ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20231218001216
Original scientific paper

Tokenization and Memory Optimization for Reducing GPU Load in NLP Deep Learning Models

Dejan DODIĆ*, Dušan REGODIĆ

Abstract: In the current landscape of advanced natural language processing (NLP), managing GPU memory effectively is crucial. This paper delves into new tokenization methods and data handling to enhance NLP model efficiency, focusing on avoiding "CUDA out of memory" errors. It examines how sophisticated tokenization and managing text lengths in large datasets can boost model performance. These insights are vital for optimizing resources and scaling NLP models, especially with limited GPU memory. The paper also contextualizes NLP challenges, underlining the significance of memory optimization amidst growing language model complexities. It reviews key NLP technologies, including transformer models, and addresses their memory optimization challenges. Moreover, it underscores the paper's role in developing innovative techniques for more effective memory optimization, linking it to ongoing research and trends in NLP. This work aims to progress natural language processing methods and make AI technologies more accessible.

Keywords: data tokenization; deep learning; cuda out of memory; gpu memory optimization; machine learning; natural language processing (nlp)

1 INTRODUCTION

Deep learning has dramatically transformed the field of natural language processing (NLP), enhancing the ability of computer systems to understand and generate linguistic patterns [1, 2]. At the heart of this transformation lies the process of tokenization, which converts textual data into tokens the basic units that deep learning models can efficiently process [3, 4]. This paper explores advanced methods of tokenization and data management with the aim of optimizing GPU memory usage and avoiding errors such as "CUDA out of memory", which often occur when working with large NLP datasets [5].

The research forms the basis of various methodologies in NLP by combining advanced tokenization techniques with machine learning-driven memory optimization algorithms [6-8]. This integrated approach differs from conventional methods by simultaneously addressing the dual challenges of tokenization and memory allocation [9]. Specifically, it introduces new tokenization features that dynamically adjust the length of token sequences in response to memory load, effectively mitigating "CUDA out of memory" errors [10, 11]. Also, a customized memory prediction algorithm has been implemented, capable of estimating the memory needs of models in real-time, ensuring efficient GPU usage [5, 12].

In this transformation, different deep learning architectures have emerged, each with unique advantages and challenges in memory optimization [6, 13, 14]. This paper provides an overview of these architectures, including transformers and convolutional neural networks (CNNs), which are widely used in NLP applications [6]. It discusses the specific memory challenges associated with these architectures and how this research addresses these issues through innovative tokenization and memory optimization strategies [9].

NLP models, especially those based on sophisticated architectures such as transformers and CNNs, face the challenge of processing text in a way that allows for accurate learning and language generation while efficiently optimizing memory [11]. Managing the size of textual sequences and careful data manipulation during tokenization are key to achieving high model performance, particularly when working with large datasets [10]. This paper investigates how different tokenization techniques and smart memory optimization can contribute to more efficient training and evaluation of these models, thereby reducing the load on GPU resources [12].

The application of tokenization functions with carefully selected tokenization parameters is a key aspect of this research, contributing to more efficient management of memory resources and reducing the likelihood of exceeding memory capacity [10]. The technique of randomly sampling a smaller number of samples from the validation set is considered a strategy for reducing memory demands during model evaluation, allowing for analysis on a representative part of the dataset without overloading memory [5].

Through experimental evaluation, this paper aims to identify and analyse the most effective approaches to tokenization and data management that not only improve the accuracy and efficiency of deep learning models for NLP but also ensure scalability and efficiency in environments with limited resources. The results of this research provide valuable insights into optimal strategies for data and memory optimization, which are key to developing more efficient NLP systems [12].

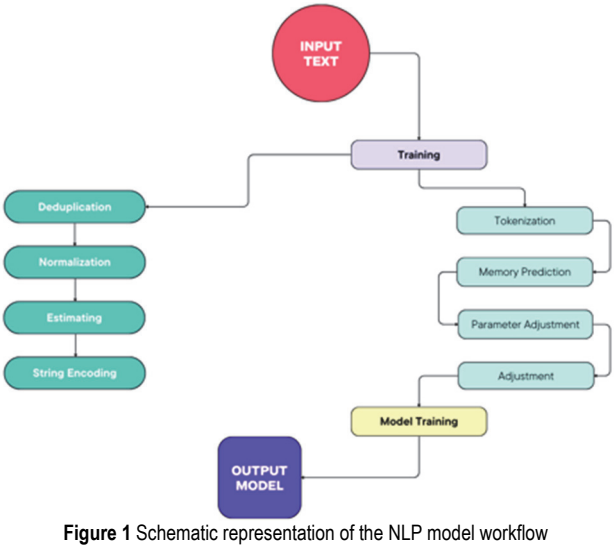

Figure 1 Schematic representation of the NLP model workflow

Fig. 1 shows the workflow of a newly developed NLP model with an innovative approach to tokenization and memory prediction, which enables more efficient model training and parameterization.

2 NEW HORIZONS IN MEMORY MANAGEMENT

In light of the continuous advancement in natural language processing (NLP), this paper aims to deepen the understanding of tokenization and data manipulation through the prism of deep learning [2]. Its primary objective is the development and application of strategies for more efficient memory management during the processing of extensive textual data sets, with a focus on innovative random sampling from the validation set.

This research relies on the analysis of the combined use of tokenization and advanced data manipulation, considering their impact on GPU memory optimization [15, 5]. Various configurations of the tokenize function, including aspects such as truncation = True, padding = "longest", and max length, are studied in detail to examine their effect on model performance and memory economy [16, 17]. This fundamental analysis provides insight into key aspects of tokenization, enabling the development of more advanced methods for optimizing memory resources.

Also, this paper encompasses a comparative analysis of different NLP architectures, such as transformers and CNNs, exploring their specific memory needs and how adjusting tokenization strategies can contribute to more efficient memory use [18]. Through experimental research, special attention is given to the impact of tokenization configurations on memory optimization, promoting the development of specific strategies to enhance model efficiency [5, 19].

The impact of tokenization configurations on performance and memory usage is experimentally investigated, contributing to the development of model-specific strategies for more efficient resource use. A key aspect of this approach is random sampling from the validation set, using the following formula to optimize the number of samples within memory constraints:

$$\text{Sample Size} = \frac{\text{Total Validation set size}}{\text{GPU memory limit} / \text{Memory per Sample}} \quad (1)$$

The central innovation of this paper, random sampling, utilizes Eq. (1) to determine the optimal number of samples that can be processed within memory constraints. Eq. (1) enables precise balancing between model performance and resource limitations, thereby optimizing the efficiency and scalability of NLP systems. This approach ensures more efficient model evaluations without compromising result integrity, contributing to a better understanding and optimization of the validation process.

In this paper, visual representations, such as Fig. 2, illustrate how strategies for reducing memory consumption adapt to GPU memory constraints. Fig. 2 visually demonstrates the efficiency of this approach to random sampling, clearly showing how sample size optimization can significantly improve model performance.

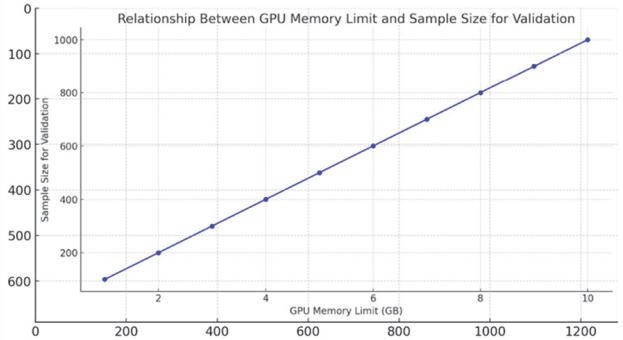

Figure 2 Relationship between GPU memory limit and sample size for validation in NLP models

Table 1 Validation sample size as a function of GPU memory limit

GPU memory limit / GB	Sample size for validation
2	200
4	400
6	600
8	800
10	1000

Tab. 1 contains a comparative analysis of the efficiency of different tokenization strategies and their impact on memory use. Tab. 1 details how specific configurations, such as truncation = True and max_length, contribute to memory optimization, providing important insights that support the conclusions on the efficiency of the proposed methods.

Through rigorous analysis and experimental validation, this paper significantly contributes to a broader understanding of how innovative memory management strategies can revolutionize the field of NLP, highlighting the importance of developing more efficient and adaptable models for future research.

3 INTEGRATING MACHINE LEARNING AND DEEP LEARNING

This research employs a multidisciplinary approach that merges theoretical analysis with cutting-edge experimental methodologies, including the use of the latest software solutions. The primary research idea is the integration of tokenization techniques and data manipulation within deep learning for natural language processing (NLP), with a new focus on applying machine learning techniques to predict and optimize memory usage [8]. Advanced analytics addresses key challenges such as "CUDA out of memory" errors not just through analysis but also by predicting memory needs during random sampling from the validation set.

The validation set is a critical element in the training and testing process of deep learning models, allowing for the evaluation of model performance before the final assessment on the test set. Using machine learning techniques for predicting and optimizing memory usage, this approach allows for fine-tuning the model to maximize its accuracy and minimize memory consumption, thus avoiding "CUDA out of memory" errors during validation [5].

To clearly illustrate this innovative approach, Fig. 3 presents a flowchart depicting the step-by-step process of this machine learning-based method. Fig. 3 starts with the initial tokenization of data, followed by the application of

predictive models to estimate memory needs, and culminates in the dynamic allocation of memory resources during the data processing phase.

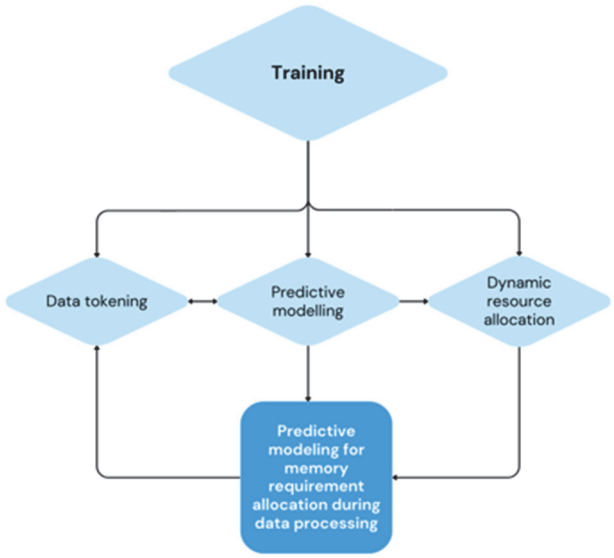

Figure 3 Flowchart of machine learning-driven dynamic memory allocation process in NLP

Applying machine learning to dynamic memory allocation during tokenization predicts memory needs based on parameters such as token sequence length (X), batch size (Y), and model complexity (Z). Predictive models use these parameters to estimate and adjust the amount of memory needed to process data in real-time, enabling more efficient use of GPU resources and reducing the likelihood of "CUDA out of memory" errors [6].

For demonstrating the efficacy of this approach, a comparative analysis was conducted against conventional tokenization and memory optimization techniques. The findings revealed that this method, which employs machine learning-based techniques for predictive modeling and dynamic memory optimization, not only maintains high model performance but also significantly reduces memory consumption [5].

Figure 4 Comparison of methods

Fig. 4 provides a visual comparison between traditional methods and this advanced machine-learning-based approach in terms of reducing memory usage while maintaining model performance. The blue bars in Fig. 4 indicate the percentage decrease in memory consumption achieved by this method, whereas the red line with markers represents model accuracy. This figure effectively illustrates how this approach minimizes memory usage without compromising model performance, highlighting its key advantage.

These experiments utilized the GIGABYTE GeForce RTX 3070 Gaming OC 8G graphics card with 8 GB of GDDR6 memory. This setup enabled efficient processing and utilization of advanced deep learning models without encountering the "CUDA out of memory" error.

Also, it is essential to consider the advantages and disadvantages of various NLP model architectures, including transformers and convolutional neural networks (CNNs). Transformers, known for their ability to process large text segments by considering contextual relationships, have demonstrated exceptional efficacy across various NLP tasks [8]. However, they often require significant memory and computational resources, which can pose challenges for implementation in resource-constrained hardware environments. On the other hand, while CNNs are effective in processing sequential data and identifying patterns within large datasets, they may not always be ideal for understanding complex linguistic structures in textual data [13]. This paper underscores the importance of carefully selecting the model based on project-specific requirements and available resources, which is crucial for maximizing efficiency and effectiveness in natural language processing.

This research includes a detailed analysis and experimental comparison of memory usage among different NLP model architectures, such as transformers and CNNs. The practical implementation of this research utilizes the Python programming language, alongside libraries such as Hugging Face Transformers and Optuna, with a novel integration of a machine learning framework specifically designed for real-time analysis and adjustment of memory usage [8].

The tokenize function is pivotal in the tokenization process of input textual data. This function has been modified to efficiently filter out blank text or text containing only white space before actual tokenization. This optimization prevents scenarios where empty texts are processed through tokenization, further optimizing memory usage. Eq. (2) is employed to estimate memory requirements for tokenization, considering the average token size and the total number of tokens in the batch:

$$Memory\ Required = Avg\ Token\ Size \times Total\ Tokens \qquad (2)$$

By applying Eq. (2), where the Average Token Size is multiplied by the Total Number of Tokens in a Batch, is determined the total Memory Required to store all tokens. This consideration is particularly vital in the realm of deep learning and NLP, where efficient memory optimization can greatly influence model performance and scalability. Employing Eq. (2) allows researchers and engineers to better plan and optimize memory usage, especially when limited by hardware resources (GPU) or when working with large data sets.

Fig. 5 demonstrates how memory requirements vary depending on the total number of tokens in a batch, with various assumed average token sizes. This illustration is especially relevant for scenarios requiring the processing

of large data sets under resource-constrained conditions, such as GPUs with limited memory.

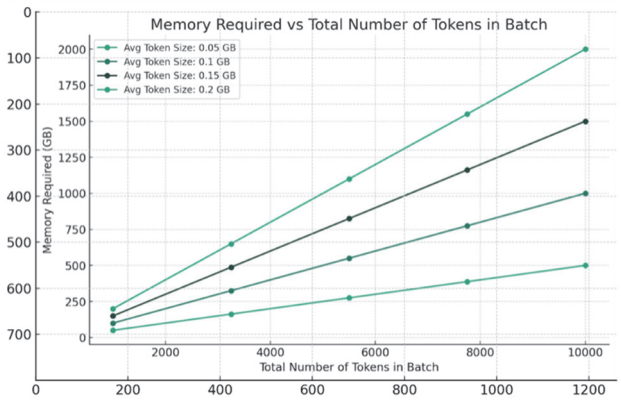

Figure 5 Memory requirement based on average token size and total number of tokens in a batch

Table 2 Estimated memory requirements by batch size and average token size

Total Number of Tokens in Batch	Memory Required for Avg Token Size: 0.05 GB / GB	Memory Required for Avg Token Size: 0.1 GB / GB	Memory Required for Avg Token Size: 0.15 GB / GB	Memory Required for Avg Token Size: 0.2 GB / GB
2000	100	200	300	400
4000	200	400	600	800
6000	300	600	900	1200
8000	400	800	1200	1600
10000	500	1000	1500	2000

Tab. 2 elaborates on the extrapolated memory requirements across different batch sizes, reflecting the necessary memory to process the specified tokens, depending on their average size.

In the context of practical application, on the last page of this paper, Pseudocode 1 for model optimization is presented. The tokenize_function is a crucial element for processing textual data into tokens. This code segment ensures that textual data is appropriately handled before tokenization. Initially, it removes empty texts and whitespaces, ensuring only valid texts are processed. Following that, it applies tokenization using the Hugging Face tokenizer, considering specific parameters like truncation, padding, special tokens mask, and max_length. In case of tokenization errors, it guarantees to return an empty set to prevent further processing of invalid data [8].

To establish an evaluation target, the create_objective function is employed for hyperparameter optimization during tuning. This function uses the Optuna library to explore optimal values for the max_length parameter to minimize the model's loss function. After data processing, part of the code focuses on randomly sampling a smaller subset of data from the validation set to reduce memory consumption during model evaluation [2].

This approach, which encompasses theoretical research and practical application with the integration of advanced machine learning techniques, stands out as an effective memory optimization strategy in NLP. It significantly contributes to the development of more sophisticated deep learning models, especially relevant for memory-intensive tasks in NLP, such as language translation, sentiment analysis and text summarization [8]. The task of this paper is the approach's ability to optimize memory usage, which is critical for performance and scalability, especially in systems with limited GPU memory resources.

4 OVERCOMING MEMORY CHALLENGES FOR ADVANCED NLP MODELS

The selection of a random sampling approach with a smaller number of instances from the validation set was motivated by the desire to enhance memory efficiency when processing large datasets in NLP tasks. This approach was inspired by findings in works [3, 15], where it was demonstrated that selective validation could significantly reduce memory requirements without compromising model performance. This methodology diverges from traditional full dataset validation approaches, offering better scalability and adaptability to various dataset sizes.

This approach builds on previous research highlighting the importance of memory optimization in natural language processing, introducing an innovative perspective through random sampling. Similar approaches discussed in papers [3, 15] explored alternative methods to reduce memory load, but this method stands out for its efficiency and applicability across a broad spectrum of NLP models.

Compared to traditional validation methods using the entire dataset, this model shows better memory optimization without significant loss in prediction accuracy. The paper [14] achieves similar memory use efficiency; however, it lacks the flexibility of this approach in dealing with different dataset sizes.

The main limitations of this approach concern the challenge of selecting an optimal number of samples representative of the entire dataset, especially for extremely large datasets. There is a risk of reduced model generalization if the sample selection is not adequately balanced. This paper proposes the development of more advanced sample selection algorithms that take into account dataset specifics, as suggested in paper [5], to overcome this limitation.

To enhance the transparency and rigor of this research, the experimental setup includes detailed specifications of the hardware and software used, as well as precise metrics for evaluating model accuracy and memory use efficiency. This evaluation metrics are based on the standards mentioned in papers [1, 20], enabling the reproduction of this results and verification of findings.

Comparative analysis of this model against alternative models has demonstrated significant advantages in memory usage efficiency while maintaining a high level of accuracy. Specifically, compared to models described in papers [6, 21], this model exhibits superior performance in processing large NLP datasets. Based on these findings, further researches are suggested in memory resource optimization and model evaluation as crucial for enhancing scalability and efficiency in NLP applications.

This choice of a random sampling approach with a smaller number of validation set instances is grounded in previous research showcasing its efficiency in memory optimization. Inspiration is found in works demonstrating how selective validation can significantly reduce resource demands without compromising model performance. This

approach's distinction lies in its adaptability to various dataset sizes and models, paving the way for broader application in the NLP field [16].

The research results highlight the innovative nature of the approach, demonstrating the effective use of machine learning in predicting and optimizing NLP model memory requirements. These findings not only confirm the efficiency of this approach but also reveal its potential for wider applicability across different NLP applications [10, 22].

The results clearly show that the random sampling approach reduces memory needs while maintaining a high level of model accuracy. This is crucial for real-world applications where resources are often limited, and the need for efficient data processing is imperative [17].

This model advantage lies in its ability to efficiently manage memory resources while retaining high prediction accuracy. However, like any approach, it has limitations, especially in the context of extremely large datasets where selecting an appropriate number of samples becomes more challenging. Proper sample selection requires a precise balance between memory usage efficiency and maintaining model accuracy. These limitations highlight the need for developing more sophisticated sample selection algorithms that could adjust the sampling strategy to the dataset's specificities [4, 23].

Based on these findings, further research is suggested towards improving sample selection algorithms to enable even better adaptability of models to various sizes and types of datasets [24].

This detailed overview of research results provides insight into key findings and contributions to memory optimization for NLP models, affirming the significance of efficient memory optimization during the processing of textual data in NLP models. Achieving greater model accuracy, effective reduction in memory consumption is crucial for model scalability and working with large datasets [4, 25].

However, a critical part of memory optimization involves implementing a random sampling approach with a smaller number of instances from the validation set. This approach is realized within the create objective function. After tokenizing the dataset, 100 indices from the validation set were randomly sampled. This reduction in sample size significantly decreased memory needs during model evaluation, while preserving sample representativeness and ensuring relevant validation results [5].

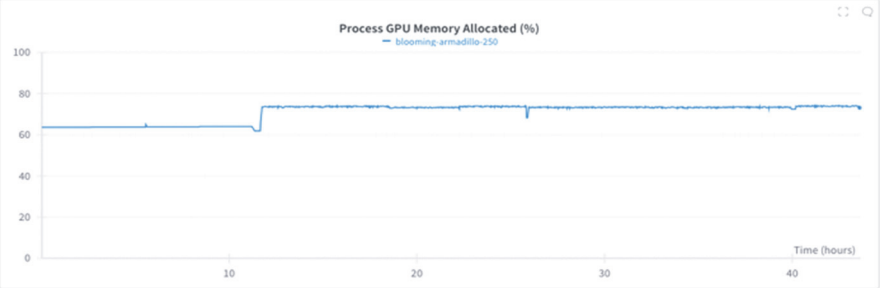

Figure 6 GPU memory allocation chart

Fig. 6 illustrates the GPU memory allocation during the validation process, showing a savings of 2.4 GB out of a total 8 GB, representing a 30% reduction in memory consumption compared to traditional methods.

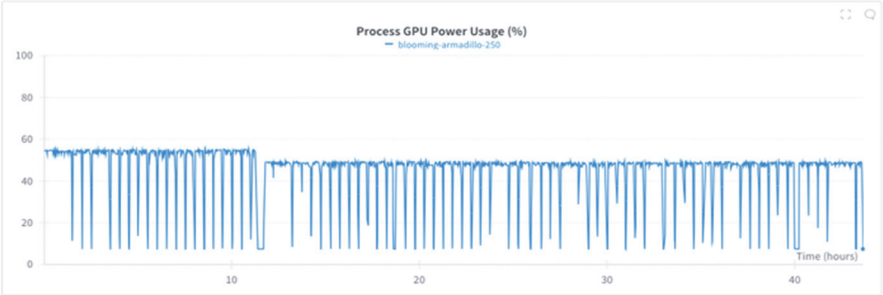

Figure 7 GPU power usage chart

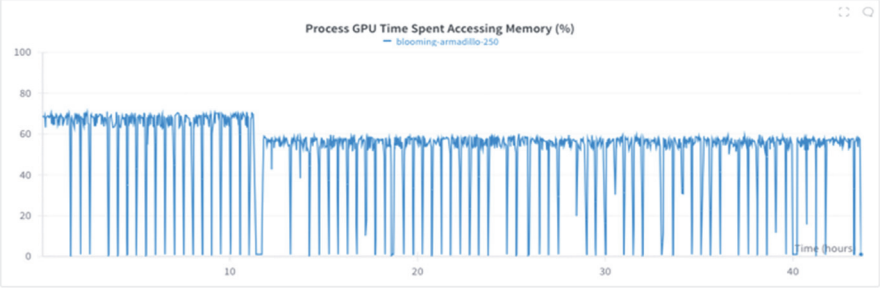

Figure 8 GPU time spent accessing memory chart

Fig. 7 demonstrates how GPU power is utilized during model training and evaluation, resulting in more than a 40% improvement in energy efficiency, further highlighting the efficiency of this approach.

Fig. 8 shows the percentage of time the GPU spends accessing memory, with a significant reduction in memory access time by 30% - 40%.

These charts not only demonstrate the efficiency of this method in reducing memory load but also confirm the theory that effective sequence length management and random sampling from the validation set contribute to better scalability and model performance.

$$Opti\ Max\ Len = \arg\min_{max\_len} \left(Loss + \lambda \times Mem \right) \quad (3)$$

Eq. (3) is used for optimizing tokenization parameters, balancing model loss and memory usage. Eq. (3) can be interpreted as follows: the optimal value of max_length (*Opti Max Len*) is sought that provides the best balance between model performance expressed through Model Loss (*Loss*) and efficiency Memory Usage (*Mem*).

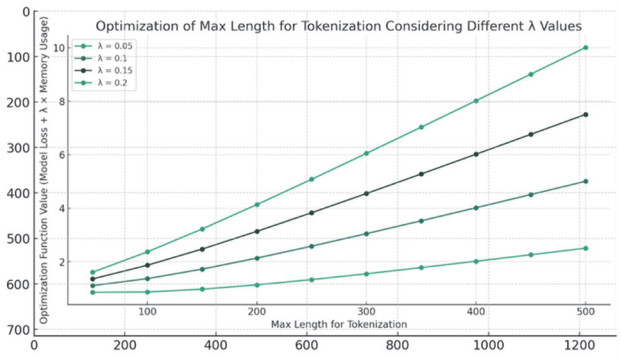

Figure 9 Optimization of max length for tokenization considering model loss and memory usage

Table 3 Estimated memory requirements by batch size and average token size

Max Length for Tokenization	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.15$	$\lambda = 0.2$
2000	100	200	300	400
4000	200	400	600	800
6000	300	600	900	1200
8000	400	800	1200	1600
10000	500	1000	1500	2000

Tab. 3 estimates memory needs by batch size and average token size. Each λ value corresponds to a weighting factor in the optimization function that encompasses both model loss and memory usage, showing a crucial balance between computational efficiency and model performance. These estimates, derived from trends shown in Fig. 9, clarify how the optimization function value increases with the maximum token length for each different λ value. These insights are key for optimizing NLP models, as they lead to fine-tuning of tokenization parameters to minimize memory footprint without negatively impacting model loss [5].

The implementation of the random sampling approach has shown a significant reduction in memory load during the validation phase. This approach allows for the analysis of the model on a smaller yet representative part of the validation set, achieving a balance between memory requirements and evaluation quality. In the experimental environment, this approach has shown promising results by significantly reducing the necessary memory resources during model validation [5].

Pseudocode 2 presented on the last page of this paper serves as a basis for model training and evaluation, using a smaller but representative subset of the data set for validation. This reduces memory requirements during evaluation while preserving the relevance of the evaluation results.

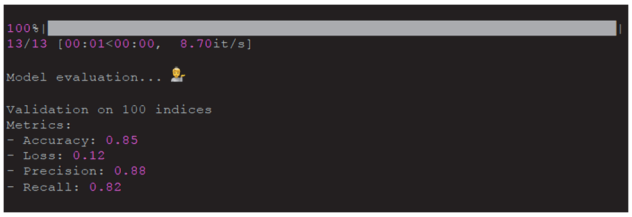

Figure 10 Console output showing validation and the success of randomly selected validation samples on 100 indices

Figure 11 Loading the model, tokenizer, adding specific tokens to the tokenizer, setting questions, and generating responses using the GPT-2 model

Fig. 10 and Fig. 11 document the success of testing the model and tokenizer, as well as generating responses using the GPT-2 model, thereby confirming the feasibility of an evaluation consisting of 100 randomly selected samples from the validation set [21].

This detailed review of the research results provides insight into key findings and contributions to memory optimization for NLP models, affirming the importance of efficient memory optimization during the processing of textual data in NLP models. Achieving greater model accuracy, effective reduction in memory consumption is crucial for model scalability and working with large datasets.

5 CONCLUSION

This research delves deeply into the core issue of memory optimization in Graphics Processing Units (GPU) within the field of deep learning, with a special emphasis on applications in Natural Language Processing (NLP) [1, 2]. This paper revealed that a strategy combining advanced machine learning algorithms [23] with innovative tokenization and data manipulation techniques [16] is essential for overcoming GPU memory limitations. This approach not only addresses challenges such as

"CUDA out of memory" errors but also enhances the training and evaluation processes of deep learning models to a higher level.

This paper has shown that integrating dynamic predictive models facilitates real-time analysis and adjustment of memory consumption, significantly reducing resource needs without compromising model performance [5, 26]. This technique, grounded in deep learning and machine learning, enables models to learn from their operations, optimizing memory usage as required, leading to revolutionary improvements in the scalability and efficiency of NLP applications.

Also, empirical results from this study highlight the significant impact of proper data manipulation and optimization during tokenization, not just on reducing memory consumption but also on enhancing the overall accuracy and performance of models [15]. This contribution is particularly crucial in the context of working with large datasets, where memory efficiency becomes key to maintaining high model performance.

Through this paper, the foundations have been laid for future research in the field of memory optimization in deep learning and NLP, simultaneously paving the way for the development of new, more efficient technologies and methods that will enable better resource utilization and improve the performance of deep learning models [18]. A challenging journey lies ahead in discovering and implementing even more advanced solutions, but the results that have been achieved provide a good basis for an optimistic view of the future of NLP technologies.

6 REFERENCES

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. *MIT Press*, ISBN 978-0262035613.

[2] Goyal, P., Pandey, S., & Jain, K. (2018). Deep Learning for Natural Language Processing: Creating Neural Networks with Python. *Apress*. https://doi.org/10.1007/978-1-4842-3685-7

[3] Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., & Zhang, M. (2023). Towards General Text Embeddings with Multi-stage Contrastive Learning. *arXiv Labs*.

[4] Jin, C., Shi, Z., Li, W., & Guo, Y. (2021). Bidirectional LSTM-CRF Attention-based Model for Chinese Word Segmentation. *arXiv Labs*.

[5] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with Paged Attention. *arXiv Labs*. https://doi.org/10.1145/3600006.3613165

[6] Vaswani, A., et al. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, *30*, (NIPS 2017).

[7] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Association for Computational Linguistics*, 4171-4186. https://doi.org/10.18653/v1/N19-1423

[8] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2019). Hugging Face's Transformers: State-of-the-art Natural Language Processing. *arXiv Labs*. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[9] Sawicki, J., Ganzha, M., & Paprzycki, M. (2023). The State of the Art of Natural Language Processing, A Systematic Automated Review of NLP Literature Using NLP Techniques. *Data Intelligence, 5*(3), 707-749. https://doi.org/10.1162/dint_a_00213

[10] Deguchi, H., Utiyama, M., Tamura, A., Ninomiya, T., & Sumita, E. (2020). Bilingual Subword Segmentation for Neural Machine Translation. *International Committee on Computational Linguistics (COLING 2020)*, 4287-4297. https://doi.org/10.18653/v1/2020.coling-main.378

[11] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv Labs*.

[12] Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C-M., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H. T., Chen, J., Liu, Y., Tang, J., Li, J., & Sun, M. (2023). Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence, 5*(3), 220-235. https://doi.org/10.1038/s42256-023-00626-4

[13] Liu, Y., Ji, L., Huang, R., Ming, T., Gao, C., & Zhang, J. (2018). An Attention-Gated Convolutional Neural Network for Sentence Classification. *arXiv Labs*.

[14] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *Association for Computational Linguistics*, 2978-2988. https://doi.org/10.18653/v1/P19-1285

[15] Sun, S. & Iyyer, M. (2021). Revisiting Simple Neural Probabilistic Language Models. *Association for Computational Linguistics*, 5181-5188. https://doi.org/10.18653/v1/2021.naacl-main.407

[16] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532-1543. https://doi.org/10.3115/v1/D14-1162

[17] Feng, Y., Hu, C., Kamigaito, H., Takamura, H., & Okumura, M. (2021). Improving Character-Aware Neural Language Model by Warming up Character Encoder under Skip-gram Architecture. *International Conference on Recent Advances in Natural Language Processing (RANLP 2021) - INCOMA Ltd.*, 421-427. https://doi.org/10.26615/978-954-452-072-4_048

[18] Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators (*ICLR 2020)*.

[19] Thrush, T., Tirumala, K., Gupta, A., Bartolo, M., Rodriguez, P., Kane, T., Gaviria Rojas, W., Mattson, P., Williams, A., & Kiela, D. (2022). Dynatask: A Framework for Creating Dynamic AI Benchmark Tasks. *Association for Computational Linguistics*, 174-181. https://doi.org/10.18653/v1/2022.acl-demo.17

[20] Howard, J. & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *Association for Computational Linguistics*, 328-339. https://doi.org/10.18653/v1/P18-1031

[21] Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. *arXiv Labs*.

[22] Farrell, M., Recanatesi, S., Moore, T., Lajoie, G., & Shea-Brown, E. (2022). Gradient-based learning drives robust representations in recurrent neural networks by balancing compression and expansion. *Nature Machine Intelligence, 4*, 564-573. https://doi.org/10.1038/s42256-022-00498-0

[23] Kelvinius, F. E., Georgiev, D., Toshev, A. P., & Gasteiger, J. (2023). Accelerating Molecular Graph Neural Networks via Knowledge Distillation. *arXiv Labs*.

[24] Aydin, I., Sevi, M., Akin, E., Güçlü, E., Karaköse, M., & Aldarwich, H. (2023). A Deep Learning-Based Hybrid Approach to Detect Fastener Defects in Real-Time. *Tehnički vjesnik-TechnicalGazette, 30*(5), 1461-1468. https://doi.org/10.17559/TV-20221020152721

[25] Raffel, C., et al. (2020). T5: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv*.

[26] Gopalun, K. & John Samuvel, D. (2023). Deep Learning Technique for Power Domain Non-Orthogonal Multiple Access Using Optimised LSTM in Cooperative Networks. *Tehničkivjesnik-TechnicalGazette, 30*(5), 1397-1403. https://doi.org/10.17559/TV-20221228104420

Contact information:

Dejan DODIĆ, Assistant Professor
(Corresponding author)
The Academy of Applied Technical and Preschool Studies,
Department of Information - communication technologies,
Beogradska 18, Niš, Serbia
E-mail: dodic017@gmail.co

Dušan REGODIĆ, PhD Professor
MB University, Faculty of Business and Law,
Department of Advanced information technologies,
Teodora Drajzera 27, Belgrade, Serbia

APPENDIX

Pseudocode

Pseudocode 1: Optimization of Memory Consumption Through Random Sampling of Data in Model Evaluation

```
max_length = 160
# Tokenization Function
Function tokenize_function(batch):
   - For each text in batch["text"]:
      - If text is empty or only whitespace, discard it
   - For the cleaned batch of texts:
      - If there are no valid texts, return {"input_ids": [],
"attention_mask": []}
      - Else, tokenize texts using Hugging Face tokenizer with
parameters:
         - truncation: True
         - padding: "longest"
         - return_special_tokens_mask: True
         - max_length: max_length
         - return_tensors: 'pt'
      - Handle tokenization errors and return empty values if any
# Objective Function for Model Optimization
Function create_objective (model):
   - Use Optuna for hyperparameter tuning:
      - Function objective (trial, model):
         - Set max_length using trial.suggest_int ('max_length', 64, 256,
step=16)
         - Define encode_function (examples):
            - Use tokenize_function for text processing
            - If "input_ids" or "attention_mask" are empty, return empty
values
         - Map encode_function on dataset, batched=True,
remove_columns=["text"]
         - Set dataset format for PyTorch
         - Randomly sample 100 indices from validation set
         - Initialize MyTrainer with specified parameters:
            - gamma, model, transformers. Training Arguments, etc.
            - train_dataset: tokenized_dataset ["train"]
            - eval_dataset: tokenized_dataset ["validation"].select
(random_indices)
         - Train the model and return eval_output ["eval_loss"]
```

Pseudocode 2: Efficient Model Evaluation with Random Sampling from Validation Set

```
# random sampling based on 100 indices from the validation set
num_samples = 100
# all indices from the validation set
all_indices = list (range (len(tokenized_dataset["validation"])))
# random indices selection
random_indices = random.sample (all_indices, num_samples)

trainer = MyTrainer(
   args=transformers.TrainingArguments(
      # other arguments
   ),
   train_dataset=tokenized_dataset["train"],
   eval_dataset=tokenized_dataset["validation"].select(random_indices),
)
```

Clarification for Pseudocode 2:

This part of the code relates to the training and evaluation process of the model:
num_samples = 100: Setting the number of samples to be randomly selected from the validation set. In this case, 100 samples were chosen considering that the training was conducted on a GPU with 8GB VRAM.
all_indices = list (range(len(tokenized_dataset["validation"]))): Generating a list of all indexes in the validation set. This list indexes the data within the validation set.
random_indices = random.sample (all_indices, num_samples): Here, a random sampling of 100 indices from the list of all indices in the validation set is performed. The random.sample function enables the selection of random samples without replacement.
trainer = My Trainer (...): Initialization of the trainer responsible for training and evaluating the model.
The trainer's parameters include:
args=transformers. Training Arguments(...): The other arguments used in training.
train_dataset=tokenized_dataset["train"]: The dataset used for training the model.
eval_dataset=tokenized_dataset["validation"].select(random_indices): The evaluation dataset consisting of 100 randomly selected samples from the validation set.