Work

In this project, we suggest a class discriminative explainability method for the CodeBERT model. This method leverages Integrated Gradient and Markov chains to provide the explanation. The Markov chain is used to leverage the information flow inside the CodeBERT model, and the Integrated Gradient is used to get the class discriminative explanations. As shown by the experiment results, our approach outperforms other existing state-of-the-art methods.

Although we only tested this approach for vulnerability detection in source code, this can be used for any NLP-based down-streaming tasks which use transformer models. In the future, we will apply the approach using other datasets as well as Transformer-based architectures.

References

[1] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," *CoRR*, vol. abs/2002.08155, 2020. [Online]. Available: https://arxiv.org/abs/2002.08155

[2] A. Coenen, E. Reif, A. Yuan, B. Kim, A. Pearce, F. B. Viégas, and M. Wattenberg, "Visualizing and measuring the geometry of BERT," *CoRR*, vol. abs/1906.02715, 2019. [Online]. Available: http://arxiv.org/abs/1906.02715

[3] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of bert's attention," *CoRR*, vol. abs/1906.04341, 2019. [Online]. Available: http://arxiv.org/abs/1906.04341

[4] S. Abnar and W. H. Zuidema, "Quantifying attention flow in transformers," *CoRR*, vol. abs/2005.00928, 2020. [Online]. Available: https://arxiv.org/abs/2005.00928

[5] T. Yuan, X. Li, H. Xiong, H. Cao, and D. Dou, "Explaining information flow inside vision transformers using markov chain," in *eXplainable AI approaches for debugging and diagnosis.*, 2021. [Online]. Available: https://openreview.net/forum?id=TT-cf6QSDaQ

[6] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," *CoRR*, vol. abs/1703.01365, 2017. [Online]. Available: http://arxiv.org/abs/1703.01365

[7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[8] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013. [Online]. Available: https://arxiv.org/abs/1312.6034

[9] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *CoRR*, vol. abs/1704.02685, 2017. [Online]. Available: http://arxiv.org/abs/1704.02685

[10] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[11] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 508–512. [Online]. Available: https://doi.org/10.1145/3379597.3387501

[12] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 608–620.

[13] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.

[14] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning.* PMLR, 2017, pp. 3145–3153.

[15] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[17] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern recognition*, vol. 65, pp. 211–222, 2017.

[18] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[19] H. Chefer, S. Gur, and L. Wolf, "Transformer interpretability beyond attention visualization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 782–791.

[20] T. Yuan, X. Li, H. Xiong, H. Cao, and D. Dou, "Explaining information flow inside vision transformers using markov chain," in *eXplainable AI approaches for debugging and diagnosis.*, 2021.