DOI: https://doi.org/10.26565/2304-6201-2023-59-02

УДК (UDC) 004.4`6:004.4`4

Deineha Oleksandr *PhD student*
V. N. Karazin Kharkiv National University, 4 Svobody Sq., Kharkiv, 61022, Ukraine;
e-mail: oleksandr.deineha@karazin.ua
https://orcid.org/0000-0001-8024-8812

The Clustering of Lambda Terms by Using Embeddings

Relevance. The importance of optimizing compilers and interpreters for functional programming languages, mainly through the lens of Lambda Calculus, is paramount in addressing the increasing complexity and performance requirements in software engineering. The emphasis of this study lies in this critical area, aiming to leverage advanced machine learning techniques to enhance identification and application of code reduction strategy.

Goal. The primary goal is to improve the performance and efficiency of compilers and interpreters by deepening the understanding of program code reduction strategies within Lambda Calculus. The research is aimed at using machine learning to convert lambda terms into feature vectors, facilitating the exploration of optimal reduction strategies.

Research methods. The study employs a comprehensive approach, generating a wide range of lambda terms for analysis. It utilizes OpenAI's text embedding model to transform these terms into embedding vectors, employing clustering analyses (DBSCAN with Euclidean measurements) and visualizations (PCA and t-SNE) to identify patterns and assess feature separability. The research navigates the complexities of choosing between specific and universal reduction strategies.

The results. Findings have revealed clear distinctions among lambda term representations within the embedding vectors, supporting the hypothesis that cluster analysis can uncover identifiable patterns. However, the challenges have been encountered due to OpenAI Embeddings' training being generally focused on human-readable text and code, and that complicates the precise representation of Lambda Calculus terms.

Conclusions. This exploration underscores the challenges in pinpointing the optimal reduction strategy for Lambda Calculus terms, highlighting the limitations of current mathematical models and the need for tailored machine learning applications. Despite the hurdles with the OpenAI Embeddings model's adaptability, the research offers significant insight into the potential of machine learning to refine the optimization processes of compilers and interpreters in functional programming environments.

Keywords: *Pure Lambda Calculus, Clustering Analysis, Pretrained Embedding, Hidden Space.*

Як цитувати: Дейнега О. А. Кластеризація Лямбда Термів з використанням Вбудувань. *Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»*. 2023. вип. 59. С.16-23.
https://doi.org/10.26565/2304-6201-2023-59-02

How to quote: Deineha O., "The Clustering of Lambda Terms by Using Embeddings" *Bulletin of V.N. Karazin Kharkiv National University, series "Mathematical modeling. Information technology. Automated control systems*, vol. 59, pp.16-23, 2023.
https://doi.org/10.26565/2304-6201-2023-59-02

1. Introduction

In the modern world of software engineering, functional programming languages are pivotal, providing sophisticated solutions to intricate challenges [1]. With escalating demands for enhanced performance, the importance of optimizing compilers cannot be overstated. Our study focuses on Lambda Calculus, an essential construct of functional programming languages, to achieve this objective. The aim is to analyze software code to identify the most effective reduction strategy, therefore improving the efficiency of both compilers and interpreters [2].

Lambda calculus is the main framework for our study, enabling the simulation of interpreters and compilers in their task to find the best reduction strategies. By generating a broad spectrum of lambda terms, we establish a solid testing environment to evaluate various methods to enhance normalization quality [2, 3]. The intricate decision-making process, whether to develop the best strategies for each

term or to adopt a universal strategy like "Rightmost Innermost," demonstrates the nuanced comprehension of its complexity.

The primary objective of this research is to enhance the functionality of interpreters and compilers used in functional programming languages, specifically through a detailed examination of lambda calculus. The focus is to increase our knowledge of reduction strategies and to improve the operational efficiency of compilers and interpreters. A novel aspect of our scientific inquiry involves applying advanced machine learning techniques to represent lambda terms as vectors of features; that includes analyzing such vectors for their ability to be separated and comparing these separation methods against a strategy prioritized for its efficiency.

Moreover, we encounter a computational dilemma: if it is better to continuously execute Lambda term reduction using various strategies in parallel, selecting the most suitable one based on a specific criterion, or transform a Lambda term into a more straightforward representation and input it into artificial neural networks (ANNs), thereby determining the optimal strategy.

2. Literature Review and Problem Statement

The use of sophisticated machine learning methods to enhance the efficiency of programming language compilers and interpreters is a concept that has been explored previously [4, 5]. CompilerGym [6] provides a platform for broader compiler research, offering an environment for experimentation without delving into specific optimizations. Most studies on optimizing compilers and interpreters focus on the most widely used object-oriented programming languages [7, 8, 9]. The clustering to identify similarities between functions have been utilized in [7]. The transforming program data by using PCA for the LLVM compiler and implementing optimizations based on expert knowledge has been examined in [8]. The iterative compilation method, testing a limited set of optimization possibilities and demonstrating its effectiveness for optimizing code fragments has been adopted in [9]. Furthermore, the application of reinforcement learning to compiler optimization, employing neural optimization agents to support manually crafted optimization sequences has been explored in [10].

Optimizations for compilers of functional programming languages have been comparatively less investigated. The heap profiling for a functional compiler using hand-crafted logic has been explored in [11]. Similarly, custom optimizations have been applied in [12], focusing on a functional web application. Furthermore, model functional languages have typically concentrated on specific reduction strategies, such as Haskell's call-by-need and call-by-value with unique mechanisms and OCaml's call-by-value.

Given this context, the challenge of optimizing compilers and interpreters for functional programming emerges as a significant area of interest. While machine learning techniques are applicable in optimizing compilers for object-oriented languages, this work seeks to identify features within functional code that could indicate the effectiveness of optimization strategies, using machine learning as a tool for exploration.

In the previous work, we have used a similar approach to clustering analysis of terms by using a large language model, Code BERT [13]. This research has shown us some possible ways of creating meaningful vector representations of vector terms.

3. Research Goals and Objectives

To research further the inner structure of generated lambda terms space, the main aim of this study is to enhance the performance of interpreters and compilers for functional programming languages, with a special emphasis on Lambda Calculus. The primary objective is to deepen our comprehension of program code reduction strategies and to improve the efficiency of both compilers and interpreters. The scientific novelty of this research is utilizing of sophisticated machine learning methods to encode lambda calculus terms into feature vectors, followed by an analysis of these vectors for separability and a comparison of their separation to the prioritization of term strategies.

In line with the main goal, the specific objectives of this research are outlined as follows:

- To conduct a clustering analysis of the lambda terms dataset to identify patterns or potential for data division.
- To evaluate the effectiveness of OpenAI's text embedding model in extracting features from lambda terms.
- To investigate the relationships between the extracted features and the identified clusters.

4. Materials and Methods of Research

The objective of this study is to refine the functionality of existing interpreters and compilers for functional programming languages. Lambda Calculus has previously been recognized as a basic model of functional programming languages [2, 3]. It facilitates the simulation of interpreters or compilers to select the most effective reduction strategies. Moreover, it provides a way for the synthetic generation of a wide array of lambda terms, enabling accurate evaluation of the strategies. Selecting an optimal reduction strategy involves devising a unique strategy for each term or applying a specific strategy, such as Rightmost Innermost, for the whole term reduction process. Both methodologies have been explored. The former strategy allows for the creation of a greedy algorithm that selects the best redex for reduction at any given moment. In this context, we have evaluated the disparity in redexes, indicating the varying resources needed for their reduction, with computational complexity as the measure of this disparity, gauged by the time taken per reduction step [3]. Additionally, our research aimed to predict the number of lambda term reduction steps by using a particular strategy, employing a simplified representation of terms that keeps only their tree structure, and excluding variable information [14]. For this purpose, standard Artificial Neural Network (ANN) models commonly used in Natural Language Processing have been applied to forecast the number of steps required for specific strategies [14].

Through our experiments, we have observed that certain redexes suggest an inclination towards one or another standard reduction strategy. Nonetheless, the mere presence of these redexes does not guarantee that a term is best suited to be reduced by that strategy. It indicates a need for a deep analysis of the term to determine if a redex indeed suggests a priority for reduction. Previous studies utilized a simplified term representation [3, 14], which was found to lack a qualitative analysis of terms and omitted critical information. Therefore, we have decided to analyze terms while preserving their variable information, potentially enhancing the differentiation of terms according to their preferred reduction strategy. This approach suggests the possibility of identifying terms that require fewer reduction steps under a specific strategy without specifying the exact number of reductions. As one of the most sophisticated and modern approaches to transforming text representations into vectors, we have selected the text-embedding-ada-002 model of OpenAI embedding as one of the most stable and the one that has shown promising results during our research. During our work, we also experimented with other new OpenAI embedding models but focused on the one mentioned above.

4.1. Gathering Feature Representations

Our work focuses on OpenAI embeddings which are one of the most popular and easy-to-use. Such an approach also significantly reduces computational requirements, as on our side we use a simple API that allows us to input text representations of lambda terms and retrieve vectors as an output. All the calculations are done on the OpenAI servers.

What are text embeddings? Text embeddings, as developed by OpenAI, evaluate how closely text strings are related. Common uses for embeddings include:

- Search (ranking search results by how relevant they are to a search term);
- Clustering (grouping text strings based on how similar they are);
- Recommendations (suggesting items that have text closely related to each other);
- Anomaly detection (spotting text strings that do not fit in with the rest due to low relatedness);
- Measuring diversity (examining how varied the similarity among text strings is);
- Classification (assigning text strings to categories they closely align with); The distance between their vectors gauges the degree of similarity between texts; shorter distances suggest more significant similarity in content, while longer distances signify reduced similarity.

Embeddings can serve as versatile encoders for free-text features within machine learning models. By integrating embeddings, the efficiency of any machine learning model can be significantly enhanced, particularly when some of the input data includes free text. Additionally, embeddings can encode categorical features in a machine learning framework. This is especially useful when dealing with categorical variables that have meaningful and numerous names, like job titles, where similarity embeddings tend to outperform search embeddings for such applications.

4.2. Analysis of Embeddings

For our analysis, we have utilized a collection of 4,000 artificially generated lambda terms. These terms were produced using a method based on random recursion, ensuring a balanced distribution of

Variables, Applications, and Abstractions throughout the structure of each term to include a wide variety of term types. The specific method of generating these terms is described in [14]. These lambda terms are represented as text, making them suitable for direct input into our chosen embedding model.

Hence, we have employed the OpenAI Embeddings model to transform lambda terms, including variable data, into vectors that should represent their unique meaning. This model interprets the textual representations of lambda terms, translating them into vectors.

The outcome, referred to as embeddings, captures the essence of the input text [15]. Then the clustering algorithm processes these vectors. This approach is akin to the Word2Vec methodology [16], which involves manipulating word embeddings (vectors representing the significance of words in a multidimensional space) through operations like addition or subtraction, thus creating new embeddings that retain the semantic value of the original word embeddings involved in the operation [16, 17].

4.3. Clustering Algorithms

During our previous work and analysis of the distribution patterns in the embedding space within this particular research, we noticed that unsupervised learning techniques, especially clustering analysis, are quite effective in distinguishing among the data. We have identified potential groupings with a natural, logical structure by illustrating various strategy priorities on a graphical plot, indicating a viable path for automated segmentation. It is essential to underline that our research aimed not to classify strategies per se but to investigate how data is distributed and uncover significant trends using unsupervised methods. Common methods for carrying out clustering analysis include K-means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), the Gaussian Mixture Model (GMM), and Agglomerative Clustering. Our study specifically has delved into the DBSCAN technique.

DBSCAN clusters data points based on proximity, effectively distinguishing between densely populated and less dense regions. It defines clusters by identifying areas of high density separated by regions of low density [18].

The DBSCAN algorithm uses common metrics such as Euclidean, cosine, L1, and L2. This algorithm exhibits resilience against outliers and offers the flexibility of not needing a predetermined number of clusters to operate effectively. It excels in identifying clusters of diverse shapes and sizes, showcasing its versatility in cluster formation. However, the performance of DBSCAN is significantly influenced by the selection of hyperparameters, and it may face challenges in accurately clustering data with variable densities.

Taking into account our previous studies [13], in this research we have focused on one clustering method: DBSCAN, with the Euclidean metric as one of the most promising. Since the chosen clustering method depends on fine-tuning hyperparameters, selecting the proper ones is crucial. To determine the most suitable hyperparameter set, we have evaluated the following metrics to assess the quality of clustering:

- The Silhouette score [19] evaluates how well an object fits within its cluster compared to others. It ranges from -1 to +1, where a higher score signifies better matching within the cluster and poorer matching with adjacent clusters. Scores above 0.7 denote robust clustering, above 0.5 indicate reasonable clustering, and above 0.25 suggest weak clustering. However, silhouette scores may converge in high-dimensional clustering, making differentiation challenging. The Silhouette score excels in assessing cluster quality for convex-shaped clusters but may falter with irregularly shaped or variably sized data clusters. This score is adaptable to any metric.

- The Davies-Bouldin Index (DBI) [20] measures the average similarity of each cluster with its closest cluster, based on the ratio of distances within the cluster to distances between clusters. Clusters that are more separated and less scattered receive better scores. The best score is zero, and lower scores signify superior clustering quality.

- The Calinski-Harabasz Index (CH Index) [21] is an internal metric that judges clustering quality based solely on the dataset and clustering outcomes without reference to external truth labels. It calculates the ratio of between-cluster dispersion to within-cluster dispersion, offering insight into effectiveness of clustering.

- The Within-Cluster Sum of Squares (WCSS) [22] evaluates the cohesiveness of clusters, especially in K-Means clustering. It totals the squared distances between each point in a cluster and its center, providing a numeric value for the cluster compactness.

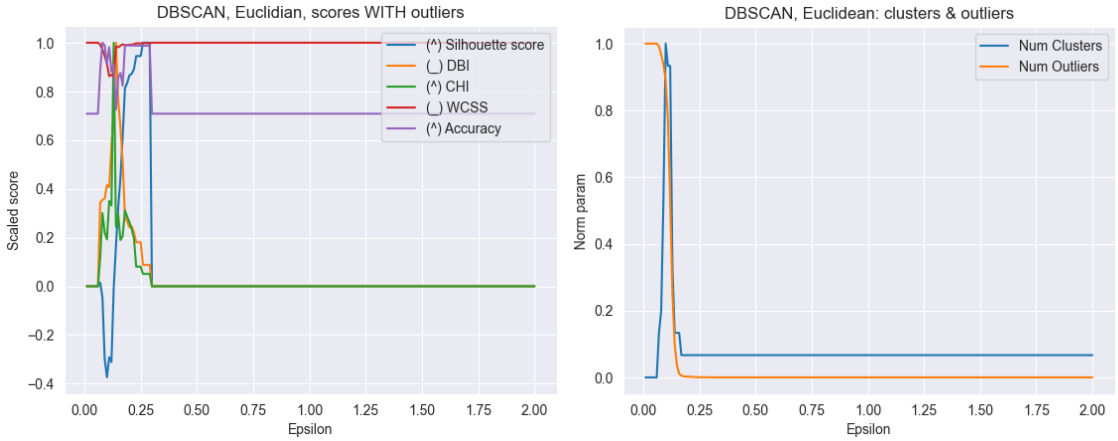

Fig. 1. Tuning the DBSCAN epsilon hyperparameter with various metrics (Silhouette, DPI, CHI, WCSS) for the Euclidean metric and estimating the number of outliers and clusters.

Collected mean embedding vectors can be visualized with Principal Component Analysis (PCA) [23] and t-distributed Stochastic Neighbor Embedding (t-SNE) [24]. The results of such data compression of the mean embeddings are shown in Fig. 2 for PCA and Fig. 3 for t-SNE.

5. Findings from Analyzing Lambda Terms

5.1. Collecting Embeddings

The mean embedding vectors, we have collected, can be graphically represented by using Principal Component Analysis (PCA) [23] and t-distributed Stochastic Neighbor Embedding (t-SNE). The visual compression of these mean embeddings through these techniques is presented in Fig. 2 for PCA and Fig. 3 for t-SNE. As can be seen, some clusters can be visually distinguished, particularly in the t-SNE plot. On the other hand, PCA does not show us such results; most clusters are visually interconnected and cannot be easily separated.

5.2. Analysis of Clustering

In our evaluation by using the chosen clustering techniques and quality assessment metrics, we initially sought to identify an optimal epsilon value for DBSCAN applied to the dataset of 4k embeddings (illustrated in Fig. 1). The selection of the epsilon value has proved difficult for this dataset, given the conflicting results from the clustering quality metrics and our aim to reduce the number of outliers. Notably, the Elbow method did not apply to determining WCSS for clustering. We aimed to achieve the highest possible Silhouette score and CH Index while aiming to lower the Davies-Bouldin Index (DBI). Consequently, for DBSCAN, we have opted for an epsilon value of 0.125.

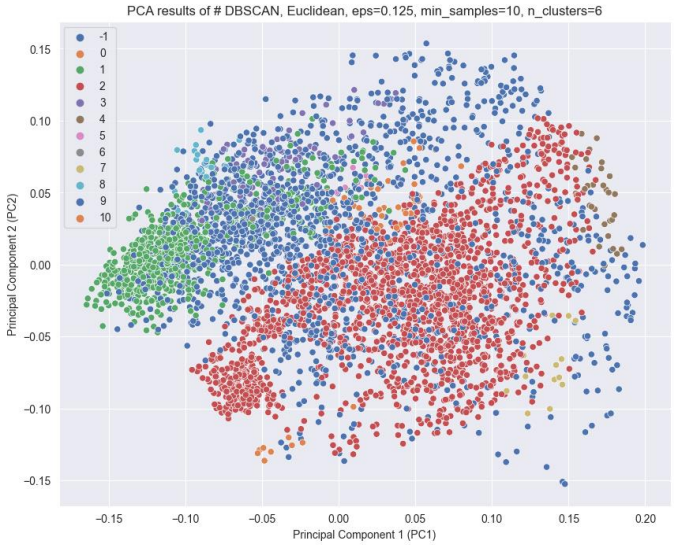

Fig. 2. Visualizing clustering results by using PCA dimension shrinking of embedding data with DBSCAN clustering.

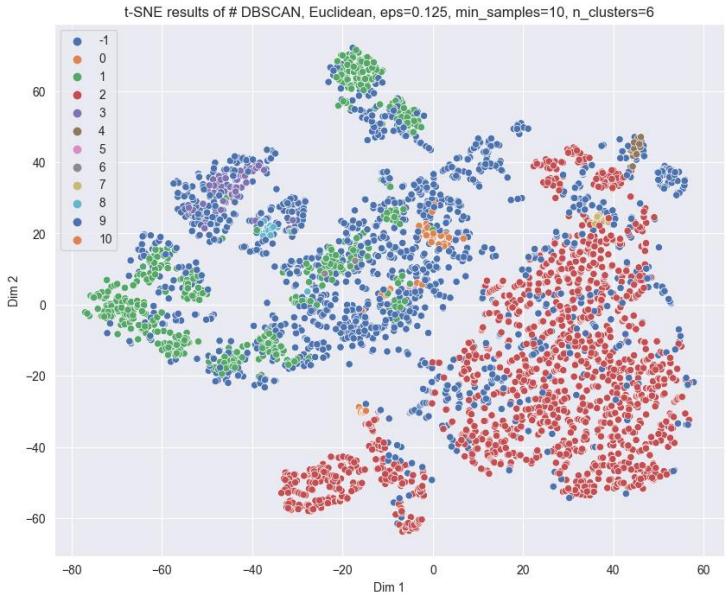

Fig. 3. Visualizing clustering results by using t-SNE *dimension shrinking of embedding data with DBSCAN clustering.*

6. Discussion of the results

In the discussion on the outcomes of our research, identifying the optimal strategy for term reduction emerges as a complex challenge, fundamentally unsolvable through mathematical means, as indicated by [2, 3, 14]. This complexity underscores the absence of a singular solution for optimal strategy selection across all conceivable terms or a universal reduction strategy. Nevertheless, developing viable methods remains feasible within certain constraints. Our approach, which involves generating lambda terms as a cost-effective means of data collection, may not fully encapsulate the breadth or critical characteristics of real-world terms. An additional challenge is the reliance on the OpenAI Embeddings model, originally trained on human text and code but not specifically on lambda terms or similar representations. That could potentially lead to inaccurate representations of lambda calculus terms in embedding matrices. This misalignment also complicates the translation of these embeddings into a universal format for subsequent analysis.

7. Summary of Findings

This research has transformed Lambda terms into embedding vectors with 1536 dimensions by applying the OpenAI Embeddings model, as explained in section 5.1. Analysis by using PCA and t-SNE to visualize these compressed mean vectors has revealed clear distinctions among Lambda term representations in these embeddings, confirming our initial hypothesis that patterns could be recognized through cluster analysis.

Then we have examined data cluster formation by using the DBSCAN technique with Euclidean measurements as could be seen in section 5.2. This exploration highlighted the capacity of the OpenAI Embeddings model to draw out significant attributes from Lambda terms. However, the broad training of OpenAI Embeddings has not be done on lambda term representations explicitly, but mostly for human-readable text and code, adding complexity to depicting Lambda calculus terms within embedding matrices precisely.

REFERENCES

1. Chitil, O. (2020). Functional Programming. *Clean C++20.* doi: https://doi.org/10.1007/978-1-4471-3166-3.
2. Deineha, O., Donets, V., & Zholtkevych, G. (2023). On Randomization of Reduction Strategies for Typeless Lambda Calculus. *Communications in Computer and Information Science* 1980, pp. 25–38.

3. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Estimating Lambda-Term Reduction Complexity with Regression Methods. *International Conference "Information Technology and Interactions"*.
4. Ashouri, A.H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S., & Cavazos, J. (2017). “MiCOMP: Mitigating the Compiler Phase-Ordering Problem Using Optimization Sub-Sequences and Machine Learning”. *ACM Trans. Archit. Code Optim., 14, 29:1-29:28.* doi: https://doi.org/10.1145/3124452.
5. Chen, J., Xu, N., Chen, P., & Zhang, H. (2021). Efficient Compiler Autotuning via Bayesian Optimization. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 1198-1209. doi: https://doi.org/10.1109/ICSE43902.2021.00110.
6. Cummins, C., Wasti, B., Guo, J., Cui, B., Ansel, J., Gomez, S., Jain, S., Liu, J., Teytaud, O., Steiner, B., Tian, Y., & Leather, H. (2021). CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research. *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 92-105. doi: https://doi.org/10.1109/CGO53902.2022.9741258.
7. Martins, L.G., Nobre, R., Cardoso, J.M., Delbem, A.C., & Marques, E. (2016). Clustering-Based Selection for the Exploration of Compiler Optimization Sequences. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13, 1 - 28. doi: https://doi.org/10.1145/2883614.
8. Ashouri, A.H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S., & Cavazos, J. (2017). MiCOMP: Mitigating the Compiler Phase-Ordering Problem Using Optimization Sub-Sequences and Machine Learning. *ACM Trans. Archit. Code Optim.*, 14, 29:1-29:28. doi: https://doi.org/10.1145/3124452.
9. Xavier, T.C., & Silva, A.F. (2018). Exploration of Compiler Optimization Sequences Using a Hybrid Approach. *Comput. Informatics*, 37, 165-185. doi: https://doi.org/10.4149/cai_2018_1_165.
10. Mammadli, R., Jannesari, A., & Wolf, F.A. (2020). Static Neural Compiler Optimization via Deep Reinforcement Learning. 2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar), 1-11. doi: https://doi.org/10.1109/LLVMHPCHiPar51896.2020.00006.
11. Runciman, C., & Wakeling, D. (1992). Heap Profiling of a Lazy Functional Compiler. *Functional Programming*. doi: https://doi.org/10.1007/978-1-4471-3215-8_18.
12. Chlipala, A. (2015). An optimizing compiler for a purely functional web-application language. *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*. doi: https://doi.org/10.1145/2784731.2784741.
13. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Unsupervised Data Extraction from Transformer Representation. *EEJET, vol. 3, In press*.
14. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Deep Learning Models for Estimating Number of Lambda-Term Reduction Steps. *3rd International Workshop of IT-professionals on Artificial Intelligence “ProfIT AI 2023”.*
15. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: *A Pre-Trained Model for Programming and Natural Languages*. doi: https://doi.org/10.18653/v1/2020.findings-emnlp.139.
16. Dwivedi, V.P., & Shrivastava, M. (2017). Beyond Word2Vec: *Embedding Words and Phrases in Same Vector Space*. ICON.
17. Hartigan, J.A., & Wong, M.A. (1979). *A k-means clustering algorithm*. doi: https://doi.org/10.2307/2346830.
18. Zhang, Y., Li, M., Wang, S., Dai, S., Luo, L., Zhu, E., Xu, H., Zhu, X., Yao, C., & Zhou, H. (2021). Gaussian Mixture Model Clustering with Incomplete Data. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 17, 1 - 14. doi: https://doi.org/10.1145/3408318.
19. Ros, F., Riad, R., & Guillaume, S. (2023). PDBI: A partitioning Davies-Bouldin index for clustering evaluation. Neurocomputing, 528, 178-199. doi: https://doi.org/10.1016/j.neucom.2023.01.043.
20. Lima, S.P., & Cruz, M.D. (2020). A genetic algorithm using Calinski-Harabasz index for automatic clustering problem. *Revista Brasileira de Computação Aplicada*. doi: https://doi.org/10.5335/rbca.v12i3.11117.

21. Li, X., Liang, W., Zhang, X., Qing, S., & Chang, P. (2020). *A cluster validity evaluation method for dynamically determining the near-optimal number of clusters*. Soft Computing, 24, 9227-9241. doi: https://doi.org/10.1007/s00500-019-04449-7.
22. Chung, Heewon, Hoon Ko, Wu Seong Kang, Kyung Won Kim, Hooseok Lee, Chul Park, Hyun-Ok Song, Tae-Young Choi, Jae Ho Seo and Jinseok Lee. "Prediction and Feature Importance Analysis for Severity of COVID-19 in South Korea Using Artificial Intelligence: Model Development and Validation." *Journal of Medical Internet Research 23 (2021): n. Pag*. doi: https://doi.org/10.2196/27060.
23. Oliveira, F.H., Machado, A.R., & Andrade, A.D. (2018). On the Use of t-Distributed Stochastic Neighbor Embedding for Data Visualization and Classification of Individuals with Parkinson's Disease. *Computational and Mathematical Methods in Medicine*, 2018. doi: https://doi.org/10.1155/2018/8019232.

Дейнега Олександр Андрійович
Аспірант
Харківський Національний Університет імені В.Н. Каразіна, майдан Свободи 4, 61022, Харків, Україна;
e-mail: oleksandr.deineha@karazin.ua
https://orcid.org/0000-0001-8024-8812

Кластеризація Лямбда Термів з використанням Вбудувань

Актуальність. Важливість оптимізації компіляторів та інтерпретаторів для функціональних мов програмування, зокрема через призму лямбда-числення, має першочергове значення для вирішення зростаючих вимог до складності та продуктивності в розробці програмного забезпечення. Це дослідження розміщується в цій критичній області, спрямоване на використання передових методів машинного навчання для покращення ідентифікації та застосування стратегії скорочення коду.

Мета. Основною метою є підвищення продуктивності та ефективності компіляторів та інтерпретаторів шляхом поглиблення розуміння стратегій скорочення програмного коду в лямбда-численні. Дослідження спрямоване на використання машинного навчання для перетворення лямбда-термінів у вектори ознак, полегшуючи дослідження оптимальних стратегій зменшення.

Методи дослідження. Дослідження використовує комплексний підхід, створюючи широкий спектр лямбда-термінів для аналізу. Він використовує модель вбудовування тексту OpenAI для перетворення цих термінів у вектори вбудовування, використовуючи кластеризаційний аналіз (DBSCAN з евклідовими вимірюваннями) та візуалізацію (PCA та t-SNE) для виявлення шаблонів і оцінки відокремлюваності функцій. Дослідження орієнтується через складність вибору між конкретними та універсальними стратегіями скорочення.

Результати. Отримані дані виявляють чіткі відмінності між представленнями лямбда-термів у векторах вбудовування, підтверджуючи гіпотезу про те, що кластерний аналіз може виявити шаблони, які можна ідентифікувати. Однак виникли труднощі через загальну спрямованість навчання OpenAI Embeddings на текст і код, які читаються людиною, що ускладнює точне представлення термінів лямбда-числення.

Висновки. Це дослідження підкреслює труднощі у визначенні оптимальної стратегії скорочення термінів лямбда-числення, підкреслюючи обмеження поточних математичних моделей і потребу в адаптованих програмах машинного навчання. Незважаючи на перешкоди з адаптивністю моделі OpenAI Embeddings, дослідження з'ясовує суттєве розуміння потенціалу машинного навчання для вдосконалення процесів оптимізації компіляторів та інтерпретаторів у середовищах функціонального програмування.

Ключові слова: *Лямбда-терми, Кластерний Аналіз, Претреновані Вбудування, Прихований Простір.*

Надійшла у першій редакції 11.07.2023, в останній - 03.09.2023.