GPUTextSHAP: a GPU-Accelerated Permutation SHAP for Textual Data

Henrique Margotte[1,2,*], Bruno H. Meyer[1,2], Carmem S. Hara[1,2], Aurora T. R. Pozo[1,2] and Wagner M. Nunan Zola[1,2]

[1]Departamento de Informática, Universidade Federal do Paraná (UFPR), Centro Politécnico, Curitiba, PR, Brazil
[2]Núcleo de Modelagem e Computação Científica, Universidade Federal do Paraná (UFPR), Brazil

Abstract

Explainable Artificial Intelligence (XAI) techniques such as SHAP provide valuable insights into model predictions, but their computational cost can limit applicability in some scenarios. This paper presents GPUTextSHAP, a GPU-accelerated implementation of the Permutation SHAP algorithm designed to efficiently generate token-level explanations for text classification models. The proposed approach exploits GPU parallelism and optimized memory usage to efficiently compute SHAP values across permutations. Experimental results show that GPUTextSHAP achieves a speedup of 118× compared to the Python implementation of the SHAP Permutation Explainer while preserving the same explanation procedure. To demonstrate its capabilities, we developed an interactive application that allows users to explore token-level explanations for sentiment classification predictions, highlighting the contribution of individual words to model outputs. The system illustrates how GPU acceleration can enable practical use of SHAP explanations in natural language processing tasks.

Keywords

Explainable Artificial Intelligence (XAI), Permutation SHAP, GPU Acceleration, Natural Language Explanations, Model-Agnostic

1. Introduction

Among the most common types of problems in the Artificial Intelligence (AI) field, those based on text, such as Natural Language Processing (NLP) tasks, have become increasingly prevalent, especially with the popularization of Large Language Models (LLMs). As in many other AI domains, the need for explanations for these models has grown as they are increasingly applied to critical areas such as law, economics, and politics. This trend has made the use of eXplainable AI (XAI) techniques more relevant.

SHapley Additive exPlanations (SHAP) [1] is a widely used model-agnostic XAI technique for estimating the contribution of input features to a machine learning (ML) model's prediction. However, computing SHAP values can be computationally expensive because it requires evaluating the model on an exponential number of feature coalitions. This limitation becomes particularly significant for high-dimensional inputs such as images [2] or long texts [3].

In the context of NLP tasks, the use of SHAP can become impractical due to its high computational cost. These problems often involve large text inputs composed of many tokens, which increases the number of coalitions and model inferences required to compute SHAP values. In addition, some models, such as LLMs, have high inference costs, further increasing the time required to generate explanations. In some cases, computing SHAP values can take hours for large models [3, 4].

Many works have proposed solutions to accelerate SHAP or reduce its computational cost [3, 5, 6, 7, 8]. Some approaches develop CUDA [9]-based algorithms to implement GPU-accelerated versions, such as

Late-breaking work, Demos and Doctoral Consortium, colocated with the 4th World Conference on eXplainable Artificial Intelligence: July 01–03, 2026, Fortaleza, Brazil

*Corresponding author.
hmargotte@inf.ufpr.br (H. Margotte); bhmeyer@inf.ufpr.br (B. H. Meyer); carmem@inf.ufpr.br (C. S. Hara); aurora@inf.ufpr.br (A. T. R. Pozo); wagner@inf.ufpr.br (W. M. N. Zola)
https://github.com/henriquemargotte (H. Margotte)
0000-0001-6480-5236 (H. Margotte); 0000-0001-6747-1155 (B. H. Meyer); 0000-0002-0674-9229 (C. S. Hara); 0000-0001-5808-3919 (A. T. R. Pozo); 0000-0002-2282-0398 (W. M. N. Zola)

GPUTreeSHAP [10] and implementations in the RAPIDS cuML package [11]. GPUs exploit massive parallelism by executing many similar small tasks across a large number of threads, enabling algorithms to run significantly faster than sequential methods or CPU-based parallelization. However, most existing GPU implementations are designed for structured data, such as tabular datasets, and are not directly suitable for unstructured data like text documents used in NLP tasks. Additionally, some approaches are model-specific, such as GPUTreeSHAP, which is limited to tree-based models.

This work proposes GPUTextSHAP, a GPU-accelerated, model-agnostic SHAP implementation designed for unstructured text data.[1] The approach is based on the permutation-based approximation of SHAP [12], similar to the Permutation Explainer available in the SHAP library [1] (hereafter referred to as the SHAP Permutation Explainer). A demonstration of the system is presented using an embedding-based Multi-Layer Perceptron (MLP), hereafter referred to as the EmbeddingMLP, applied to a sentiment analysis task. In the experimental scenario, GPUTextSHAP achieved up to 118× speedup compared to the SHAP Permutation Explainer under the same experimental conditions, both for the entire test dataset and on average per input, while producing similar results.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents an overview of the proposed system. Section 4 describes the experiments conducted. Section 5 details the demonstration setup. Finally, Section 6 concludes the paper and discusses future work.

2. Related Work

SHAP [1] is one of the most widely used XAI techniques. Based on Shapley values from cooperative game theory, SHAP defines SHAP values as a unified measure of feature importance for a ML model. These values are calculated by masking the input features across multiple coalitions and measuring how the presence or absence of each feature affects the model output compared to a background reference dataset. Due to the increasing computational complexity of the exact SHAP calculation for inputs with many features, several algorithms have been proposed to approximate SHAP values with lower computational cost. Permutation SHAP is an implementation based on an approximation of Shapley values [12]. The method generates permutations of the input features and iteratively applies a masking process to compute feature contributions. For a given permutation, the contribution of a feature i is estimated as the difference between the model output obtained using all features up to i and the output obtained using only the features preceding i. These estimations are computed across a set of random permutations, and the average contribution of each feature provides an approximation of its SHAP value.

The Python SHAP library [1] implements several SHAP variants, including SHAP Permutation Explainer. However, only one implementation applies GPU acceleration for SHAP value computation: GPUTreeSHAP [10]. This method is designed for tree-based ML models, such as Decision Trees and XGBoost. Although highly efficient, GPUTreeSHAP is limited to these model types and cannot be directly applied to other models, such as embedding-based or transformer-based architectures, which prevents its use as a generic black-box explanation method.

The RAPIDS cuML package [11], which provides CUDA-based implementations of various ML algorithms, includes three SHAP variants. These include a version of GPUTreeSHAP, which shares the same model limitations as the Python SHAP library, as well as implementations of Kernel SHAP and Permutation SHAP. However, according to the package documentation, these implementations are designed for tabular data inputs, which makes their direct use difficult for unstructured data such as text sentences and documents.

This work proposes GPUTextSHAP as an alternative CUDA-based implementation of Permutation SHAP designed for unstructured text data. The system is also model-agnostic, treating the ML model as a black box, which allows it to be applied to different models with minimal adaptations for GPU-based processing.

[1]The GPUTextSHAP implementation is publicly available at: https://github.com/henriquemargotte/gpu-text-shap

3. System Overview

The system proposed in this work consists of three main components: the training and preprocessing stage, described in Subsection 3.1; the GPUTextSHAP implementation in CUDA, presented in Subsection 3.2; and the model used in the experiments, referred to as the EmbeddingMLP, described in Subsection 3.3.

3.1. Training and Preprocessing

SHAP explains how a given input of an ML model produces a specific output, therefore, a trained model is required. Although GPUTextSHAP treats the model as a black box, this implementation requires access to the model parameters in order to replicate the inference step on the GPU. In the case of the EmbeddingMLP used in this demonstration, the model was trained in Python using PyTorch [13]. After training, the embedding matrix and the MLP weights and biases were exported so they could be used during the inference stage implemented on the GPU.

The same preprocessing steps must be applied both to the training data used by the EmbeddingMLP and to the input samples explained by SHAP. For this demonstration, a word-level tokenization using PyTorch's `basic_english` tokenizer was applied to the data, segmenting the input text into tokens. A vocabulary was then constructed from the tokens observed in the training dataset, mapping each token to a numerical index. Finally, the tokenized inputs were padded or truncated to a fixed maximum sequence length so that they could be processed as fixed-size inputs by the model.

3.2. GPUTextSHAP

After receiving the tokenized input and the model parameters, the algorithm launches a number of GPU thread blocks equal to twice the number of permutations defined. The input and the model parameters are copied to the shared memory of each thread block. For each pair of GPU thread blocks, a random permutation of the input indices is generated, where one block processes the permutation in the forward direction while the other processes it in reverse order.

Inside each GPU thread block, the permuted index vector is iterated sequentially, adding one feature at a time to an input buffer. The buffer is then used to evaluate the black-box model (EmbeddingMLP) using the currently active features, and the resulting output is stored in an output buffer. After the outputs for all iterations are computed, the Permutation SHAP approximation is applied. In this step, the marginal contribution of feature i is computed as the difference between the model output at iteration i and the output at iteration $i-1$. Once all GPU thread blocks complete execution, the SHAP values are obtained by averaging the marginal contributions across permutations. Algorithm 1 illustrates the programming logic executed inside a GPU thread block, while Figure 1 presents a diagram of the system architecture, including the preprocessing and training stages.

The performance improvement of GPUTextSHAP is primarily achieved through the parallel execution model of the GPU and the optimized use of on-chip memory. Each permutation pair is processed concurrently by separate GPU thread blocks, allowing multiple marginal contribution computations to be performed simultaneously. In addition, the permutations and intermediate data structures are stored in shared memory within each block, reducing the need for repeated access to global memory and improving memory access latency. These design choices allow the explanation process to efficiently evaluate a large number of permutations while minimizing memory transfer overhead.

3.3. EmbeddingMLP Classifier

For this demonstration experiment, the inference of an EmbeddingMLP classifier was implemented in CUDA. The algorithm receives as input a tokenized vector, the hyperparameters and weights of the MLP, and the embedding matrix, returning as output the logit for class 0. Although the system has access to the model weights and the embedding matrix, the model interacts with SHAP only through

Algorithm 1 Permutation SHAP Computation inside a GPU thread Block

Require: Input token indices $x[0..seq\_len-1]$, black-box model f
Require: Boolean $backward$
Ensure: Marginal feature contributions

Define a random permutation $perm[0..seq\_len-1]$ of the input indices
Initialize empty $input\_buffer$
Initialize $outputs[0..seq\_len-1]$
for $i = 0$ to $seq\_len - 1$ **do**
    if $backward$ **then**
        Add $x[perm[seq\_len - 1 - i]]$ to $input\_buffer$
    else
        Add $x[perm[i]]$ to $input\_buffer$
    end if
    $outputs[i] \leftarrow f(input\_buffer)$ ▷ black-box model
end for
for $i = seq\_len - 1$ downto 1 **do**
    $outputs[i] \leftarrow outputs[i] - outputs[i-1]$
end for
for $i = 0$ to $seq\_len - 1$ **do**
    if $backward$ **then**
        $feature \leftarrow perm[seq\_len - 1 - i]$
    else
        $feature \leftarrow perm[i]$
    end if
    Add $outputs[i]$ to SHAP accumulator of $feature$
end for
Store partial SHAP contributions for aggregation across permutations

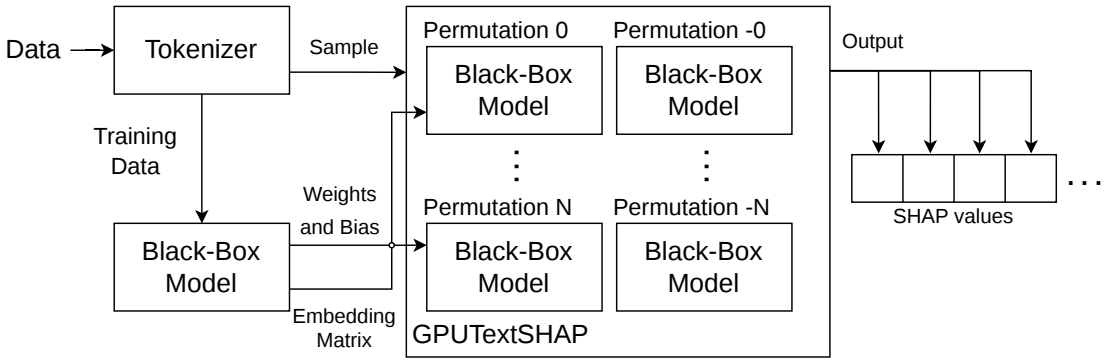

Figure 1: Workflow of SHAP value computation in GPUTextSHAP. Input text is tokenized and used to train the black-box model. During explanation, tokenized samples are provided to GPUTextSHAP, which generates permutations of the input features. Each permutation is evaluated in forward (0 to N) and reversed (-0 to $-N$) order by paired GPU thread blocks using the trained model parameters. After processing all permutations, SHAP values are computed for each token.

the input sample and the resulting logit, preserving the black-box characteristic of SHAP while enabling several time optimization strategies.

The first step consists of computing the embedding vector. An embedding vector represents the input in a lower-dimensional latent semantic space [14]. The embedding matrix contains the representation of each token, in this case words, in this semantic space and is used to map each token to its corresponding embedding vector. The embedding of the full input sequence is computed as the mean of the embedding

vectors of its tokens. As an optimization step, the token embedding vectors are aggregated incrementally into the input embedding vector during the GPUTextSHAP iteration described in Subsection 3.2. The embedding computation is also parallelized on the GPU, with each thread computing one or more dimensions of the vector simultaneously, as illustrated in Algorithm 2.

Algorithm 2 Parallel Incremental Embedding Computation

Require: Token sequence $tokens[0..seq\_len - 1]$
Require: Embedding matrix E
Require: Current permutation index pos
Ensure: Updated pooled embedding vector $embed\_vec$
1: $tx \leftarrow$ thread index
2: $stride \leftarrow$ number of threads in block
3: $token \leftarrow tokens[pos]$ ▷ token added at current permutation step
4: $token\_embedding \leftarrow E[token]$
5: **for** $d = tx$ to $embed\_dim - 1$ step $stride$ **do** ▷ Parallel accumulation across embedding dimensions
6: $\quad embedding\_sum[d] \leftarrow embedding\_sum[d] + token\_embedding[d]$
7: **end for**
8: **for** $d = tx$ to $embed\_dim - 1$ step $stride$ **do** ▷ Mean pooling of included tokens
9: $\quad embed\_vec[d] \leftarrow embedding\_sum[d]/current\_tokens$
10: **end for**

Once the embedding vector is computed, it is processed by a MLP. Each layer of the network consists of a fully connected set of neurons, where each neuron computes a weighted sum of the input values and adds a bias term. In the GPU implementation, each thread in a block is responsible for computing one or more neurons of the layer. The resulting activation is then passed through the ReLU activation function, which sets negative values to zero, except in the final layer where no activation function is applied. After all neurons in a layer are computed, the outputs are synchronized and used as input to the next layer. The output of the final layer corresponds to the model logits, which are used as the prediction score for the explanation method.

Together, the incremental embedding computation and the parallel MLP inference allow GPUTextSHAP to efficiently evaluate the model output for each permutation step directly on the GPU.

4. Experimental Test Case

To showcase and evaluate the applicability of the system, an experimental test case was conducted using the IMDB dataset [15]. The task consists of a binary sentiment analysis classification problem on movie reviews, divided into positive and negative classes. The EmbeddingMLP model was implemented in Python using the PyTorch library [13]. The model uses a word embedding layer with an embedding size of 64, followed by a fully connected MLP with four layers of 64 neurons and a Dropout rate of 50%.

The model was trained using 2000 random examples from the IMDB dataset for 20 epochs, applying early stopping with a patience of 5. The training used a learning rate of 0.001, weight decay of 0, and a batch size of 64. The input text was tokenized using the word-level `basic_english` tokenizer and padded or truncated to a maximum sequence length of 128 tokens. The resulting model achieved an accuracy of 72% on a test dataset composed of 256 random samples. These hyperparameters were defined empirically, as the model serves only as an illustrative example and does not aim to represent a state-of-the-art approach.

For comparison, both the SHAP Permutation Explainer [1], invoked through `shap.Explainer` with the `algorithm="permutation"` option, and the proposed GPUTextSHAP implementation were executed on the 256 samples of the test dataset, using 500 permutations in both cases. The processing time was measured by isolating the explanation computation step for each input sample, which includes the execution of all permutations and the corresponding model evaluations. In other words, the measured

time accounts for the complete explanation pipeline, including the generation of masked inputs, the embedding lookup for each token, and the inference of the MLP classifier for every permutation sample, ensuring that both implementations were evaluated under the same explanation workload.

The experiments were conducted on an Intel Xeon Silver 4314 @ 2.40GHz processor with 16 cores, 32GB of RAM, and an NVIDIA RTX A4500 GPU (Ampere GPU architecture) with 56 multiprocessors and 7168 CUDA cores. The operating system is Linux Ubuntu 20.04.6 LTS, and all implementations were compiled using CUDA version 12.4. The GPU kernel was configured with 128 threads per block.

The results of the experiments are summarized in Table 1. The SHAP Permutation Explainer required 599.17 seconds to process the 256 test samples, corresponding to approximately 2.34 seconds per sample. In contrast, GPUTextSHAP processed the same samples in 5.08 seconds, corresponding to approximately 0.02 seconds per sample. This improvement is largely due to the GPU implementation, which exploits parallel execution across permutations and reduces memory access latency through the use of shared memory, resulting in a speedup of 118× in this experiment.

Table 1
Total execution time for 256 test samples from the IDMB dataset and average time per sample for each SHAP implementation. Speedup is reported relative to the SHAP Permutation Explainer.

SHAP Implementation	Total time (s)	Average time (s/sample)	Speedup
SHAP Permutation Explainer	599.17	2.34	1×
GPUTextSHAP	**5.08**	**0.02**	**118**×

Some samples from the dataset were arbitrarily selected and the SHAP values produced by GPU-TextSHAP were analyzed, demonstrating promising results, with an example presented in Section 5. Since the model achieved an accuracy of only 72%, it is also expected that the feature importance learned by the model do not always align with the true semantic sentiment of the text.

5. Demonstration Scenario

To demonstrate the results produced by GPUTextSHAP, an application was developed using the same experimental scenario described in the test case of Section 4. The system allows users to search and select samples from the test dataset and interactively explore explanations for different reviews. For each selected sample, the interface displays the importance of each token individually.

To facilitate the interpretation of the results, two coloring methods, Continuous and Discrete, were implemented to indicate the importance of each token based on its SHAP value. In both methods, red represents a contribution toward the negative class, blue represents a contribution toward the positive class, and white represents neutral impact. The *Continuous* method presents a continuous color scale based on the raw SHAP values, as illustrated in Figure 2. The *Discrete* method, shown in Figure 3, divides the scale into five levels of impact: Very Positive, Positive, Neutral, Negative, and Very Negative. Figure 4 presents the visualization of the same sample using both coloring methods, highlighting the positive importance of the word “funniest” and the negative importance of the word “worst”.

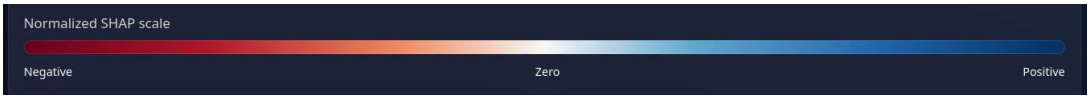

Figure 2: Color scale used in the Continuous coloring method. Red indicates contributions toward the negative class, blue toward the positive class, and white represents neutral impact (SHAP value near zero).

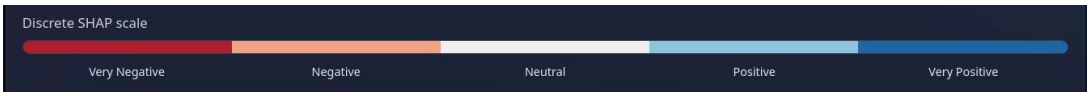

Figure 3: Color scale used in the Discrete coloring method, grouping SHAP values into five levels of impact: Very Negative (dark red), Negative (light red), Neutral (white), Positive (light blue), and Very Positive (dark blue).

(a) Continuous (b) Discrete

Figure 4: Example SHAP explanation for the same sample using the Continuous (left) and Discrete (right) coloring methods, highlighting token-level contributions to the model prediction.

6. Conclusion

This work presented GPUTextSHAP, a GPU-accelerated implementation of the Permutation SHAP algorithm designed to efficiently generate token-level explanations for text classification models. By leveraging GPU parallelization and optimized memory usage, the proposed approach significantly reduces the computational cost of SHAP explanations. Experimental results demonstrated a speedup of 118× compared to the Python implementation of the SHAP Permutation Explainer while maintaining the same explanation procedure. Additionally, an interactive application was developed to demonstrate the explanations produced by GPUTextSHAP, allowing users to explore token-level contributions for sentiment classification predictions.

Future work will evaluate GPUTextSHAP on additional datasets and natural language processing tasks, as well as with different neural architectures. Further comparisons with other SHAP variants and explainable AI techniques will be conducted to better assess the strengths and limitations of the proposed approach. We also plan to explore performance optimization through different GPU configurations, including the number of threads per block, and analyze the impact of parameters such as the number of permutations, embedding dimensions, and sequence lengths. Finally, future studies will investigate the quality and faithfulness of the generated explanations using established XAI validation methods.

Acknowledgments

This study was partially supported by Coordination for the Improvement of Higher Education Personnel (CAPES) - Program of Academic Excellence (PROEX), and by the National Council for Scientific and Technological Development (CNPq), process 408432/2024-1.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check, Paraphrase and reword. After using the tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017, pp. 1–10.

[2] X. Li, Y. Zhou, N. C. Dvornek, Y. Gu, P. Ventola, J. S. Duncan, Efficient shapley explanation for features importance estimation under uncertainty, in: Medical Image Computing and Computer Assisted Intervention – MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part I, Springer-Verlag, Berlin, Heidelberg, 2020, p. 792–801.

[3] J. Enouen, H. Nakhost, S. Ebrahimi, S. Arik, Y. Liu, T. Pfister, TextGenSHAP: Scalable post-hoc explanations in text generation with long documents, in: L.-W. Ku, A. Martins, V. Srikumar (Eds.), Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 13984–14011.

[4] Y. Rychener, X. Renard, D. Seddah, P. Frossard, M. Detyniecki, On the granularity of explanations in model agnostic nlp interpretability, in: I. Koprinska, et al. (Eds.), Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Springer Nature Switzerland, Cham, 2023, pp. 498–512.

[5] J. Enouen, Y. Liu, Instashap: Interpretable additive models explain shapley values instantly, 2025. arXiv:2502.14177.

[6] N. Jethani, M. Sudarshan, I. C. Covert, S.-I. Lee, R. Ranganath, FastSHAP: Real-time shapley value estimation, in: International Conference on Learning Representations, 2022, pp. 1–23.

[7] M. David, W. Jones, H. Horn, Enhancing shap with multi-core parallelization and distributed computation, SMU Data Science Review 8 (2024) 1–24.

[8] L. H. B. Olsen, M. Jullum, Improving the weighting strategy in kernelshap, in: R. Guidotti, U. Schmid, L. Longo (Eds.), Explainable Artificial Intelligence, Springer Nature Switzerland, Cham, 2026, pp. 194–218.

[9] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with cuda, in: ACM SIGGRAPH 2008 Classes, SIGGRAPH '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 1–14.

[10] R. Mitchell, E. Frank, G. Holmes, Gputreeshap: massively parallel exact calculation of shap scores for tree ensembles, PeerJ Computer Science 8 (2022) e880.

[11] S. Raschka, J. Patterson, C. Nolet, Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence, Information 11 (2020) 1–44.

[12] E. Strumbelj, I. Kononenko, An efficient explanation of individual classifications using game theory 11 (2010) 1–18.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, Curran Associates Inc., Red Hook, NY, USA, 2019, pp. 1–12.

[14] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, L. He, A survey on text classification: From traditional to deep learning 13 (2022) 1–41.

[15] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: D. Lin, Y. Matsumoto, R. Mihalcea (Eds.), Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA, 2011, pp. 142–150.