The study's object is the process of extracting the characteristics of lambda terms, which indicate the optimality of the reduction strategy and increase the productivity of compilers and interpreters. The solution to the problem of extracting specific strategy priority data from lambda terms using Machine Learning methods was considered.

Such data was extracted using the large language model Microsoft CodeBERT, which was trained to solve the problem of summarizing the software code. The resulting matrices of embeddings were used to obtain vectors of average embeddings of size 768 and a latent space of size 8 thousand. Further, vectors of average embeddings were used for cluster analysis using the DBSCAN and Hierarchical Agglomerative clustering methods. The most informative variables affecting clustering were determined. Next, the clustering results were compared with the priorities of reduction strategies, which showed the impossibility of separating terms with RI priority. A feature of the obtained results is using machine learning methods to obtain knowledge.

The clustering results showed many of the same informative variables, which is explained by the similar shape of the obtained clusters. The results of comparing the clustering values with the real priority are explained by the impossibility of clearly determining the priority and the use of the Microsoft CodeBERT model, which was not trained for the analysis of lambda terms.

The proposed approach can find application in the development of compilers and interpreters of functional programming languages, allowing to analyze the code and extract important data to optimize the execution of programs. The obtained data can be used to develop rules aimed at improving the efficiency of compilation and interpretation

Keywords: functional programming, Lambda Calculus, Large Language Model, unsupervised learning methods

UDC 004.4`6:004.4`4

DOI: 10.15587/1729-4061.2024.298991

THE APPROACH DEVELOPMENT OF DATA EXTRACTION FROM LAMBDA TERMS

Oleksandr Deineha
Corresponding author
PhD Student*
E-mail: oleksandr.deineha@karazin.ua
Volodymyr Donets
PhD Student*
Grygoriy Zholtkevych
Doctor of Technical Sciences, Professor*
*Department of Theoretical and Applied Computer Science
V. N. Karazin Kharkiv National University
Svobody sq., 4, Kharkiv, Ukraine, 61022

Received date 26.02.2024
Accepted date 06.05.2024
Published date 28.06.2024

***How to Cite**: Deineha, O., Donets, V., Zholtkevych, G. (2024). The approach development of data extraction from lambda terms. Eastern-European Journal of Enterprise Technologies, 3 (2 (129)), 42–54. https://doi.org/10.15587/1729-4061.2024.298991*

1. Introduction

Functional programming languages form the basis of modern software development, offering elegant solutions to complex problems [1]. As performance requirements increase, compiler optimization becomes paramount. To achieve this goal, Lambda Calculus was considered a fundamental representation of functional programming languages. The point is to reveal the hidden connections of the program code to unravel reduction strategies, significantly increasing the performance of compilers and interpreters [2].

Lambda Calculus is the most straightforward functional programming language for studying the execution and interpretation of programs. Lambda Calculus allows for the simulation of the processes of interpreters and compilers in order to find optimal strategies for code execution and interpretation. The developed method of lambda term generation provides a reliable test ground for the proposed approaches in improving the quality of reduction [2, 3]. The complex process of choosing between building unique strategies for individual terms and choosing a global strategy such as the Rightmost Innermost approach reveals a subtle understanding of the complexity of the reduction. Machine learning methods to extract term data and find internal relationships between extracted data and normalization strategies will optimize the normalization process. The successful application of machine learning methods to extract these terms will allow to extend this approach to a broader range of functional programming languages. Such machine learning methods will optimize not only the reduction of lambda terms but also the interpretation and compilation of functional programming languages.

Therefore, the study of program execution efficiency is needed in practice due to the growing importance of optimizing compilers of functional programming languages.

2. Literature review and problem statement

The use of advanced machine learning techniques to optimize compilers and interpreters of programming languages is not new. Thus, work [4] considered the optimization of the GCC and LLVM compilers for the C programming language using Bayesian optimization. This work is an example of the application of machine learning methods in the field of optimizing C compilers. The advantage of the study is that, as a result of such optimization, it was possible to significantly improve the compilation speed compared to the standard approach of automatic tuning. However, the work does not consider other means of optimization and does not consider the optimization of interpreters and compilers for a functional programming language. This can be solved by researching the application of machine learning methods to optimize a functional programming language.

In turn, work [5] considered CompilerGym, which was proposed to use artificial intelligence methods to study compilers and interpreters. The strength of the work is how the proposed CompilerGym acts as a testing ground, providing a means to explore programming languages. However, the proposed platform does not allow for the optimization of programming languages, and CompilerGym's compatibility with functional programming languages is not specified.

Most of the works related to compiler and interpreter optimization are considered the most popular programming languages, and they are object-oriented [6–8]. Research [6] used cluster analysis to identify the similarity of functions. The study [7] considered the transformation of program data using the PCA method for the LLVM compiler and applied optimization using expert logic. Paper [8] examines the iterative compilation approach, exploring only part of the optimization space and demonstrating its effectiveness in optimizing code segments. The advantage of studies [6–8] is an overview of the problem of optimization of compilers and interpreters from different angles with the introduction of various machine learning methods. However, the works do not show the effectiveness of the proposed approaches for functional programming languages. This can be corrected with appropriate research. Also, the article [9] discusses the use of reinforcement learning to optimize compilers using neural optimization agents to replace manually created optimization sequences. This shows that it is possible to effectively apply the methods of uninformed learning to solve the problem of optimizing compilers and interpreters.

Fewer studies devoted to their optimization are considered for functional programming compilers. The research [10] considered heap profiling of a functional compiler with expert logic. In addition, in the study [11], expert optimization of the interpreter of the functional language of the web application was carried out. The advantage of studies [10, 11] is that such studies consider optimizing functional programming languages and show the possibility of successful implementation of such modifications. However, studies have not focused on a more detailed investigation of such optimizations. This is due to the lack of appropriate technologies for work [10] and for research [11] due to the introduction of excessive complexity. The solution to these shortcomings can be the application of modern machine learning methods for the analysis of functional programming languages.

The solution to the problem of the lack of a research playground for Lambda Calculus is considered in works [2, 3]. Here, the Pure Calculus Environment was proposed to explore Lambda Calculus, which represents functional programming languages. The papers consider a random strategy of reducing and adjusting this strategy by estimating the computational price of the corresponding steps. It is worth noting that the works considered either a random or a greedy algorithm for the reduction of lambda terms, and did not consider the root causes of the priority of some strategies over others. The solution to this problem can be applying machine learning methods to find such connections.

Also, in the study [12], the analysis of the influence of the structure of lambda term graphs on the number of reduction steps during their normalization according to the chosen strategy was considered. The advantage of this study is the application of various Artificial Neural Networks (ANNs) to analyze lambda term graphs through a simplified textual representation of terms. The disadvantage of this study is the lack of consideration of the influence of variables on the normalization process. The solution to this problem can be using more voluminous ANN models to perceive information about lambda-term variables.

Considering all the above, optimizing compilers and interpreters of functional programming languages is essential. In addition, machine learning methods have been successfully applied to optimize object-oriented compilers. It is shown that lambda calculus can serve to simplify the representation of functional programming languages. However, existing studies have not thoroughly analyzed the relationship between lambda terms and priorities in the reduction strategy. Therefore, a detailed study of the relationship between terms and reduction strategies is appropriate. Features obtained using machine learning as a term analysis tool can indicate such a relationship.

3. The aim and objectives of the study

The study aims to develop an approach to extracting lambda term data related to reduction strategies. This will make it possible to apply advanced machine learning techniques to represent lambda calculus terms as feature vectors. In turn, analyzing feature vectors for resolution and comparing resolution with the priority of the term reduction strategy will allow the development of approaches to optimizing compilers and interpreters.

To achieve the aim, the following objectives were set:

– analyze the applicability of the chosen method of machine learning as an instrument for extracting features from lambda terms;

– perform a cluster analysis of lambda-term data and check the presence of internal relationships or the possibility of data separation;

– check the dependencies between the obtained features and existing clusters;

– check the dependencies between the obtained clusters and reduction strategies.

4. The study materials and methods

4. 1. Object and research hypothesis

The object of research is the process of extracting the characteristics of lambda terms, which can indicate the priorities of reduction strategies. Such characteristics may combine specific redex types, leading to different reduction trajectories. The difficulty of extracting such characteristics lies in the different volumes of terms in which they can appear, which imposes additional data noise and the presence of certain redexes that can eliminate their appearance.

The work's central hypothesis is that applying machine learning methods will make it possible to transform lambda terms into some meaningful numerical representation. The methods of uninformed learning will make it possible to extract specific characteristics from the meaningful representation, which will indicate the priority of reduction strategies.

The research aims to improve understanding of the processes occurring in existing interpreters and compilers for functional programming languages. Therefore, according to previous works, Lambda Calculus is assumed to be a simple representation of functional programming languages [2, 3]. Lambda calculus allows the simulation interpreters or compilers to choose appropriate reduction strategies. In addition, it will allow the artificial generation of a vast number of lambda terms, which can be used to test the proposed strategies accurately. An appropriate reduction strategy can be selected by constructing a unique strategy for each independent term or choosing a specific reduction strategy (e.g., Rightmost Innermost) for the entire term reduction procedure. Both approaches were considered. The first allows for building a greedy strategy that chooses the optimal redex in the current state. For this, an analysis of the inequality of redexes (which means that

they require different resources to process their reduction) is performed. Computational complexity was chosen to measure their inequality, which can be measured by timing the reduction step [3]. The second was investigated by estimating a specific strategy's number of lambda-term reduction steps. In [12], a simplified representation of terms was used, which preserves only the tree structure with the loss of information about term variables. Typical ANN models for natural language processing tasks were used to estimate the number of reduction steps for specific strategies [12].

During experiments, it has been observed that some specific redexes indicate that terms have priority in one of the standard reduction strategies. However, this study suggests that such redexes are insufficient to indicate that a term should be prioritized in a particular strategy. In other words, it is necessary to analyze the terms in depth to decide whether a particular redex is sufficient to indicate the priority of the reduction. In previous studies, a simplified representation of terms was proposed [3, 12], which, as the practice has shown, is insufficient for qualitative analysis of terms and loses important discriminatory information. Therefore, it was assumed that preserving information about variable terms can improve their distinction in priority in a specific reduction strategy. This means it is possible to know that a term has fewer reduction steps for a given strategy without researching a specific number of reductions. Also, the research assumes that the best method for solving this problem is the Large Language Model (LLM), which is trained to analyze programming languages. Due to resource constraints, a pre-trained model was considered for solving problems related to programming languages.

4. 2. Collection and analysis of embeddings

The most advanced LLMs are built on the Transformer architecture [13, 14]. Due to the nature of ANNs, especially Transformers, it is possible to use average-level outputs as a feature vector. In other words, LLMs are suitable for converting text information into vectors or matrices of attribute values [13, 15]. Therefore, it is possible to convert the lambda terms into vectors or matrices of values that can be used for further investigation.

First, existing and publicly available LLMs were compared for solving different tasks related to programming languages. Information about existing models can be found using the HuggingFace service; the most trending models are listed in Table 1. The most advanced broad-based models are the Code Llama models [16], which include three possible model sizes for parameters 7B, 13B, and 34B. These models are trained in the most popular programming languages, such as Python, Java, JavaScript, and others, to solve various code-related tasks. However, the problem with this model is the computational requirements, so the models are not available for personal use. There is a similar problem with the code completion model Replit Code [17]. Other models, such as CodeTrans [18], CodeBERT [19], and CodeT5 [20], are suitable for use on personal computers, but fine-tuning them is challenging. In addition, the problem with these models is training them for tasks related mainly to object-oriented programming languages.

Using imprecisely adjusted models can cause inconvenient results. CodeTrans and CodeT5 are based on the architecture of the T5 model [18, 20], which uses the entire Transformer architecture (encoder and decoder). At the same time, CodeBERT is based on the architecture of the BERT model [19], which uses only the encoder part of the Transformer architecture. Microsoft CodeBERT is better suited for this purpose because more model weights are used for feature extraction, and CodeBERT is suitable for narrower tasks.

In order to discriminate terms by strategy priority, two standard strategies were chosen with different approaches to reducing terms used: Leftmost Innermost (LO) and Rightmost Innermost (RI). The LO strategy uses the first redexes for reduction. The RI strategy is the opposite and accepts most internal redexes (or right-hand side redexes) (Fig. 1).

Table 1

Comparison of LLMs for Programming Languages

No.	Model	Description	Tasks	Size
1	CodeTrans model	The model is based on the T5-small model. The model has its own SentencePiece dictionary model. Pre-training was used for seven unsupervised datasets in the field of software development. The model was then adapted to the task of program synthesis for Lisp-inspired DSL code [18]	Software synthesis; documentation generation; code compilation; comment generation	242 MB
2	Replit Code	A causal language model is trained on a subset of the Stack Dedup v1.2 dataset. The training mix includes 20 different languages, listed here in descending order of tokens: Markdown, Java, JavaScript, Python, TypeScript, PHP, and others [17]	Code completion	10.4 GB
3	Code Llama	Code Llama is a set of pre-trained and fine-tuned generative text models ranging from 7 to 34 billion parameters [16]	General synthesis and understanding of the code	~13 GB – ~70 GB
4	CodeT5	CodeT5 is a unified pre-trained Transformer model. The model was pre-trained in CodeSearchNet (Go, Java, JavaScript, PHP, Python, and Ruby). In addition, the authors collected two C/C# datasets from BigQuery1 to ensure that all subsequent tasks overlap programming languages with pre-training data [20]	Code generalization, generation, translation, refinement and defect detection	892 MB
5	CodeBERT	CodeBERT is a bimodal pretrained model for programming languages and natural language using a Transformer-based neural architecture and a hybrid objective function that includes a pre-trained task of detecting substituted tokens. The authors considered datasets containing Go, Java, JavaScript, PHP, Python, Ruby, and other code samples for training [19]	Generation of code documentation	499 MB

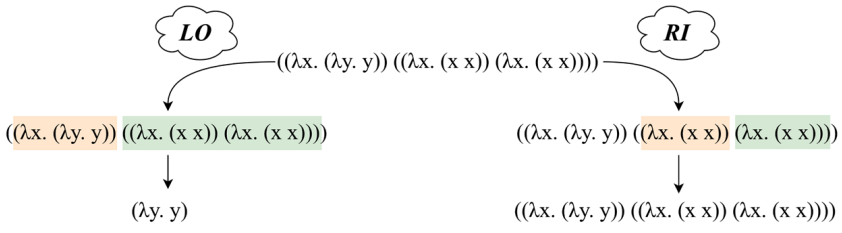


Fig. 1. Comparison of term reduction with LO and RI strategies

Thus, one can reduce the generated terms using LO and RI strategies and collect the number of reduction steps. The number of reduction steps can be used to distinguish terms by strategy priority. For this task, three possible classes of terms were considered:

1) LO priority terms (which have fewer reduction steps with the LO strategy);

2) RI priority terms (which have fewer reduction steps with the RI strategy);

3) terms of the same priority (with the same number of reduction steps with both the LO and RI strategies).

A data set of 4,000 artificially generated lambda terms was used for further analysis. The dataset was generated using a recursive randomization algorithm, evenly distributing the variables, applications, and abstractions in the term body to cover a diverse range of term variants. Details of the term generation algorithm can be found in the article [12]. The generated lambda terms can be represented as text that can be entered directly into the selected LLM model.

Therefore, a pre-trained CodeBERT model was considered for translating lambda terms with variable information in the value matrix. The selected model converts the textual information representing the lambda terms into token vectors with dimensions equal to the length of the terms. Token vectors can be fed into the CodeBERT model. Processing these token vectors results in matrices of size 768 by the number of tokens. These resulting matrices are called embeddings, encapsulating the content of the entered text [19]. It is difficult to compare the collected matrices and process them further, so the possibility of using average vectors of code embeddings of size 768 was considered. Further, it is possible to calculate such vectors through the Word2Vec approach [21], which allows word embeddings (vectors that represent the meaning of a word in some N-dimensional space). Applying addition or subtraction operations to embeddings can lead to the appearance of new embeddings, combining the meanings of the words involved in this operation [21, 22].

4. 3. Means of unsupervised data separation and assessment of such separation

When studying the distribution of data in average embedding space, it was observed that unsupervised learning approaches, in particular cluster analysis, are practical for delineating the data. By plotting different strategic priorities in the space of average embeddings, it became possible to recognize potential clusters with inherent logical coherence, offering a promising route for automatic segmentation. However, it is essential to note that the scope of this study explicitly excludes strategy classification, focusing instead on examining distributional characteristics and identifying meaningful patterns using unsupervised methods. Standard approaches for cluster analysis are the following methods:

– K-means is a partitioning method that divides a data set into 'k' clusters by minimizing the sum of squared distances between data points and the corresponding centroids of the clusters [23, 24]. The method assigns each data point to the cluster whose centroid is closest. Standard metric: Euclidean. Advantages: computational efficiency, suitable for large data sets, works well with spherical clusters. Disadvantages: sensitivity to the initial choice of the cluster center, struggles with non-linear or irregularly shaped clusters, requires a predetermined number of clusters.

– DBSCAN (Density-Based Spatial Clustering of Applications with Noise) combines data points that are close to each other and separates dense areas from sparse ones. The method defines clusters as regions with a high density of data points, separated by regions with a lower density [25, 26]. Standard metric: Euclidean, cosine, L1, L2. Advantages: resistant to outliers, not required in the specification of the number of clusters, can work with clusters of complex shapes and different sizes. Disadvantages: sensitive to the choice of hyperparameters, may have problems with clusters of different densities.

– The Gaussian Mixture Model (GMM) assumes that the data is a mixture of several Gaussian distributions. The method models each cluster as a Gaussian distribution and estimates the parameters (mean, covariance, and weight) to maximize the likelihood of the observed data [27]. Standard metric: Euclidean. Advantages: flexibility in handling different cluster shapes, provides probabilistic cluster assignment, and can estimate the density of data points. Disadvantages: computational cost, susceptibility to convergence problems.

– Hierarchical Agglomerative Clustering (HAC) starts with each data point as a single cluster, and iteratively merges the closest pairs of clusters until only one cluster remains. The process creates a binary tree or dendrogram, and the user can slice it at the desired level to obtain clusters [26]. Standard metrics: Euclidean, cosine, L1, L2. Advantages: it can work with clusters of complex shapes and different sizes, and results can be visualized using dendrograms. Disadvantages: sensitive to choice of binding method and distance metric, may scale poorly for large data.

Due to the visual analysis of possible forms of clusters (Fig. 3), the DBSCAN and HAC methods were considered for further cluster analysis of the dataset. These methods require hyperparameter tuning, so an appropriate set of hyperparameters needs to be selected and to evaluate this appropriate set of hyperparameters, the following metrics were considered to evaluate clustering quality:

1. Silhouette Score [27] measures how similar an object is to its cluster compared to others. The Silhouette Score ranges from −1 to +1, where a high value indicates that the object matches its cluster well and matches its neighbors poorly. A Silhouette Score greater than 0.7 indicates accurate clustering, a value greater than 0.5 is satisfactory, and a value smaller than 0.25 is weak. The problem lies in clustering high-dimensional data, where silhouette values become similar. Silhouette Score measures cluster quality when the clusters have a convex shape and may not perform well when the data clusters have irregular shapes or different sizes. Silhouette Score is suitable for any metric.

2. Davies-Bouldin Index (DBI) [28] is an average indicator of the similarity of each cluster with its most similar cluster, where similarity is the ratio of distances within a cluster to distances between clusters. Thus, more distant and less scattered

clusters will give a better result. The minimum score is zero, and lower values indicate better clustering.

3. CH Index (Calinski-Harabasz Index) [29] is an internal evaluation metric in which clustering quality is based solely on the data set and clustering results, not on external truth labels. The score is the ratio of the sum of intercluster variance and intracluster variance.

4. Within-Cluster Sum of Squares (WCSS) [30] is a metric used to assess the compactness or homogeneity of clusters in a clustering algorithm, particularly in the context of K-Means clustering. The metric measures the sum of squared distances between each data point in a cluster and the centroid of that cluster. WCSS provides a quantitative estimate of how closely clustered data points are.

4. 4. Overlap coefficient and strategies differentiation

The overlap ratio between the obtained clusters and the priority of the target strategy was also considered for further analysis of the clustering results. The idea behind the overlap ratio is to determine the precision for each cluster: if most elements of a cluster are labeled as a specific class, the entire cluster should be labeled as that class. It is now possible to calculate precision for all elements using the initial class labeling and cluster labeling, even if the number of clusters differs from the number of classes. This can help evaluate how well the clusters estimate the target classes. However, this indicator cannot be sufficient to evaluate the quality of clustering due to the nature of the strategy priority idea.

4. 5. Sensitivity analysis and feature distribution

For further research, sensitivity analysis was considered, which can be used to highlight the most informative parameters in cluster analysis. Due to their specificity, the methods of cluster analysis cannot provide information about the quality of clustering, so the possibility of using the analysis of the importance of features in cooperation with other methods was considered. In this case, the idea of sensitivity analysis is to use a combination of feature importance analysis [31] with the permutation importance method [32]. The core of this combination is an artificial neural network model for solving multi-class classification problems explicitly trained on data labeled with clustering models.

Permutation importance is a popular technique for evaluating the impact of individual input parameters on neural network predictions [32]. The method works by randomly changing the value of a specific variable in the training data and observing changes in the forecasts of the ANN model – the more significant the change in the model forecast, the higher the importance of this parameter.

4. 6. Autoencoder model and hidden space analysis

Autoencoders are a class of neural networks designed for unsupervised learning that aim to encode input data into a lower dimensional representation known as latent space [33, 34]. The encoder component compresses the input into this reduced representation, and the decoder reconstructs the original input from the encoded representation. The hidden space serves as a collapsed, multifunctional representation of the data, which captures its main characteristics. Autoencoders are widely used for tasks such as data compression, variable learning, and anomaly detection, where the structure and patterns within the latent space play a critical role in extracting meaningful information from the input data [33, 34]. The quality of the hidden space is fundamental to the performance of autoencoders, as it determines the ability of the model to capture and represent the main characteristics of the input data in a compact and meaningful way.

A convolution-based autoencoder best suits this problem because it can effectively reduce the size of embeddings without using many weights. However, the problem lies in the predefined size of the inputs and outputs of the model. ANN cannot be used for this configuration. Therefore, an input/output matrix size of 500×768 was used, where 768 is the size of the vector of token embeddings, and 500 is the number of tokens. The number 500 was chosen because most terms in the dataset have token counts less than 500, so zero padding was added for most, and terms that were too long were truncated. The architecture with such an autoencoder is shown in Fig. 2.

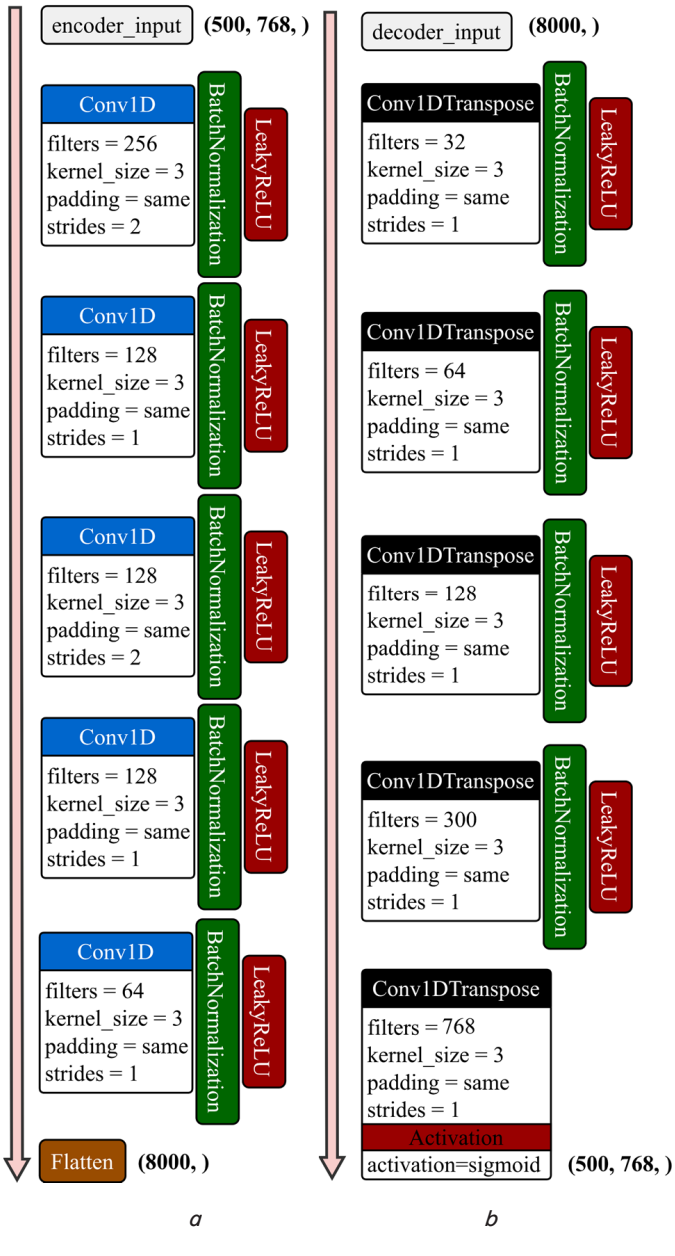


Fig. 2. Architecture of the autoencoder model, which includes parts: *a* – encoder; *b* – decoder

The developed encoder (Fig. 2, *a*) contains 5 Conv1D layers with a decrease in dimensionality after every second layer, which allows to transform the matrices of term embeddings into latent space vectors with a size of 8 thousand. The idea of the decoder (Fig. 2, *b*) reflects an encoder with 5 layers of Conv1DTranspose, which allows the transformation of hidden space vectors back into embedding matrices.

5. Results of research of feature extraction from lambda terms

5. 1. The effectiveness analysis of the selected method of machine learning as a means of extracting features from lambda terms

Using 4,000 generated terms as input to the Microsoft CodeBERT model, embedding matrices were obtained. Average embedding vectors are obtained by further applying the matrix reduction procedure shown in section 4. 1. Average embedding vectors can be visualized using principal component analysis (PCA) [35] and t-distributed stochastic neighbor embedding (t-SNE) [36]. The results of such data compression of average attachments are shown in Fig. 3, *a* for PCA and Fig. 3, *b* for t-SNE with the coloring of the best reduction strategy for terms. As shown in Fig. 3, an equal strategy priority (RI=LO) can be visually separated from LO and RI priorities. However, the priorities of the LO and RI strategies are almost impossible to separate. Furthermore, given this compression, it was investigated whether an automatic search for the appropriate strategy is possible using the information represented in the space of average embeddings.

Fig. 4 shows the proposed autoencoder's training result. The plot was achieved by reducing the PCA size from 8k to 2 principal components. As shown in Fig. 4, it is still difficult to distinguish the RI-priority and LO-priority terms from each other, but the RI=LO terms can be separated.

Fig. 3, *a* shows the results of PCA compression of average embeddings, which differ from the latent space achieved using an autoencoder (Fig. 4). However, the main idea is presented, and the fact that the terms are inseparable from the proposed priority remains.

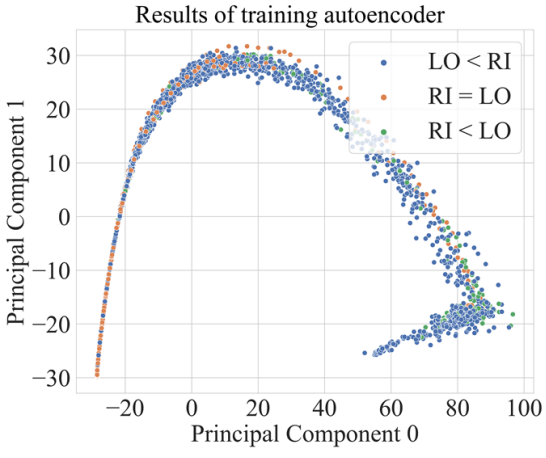
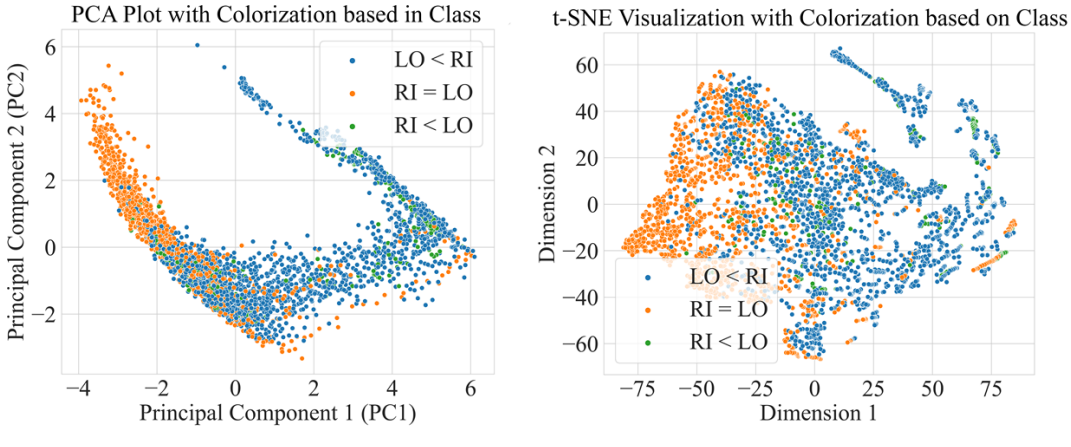


Fig. 4. 8k hidden space represented with PCA compression in 2D space

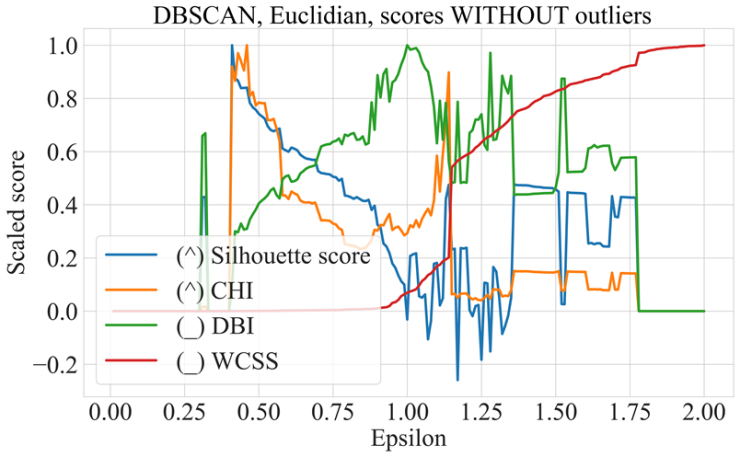
5. 2. Carrying out cluster analysis of lambda term data

After considering the selected clustering methods and metrics for evaluating the clustering quality, the appropriate epsilon value was chosen for DBSCAN clustering on 4 thousand vectors of average embeddings (Fig. 5, 6). Selecting the epsilon value is difficult for this data set due to the conflicting values of the clustering quality measures and the requirement to minimize the number of outliers. In general, WCSS cannot use the Elbow method for Euclidean and cosine metric clustering. Maximization is considered for Silhouette and CHI estimation, as well as for DBI minimization. Therefore, epsilon=1.25 was chosen for DBSCAN based on the Euclidean metric. Additionally, epsilon=0.00205 was chosen for cosine-based DBSCAN.

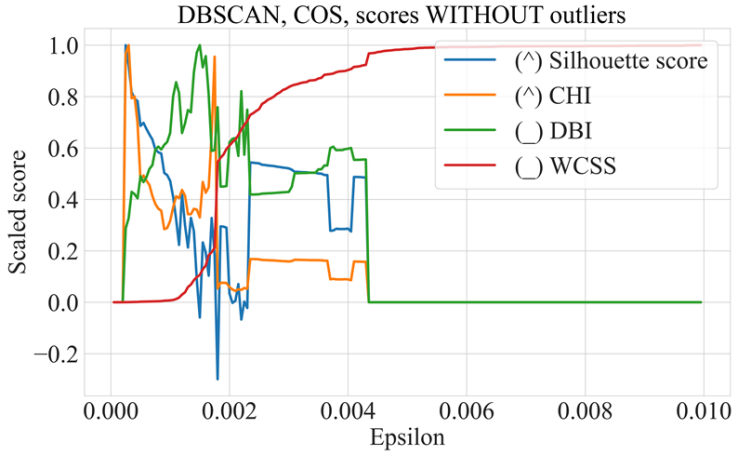


a *b*

Fig. 3. Comparison of the value of the average embeddings with the indication of the strategy priority according to the compression algorithms:
a — principle components: *b* — t-stochastic distribution

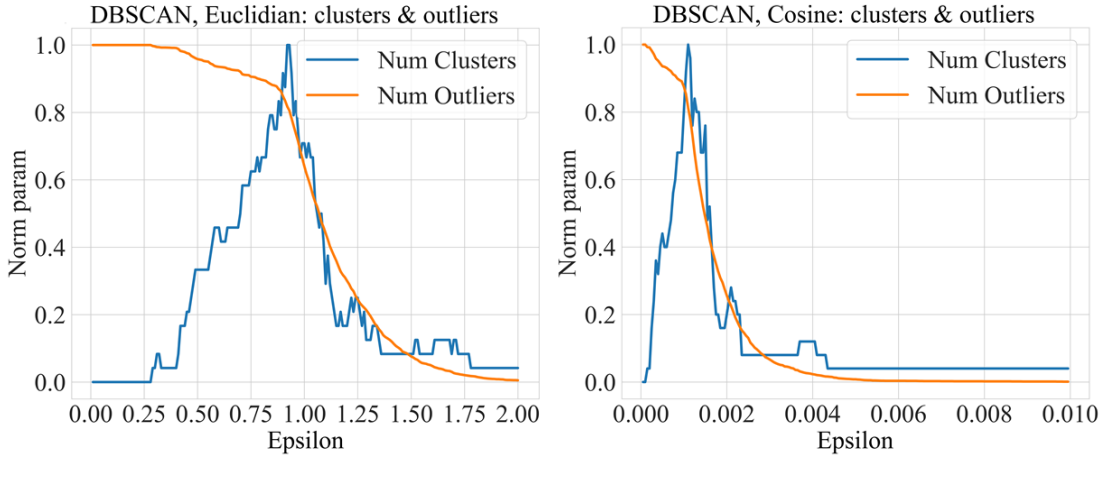


a



b

Fig. 5. Setting the DBSCAN epsilon hyperparameter by comparing the clustering quality (Silhouette, CHI, DBI, WCSS metrics) at interelement distances: *a* – Euclidean; *b* – cosine



a *b*

Fig. 6. Setting the DBSCAN epsilon hyperparameter by comparing the number of clusters and outliers at interelement distances: *a* – Euclidean; *b* – cosine

The next step was the selection of appropriate counting clusters for HAC on 4,000 average data embeddings (Fig. 7). Four metrics were considered: Euclidean, cosine, L1, and L2. For this case, the WCSS metric was applied using the Elbow method, so extreme locations on the graphs and values around those locations were considered. Corresponding rules were used for other metrics: Silhouette and CHI should be maximized, and DBI should be minimized. Choosing the optimal number of clusters gave 5 clusters for Euclidean distance, 6 for cosine similarity, and 5 for L1 and L2 distances.

Fig. 8 shows the results of clustering using DBSCAN and HAC with different metrics. It also shows the PCA honey compression results, highlighted in different colors by the corresponding color. It should also be noted that for the DBSCAN results (Fig. 8, *a*, *b*), a cluster with the number "–1" is indicated, which is an outlier and indicates data that cannot be assigned to a specific cluster. In further calculations, such data were marked as a separate cluster.

The cosine metric (popular in NLP programs [21, 22]) does not provide sufficiently different results. However, as for DBSCAN (Fig. 8, *b*) and for HAC (Fig. 8, *c*), the results do not provide significant differences from the Euclidean metric for DBSCAN (Fig. 8, *a*) and HAC (Fig. 8, *c*). HAC with L2 metric (Fig. 8, *e*) looks similar to HAC with Euclidean (Fig. 8, *c*) and cosine (Fig. 8, *d*). The results of DBSCAN (Fig. 8, *a*, *b*) look like they can distinguish terms by their strategy priorities.

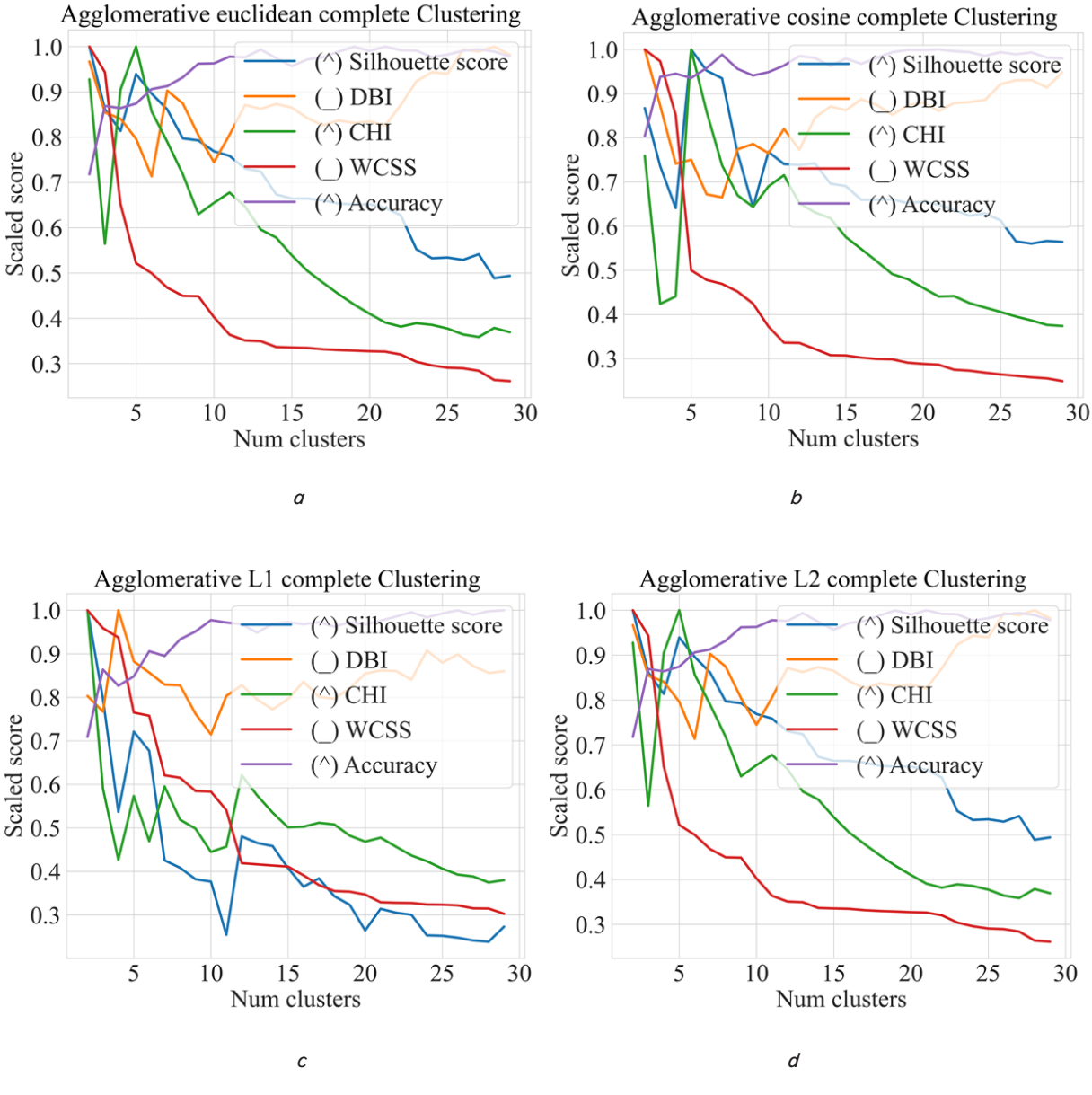


Fig. 7. Selection of the appropriate number of clusters combining Silhouette, DBI, CHI, and WCSS indicators for HAC with metrics:
a – Euclidean; *b* – cosine; *c* – L1; *d* – L2

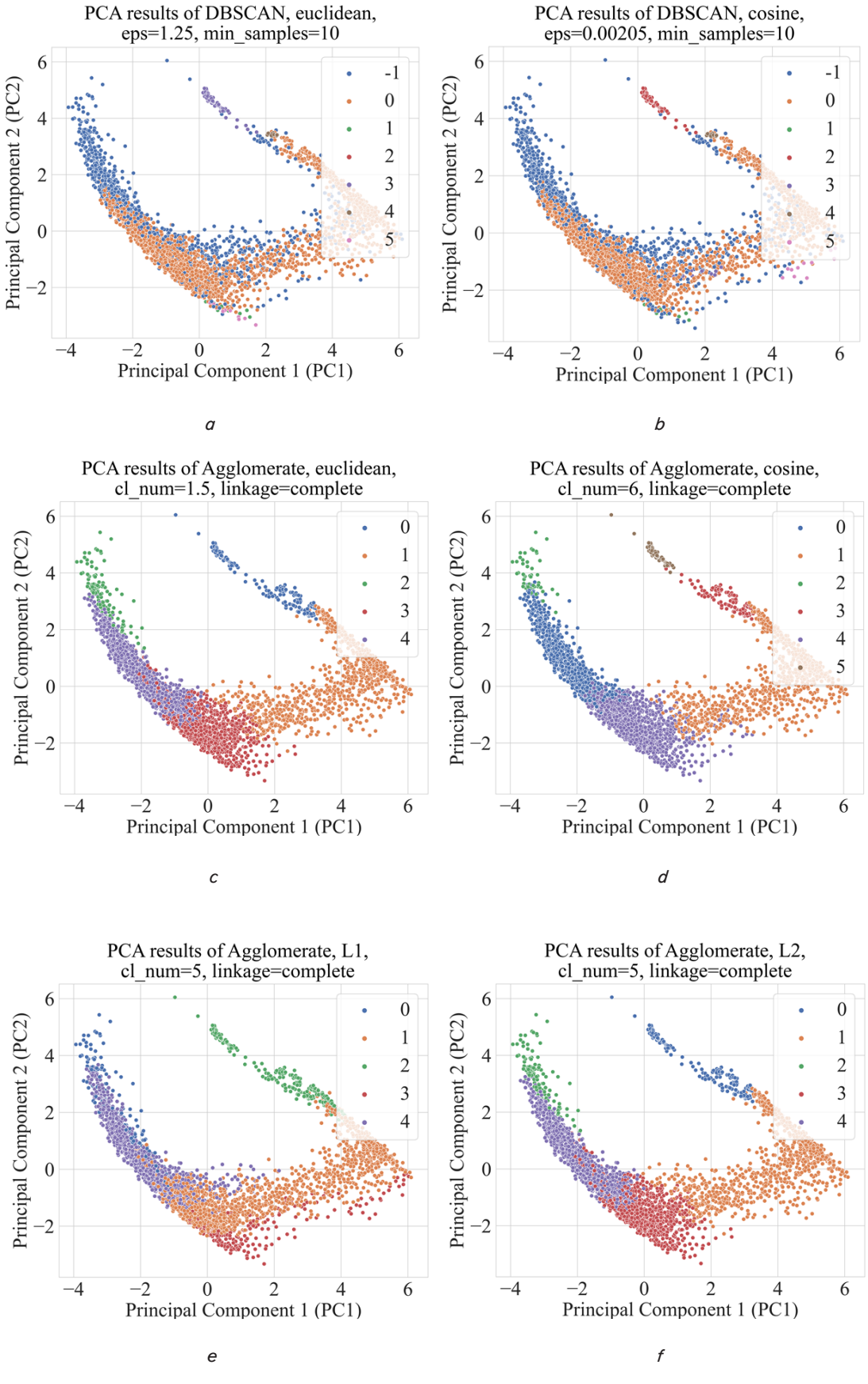


Fig. 8. Compression of the space of average embeddings with visualization of clustering results: *a* – DBSCAN, Euclidean; *b* – DBSCAN, cosine; *c* – HAC, Euclidean; *d* – HAC, cosine; *e* – HAC, L1; *f* – HAC, L2

5. 3. Checking the dependence between the obtained features and clusters

Table 2 shows the 5 most informative parameters in selected clustering models obtained by combining feature importance analysis with the permutation importance method.

Table 2

Top 5 most informative variable results of cluster analysis of average embedding vectors

Informativeness rank	DBSCAN, Euclidean	DBSCAN, cosine	HAC, Euclidean	HAC, cosine	HAC, L1	HAC, L2
1st	var_183	var_183	var_637	var_372	var_1	var_71
2nd	var_371	var_283	var_71	var_15	var_70	var_637
3rd	var_47	var_47	var_183	var_382	var_71	var_183
4th	var_284	var_117	var_229	var_260	var_683	var_43
5th	var_7	var_371	var_453	var_227	var_117	var_229

Table 2 indicates that some variables are the most informative, marked with the same colors. The most common are var_183, which appear in the DBSCAN and HAC models. Other variables are the same for DBSCAN (var_371, var_47) and HAC (var_71, var_229). Admittedly, HAC with the cosine metric does not have in the top 5 informative variables that appear in other configurations of the method. In this case, it may be related to the number of clusters, which means that the cluster configuration affects the most informative variables.

5. 4. Verification of the dependence between the obtained clusters and reduction strategies

The results shown in Table 3 indicate that the three largest overlap values were in the clustering models DBSCAN with Euclidean and cosine metrics and HAC with cosine metrics.

Table 3

The overlapping value of cluster analysis results

Strategy	DBSCAN, Euclidean	DBSCAN, cosine	HAC, Euclidean	HAC, cosine	HAC, L1	HAC, L2
LO-best	82.92 %	82.75 %	64.42 %	82.25 %	64.55 %	64.42 %
RI-best	0 %	0 %	0 %	0 %	0 %	0 %
LO=RI	42.24 %	39.32 %	73.47 %	67.31 %	72.81 %	73.47 %
Overall	83.02 %	86.44 %	72.60 %	78.43 %	69.91 %	72.60%

Furthermore, it is worth acknowledging that the pure accuracy of the RI-best priority class indicates that the unsupervised approach cannot distinguish the RI-best terms from others. However, the approach could distinguish LO=RI from LO and RI terms, and the most successful models for this task were HACs. Nevertheless, due to the imbalance of the data set according to the corresponding priority classes, the overall accuracy was higher in the DBSCAN models.

6. Discussion of research results on feature extraction from lambda terms

The advantage of this research is the use of advanced machine learning methods to highlight hidden features of terms. These features could potentially indicate priority in a particular reduction strategy. The discovery of such features in Lambda Calculus will allow to assume that such features can be found in other functional programming languages. Also, using LLM to search features is impractical due to computational inefficiency. However, it will show that with LLMs, it is possible to find features indicating the priority of a strategy, and using simpler techniques (such as pattern search), it is possible to apply them in practice). This will improve the efficiency of compilers and interpreters of functionally oriented programming languages in the future. However, it made it possible to test a new way of applying LLM as a research method. These advantages are justified by the greater capabilities of machine learning methods to analyze many terms and the ability to use trained LLMs for feature extraction. What differs from [11] is that such features for the interpreter of a functional programming language were obtained by applying expert logic. The research considered the Lambda Calculus Environment [2, 3], which, unlike CompilerGym [5], provides an opportunity to work with Lambda Calculus and generate artificial terms. Where CompilerGym is used as an experimental environment.

Several experiments were conducted using advanced machine learning methods to extract features from lambda terms. At the preparatory stage of the research, 4,000 terms were generated and played through LLM for tasks related to software code analysis. This made it possible to prepare matrices of embeddings, which carried meaningful content that characterized each term. However, using embeddings to analyze informativeness introduces limitations because it is necessary to compare data of different sizes. In order to solve this limitation, average embedding vectors and hidden space vectors obtained using an autoencoder were proposed. Fig. 3 and Fig. 4 show the results of the analysis of such vectors; it is shown that the vectors of average embeddings provide a broader idea of terms. As a result, the Microsoft CodeBERT model can be used to obtain vectors of average embeddings. This can be explained by the limitations on the size of the autoencoder input data and the relatively small size of the autoencoder model itself.

Next, a cluster analysis was performed using the DBSCAN and HAC methods. It is shown that the specified methods with selected metrics (Fig. 7) can be applied to select features in a dataset. However, the cosine metric did not provide superior results for both methods, which can be explained by the large size of the data and the specificity of the results obtained from the Microsoft CodeBERT model.

Next, the dependencies between the obtained features and clusters were checked. The results of the most informative variables are shown in Table 2. It is noted that the same variables are informative for most of the clustering results. The similarity of the obtained results can explain this. The dependence between the received clusters and the strategy's priority was also checked. It is shown in Table 3 that the RI-best terms are almost impossible to separate from the others, but the other priors had quite acceptable separation accuracy. Such results can be explained by insufficient awareness of the CodeBERT model in the terms analysis, which led to an insufficiently informative space of average embeddings. Also, the artificial origin of the considered terms and the blurring of the strategy priority are possible reasons.

The results of compressing the space of medium embeddings (indicated in Fig. 3, 4) and the overlap results (indicated in Table 3) show the proposed approach's promisingness. Machine learning methods for extracting certain features of program code can be used to optimize compilers or interpreters.

The study's limitations are the imprecise determination of the priority of the term strategy: the calculation of the steps of reduction to two strategies was removed. Also, limitations are the use of the CodeBert model, which was initially trained in other programming languages (Go, Java, Python, and others), which can lead to the incorrect representation of lambda calculus terms in embedding matrices, as well as to the problem of converting these matrices into a universal representation for further analysis. The study's shortcoming is the assumption that using LLM as a tool for representing terms can sufficiently accurately determine the features that induce priority in the reduction strategy.

Given these limitations, further research can be conducted using the retrained LLM model for problems related to Lambda Calculus. Models for text summarization can also be used, which will solve the problem of data loss during the transition to the space of medium embeddings.

7. Conclusions

1. This study transformed lambda terms into 768-dimensional average embedding vectors using the Word2Vec methodology and the Microsoft CodeBERT model. It also compared the space of average embeddings to the alternative in the form of a latent space. Through PCA and t-SNE analysis of the space visualizations of mean embeddings, indications were seen that the representation of lambda terms in these mean embeddings could be clearly distinguished. This confirmed the initial hypothesis of identifying relationships using cluster analysis.

2. The study further examined data clustering using the DBSCAN method using both Euclidean and cosine metrics and the HAC method using Euclidean, cosine, L1, and L2 metrics. This clustering effort highlighted the effectiveness of the CodeBERT model in extracting meaningful features from lambda terms. Despite this, the versatility of Microsoft CodeBERT, trained in different programming languages, introduces complexity in accurately representing lambda calculus terms in embedding matrices. This complexity extends to transforming these matrices into comprehensible average embeddings or latent space vectors, especially when using autoencoders.

3. The sensitivity analysis of the variables was conducted to assess the informativeness of the variables in the identified clusters. This made it possible to determine the influence of specific variables on the clustering results and to reveal that certain most informative variables are the same regardless of the chosen clustering method. This is explained by the existence of some multidimensional structures, which different clustering methods define in approximately the same way.

4. The introduction of the overlap factor facilitated the assessment of interdependence between clusters and applied strategies. This evaluation found no correlation between the previously determined strategy priorities and the actual achieved priority of the strategies. This indicates a potential need to fine-tune the CodeBERT model or consider alternative models more suitable for feature extraction in this domain.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

Data will be provided upon reasonable request.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

References

1. Pollak, D., Layka, V., Sacco, A. (2022). Functional Programming. Beginning Scala 3, 79–109. https://doi.org/10.1007/978-1-4842-7422-4_4
2. Deineha, O., Donets, V., Zholtkevych, G. (2023). On Randomization of Reduction Strategies for Typeless Lambda Calculus. Communications in Computer and Information Science, 25–38. https://doi.org/10.1007/978-3-031-48325-7_3
3. Deineha, O., Donets, V., Zholtkevych, G. (2023). Estimating Lambda-Term Reduction Complexity with Regression Methods. Information Technology and Implementation 2023. Available at: https://ceur-ws.org/Vol-3624/Paper_13.pdf
4. 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). https://doi.org/10.1109/icse43902.2021.00110
5. Cummins, C., Wasti, B., Guo, J., Cui, B., Ansel, J., Gomez, S. et al. (2022). CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research. 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). https://doi.org/10.1109/cgo53902.2022.9741258
6. Martins, L. G. A., Nobre, R., Cardoso, J. M. P., Delbem, A. C. B., Marques, E. (2016). Clustering-Based Selection for the Exploration of Compiler Optimization Sequences. ACM Transactions on Architecture and Code Optimization, 13 (1), 1–28. https://doi.org/10.1145/2883614
7. Ashouri, A. H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S., Cavazos, J. (2017). MiCOMP. ACM Transactions on Architecture and Code Optimization, 14 (3), 1–28. https://doi.org/10.1145/3124452

8. de Souza Xavier, T. C., da Silva, A. F. (2018). Exploration of Compiler Optimization Sequences Using a Hybrid Approach. Computing and Informatics, 37 (1), 165–185. https://doi.org/10.4149/cai_2018_1_165
9. Mammadli, R., Jannesari, A., Wolf, F. (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). https://doi.org/10.1109/llvmhpchipar51896.2020.00006
10. Runciman, C., Wakeling, D. (1993). Heap Profiling of a Lazy Functional Compiler. Workshops in Computing, 203–214. https://doi.org/10.1007/978-1-4471-3215-8_18
11. Chlipala, A. (2015). An optimizing compiler for a purely functional web-application language. Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming. https://doi.org/10.1145/2784731.2784741
12. 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 2023. Available at: https://ceur-ws.org/Vol-3641/paper12.pdf
13. Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. et al. (2017). Attention is All you Need. arXiv. https://doi.org/10.48550/arXiv.1706.03762
14. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y. et al. (2023). A Survey of Large Language Models. arXiv. https://doi.org/10.48550/arXiv.2303.18223
15. Ormerod, M., del Rincón, J. M., Devereux, B. (2024). How Is a "Kitchen Chair" like a "Farm Horse"? Exploring the Representation of Noun-Noun Compound Semantics in Transformer-based Language Models. Computational Linguistics, 50 (1), 49–81. https://doi.org/10.1162/coli_a_00495
16. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. et al. (2023). Code Llama: Open Foundation Models for Code. arXiv. https://doi.org/10.48550/arXiv.2308.12950
17. Replit. replit-code-V1-3B. Hugging Face. Available at: https://huggingface.co/replit/replit-code-v1-3b
18. Elnaggar, A., Ding, W., Jones, L., Gibbs, T., Feh r, T. B., Angerer, C. et al. (2021). CodeTrans: Towards Cracking the Language of Silicon's Code Through Self-Supervised Deep Learning and High Performance Computing. arXiv. https://doi.org/10.48550/arXiv.2104.02443
19. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M. et al. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. Findings of the Association for Computational Linguistics: EMNLP 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.139
20. Wang, Y., Wang, W., Joty, S., Hoi, S. C. H. (2021). CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. https://doi.org/10.18653/v1/2021.emnlp-main.685
21. Styawati, S., Nurkholis, A., Aldino, A. A., Samsugi, S., Suryati, E., Cahyono, R. P. (2022). Sentiment Analysis on Online Transportation Reviews Using Word2Vec Text Embedding Model Feature Extraction and Support Vector Machine (SVM) Algorithm. 2021 International Seminar on Machine Learning, Optimization, and Data Science (ISMODE). https://doi.org/10.1109/ismode53584.2022.9742906
22. Dwivedi, V. P., Shrivastava, M. (2017). Beyond Word2Vec: Embedding Words and Phrases in Same Vector Space. ICON. Available at: https://aclanthology.org/W17-7526.pdf
23. Hartigan, J. A., Wong, M. A. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. Applied Statistics, 28 (1), 100. https://doi.org/10.2307/2346830
24. Hahsler, M., Piekenbrock, M., Doran, D. (2019). dbscan: Fast Density-Based Clustering with R. Journal of Statistical Software, 91 (1). https://doi.org/10.18637/jss.v091.i01
25. Zhang, Y., Li, M., Wang, S., Dai, S., Luo, L., Zhu, E. et al. (2021). Gaussian Mixture Model Clustering with Incomplete Data. ACM Transactions on Multimedia Computing, Communications, and Applications, 17 (1s), 1–14. https://doi.org/10.1145/3408318
26. Monath, N., Dubey, K. A., Guruganesh, G., Zaheer, M., Ahmed, A., McCallum, A. et al. (2021). Scalable Hierarchical Agglomerative Clustering. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. https://doi.org/10.1145/3447548.3467404
27. Shahapure, K. R., Nicholas, C. (2020). Cluster Quality Analysis Using Silhouette Score. 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA). https://doi.org/10.1109/dsaa49011.2020.00096
28. Ros, F., Riad, R., Guillaume, S. (2023). PDBI: A partitioning Davies-Bouldin index for clustering evaluation. Neurocomputing, 528, 178–199. https://doi.org/10.1016/j.neucom.2023.01.043
29. Lima, S. P., Cruz, M. D. (2020). A genetic algorithm using Calinski-Harabasz index for automatic clustering problem. Revista Brasileira de Computação Aplicada, 12 (3), 97–106. https://doi.org/10.5335/rbca.v12i3.11117
30. Li, X., Liang, W., Zhang, X., Qing, S., Chang, P.-C. (2019). A cluster validity evaluation method for dynamically determining the near-optimal number of clusters. Soft Computing, 24 (12), 9227–9241. https://doi.org/10.1007/s00500-019-04449-7
31. Chung, H., Ko, H., Kang, W. S., Kim, K. W., Lee, H., Park, C. et al. (2021). 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 (4), e27060. https://doi.org/10.2196/27060

32. Pereira, J. P. B., Stroes, E. S. G., Zwinderman, A. H., Levin, E. (2022). Covered Information Disentanglement: Model Transparency via Unbiased Permutation Importance. Proceedings of the AAAI Conference on Artificial Intelligence, 36 (7), 7984–7992. https://doi.org/10.1609/aaai.v36i7.20769
33. Chen, X., Ding, M., Wang, X., Xin, Y., Mo, S., Wang, Y. et al. (2023). Context Autoencoder for Self-supervised Representation Learning. International Journal of Computer Vision, 132 (1), 208–223. https://doi.org/10.1007/s11263-023-01852-4
34. Yin, C., Zhang, S., Wang, J., Xiong, N. N. (2022). Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52 (1), 112–122. https://doi.org/10.1109/tsmc.2020.2968516
35. Niedoba, T. (2014). Multi-parameter data visualization by means of principal component analysis (PCA) in qualitative evaluation of various coal types. Physicochemical Problems of Mineral Processing, 50 (2), 575–589. Available at: https://bibliotekanauki.pl/articles/109595
36. Oliveira, F. H. M., Machado, A. R. P., Andrade, A. O. (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, 1–17. https://doi.org/10.1155/2018/8019232