UDC 004.4'422-048.34 **doi: 10.32620/reks.2024.2.02**

Oleksandr DEINEHA

V. N. Karazin Kharkiv National University, Kharkiv, Ukraine

SUPERVISED DATA EXTRACTION FROM TRANSFORMER REPRESENTATION OF LAMBDA-TERMS

The object of this research is the *process of compiler optimization, as it is essential in modern software development, particularly in functional programming languages like Lambda Calculus. Optimization strategies directly impact interpreter and compiler performance, influencing resource efficiency and program execution. While functional programming compilers have garnered less attention regarding optimization efforts than their object-oriented counterparts, Lambda Calculus's complexity poses unique challenges. Bridging this gap requires innovative approaches like leveraging machine learning techniques to enhance optimization strategies. This study focuses on leveraging machine learning to bridge the optimization gap in functional programming, particularly within the context of Lambda Calculus. This study delves into the extraction features from Lambda terms related to reduction strategies by applying machine learning. Previous research has explored various approaches, including analyzing reduction step complexities and using sequence analysis Artificial Neural Networks (ANNs) with simplified term representation.* ***This research aims*** *to develop a methodology for extracting comprehensive term data and providing insights into optimal reduction priorities by employing Large Language Models (LLMs).* ***Tasks*** *were set to generate embeddings from Lambda terms using LLMs, train ANN models to predict reduction steps, and compare results with simplified term representations. This study employs a sophisticated blend of machine learning algorithms and deep learning models as a* ***method*** *of analyzing and predicting optimal reduction paths in Lambda Calculus terms.* ***The result*** *of this study is a method that showed improvement in determining the number of reduction steps by using embeddings.* ***Conclusions:*** *The findings of this research offer significant implications for further advancements in compiler and interpreter optimization. This study paves the way for future research to enhance compiler efficiency by demonstrating the efficacy of employing LLMs to prioritize normalization strategies. Using machine learning in functional programming optimization opens avenues for dynamic optimization strategies and comprehensive analysis of program features.*

Keywords: *Lambda Calculus; functional programming language; strategy optimization; Large Language Model; code embeddings.*

Introduction

Modern software development relies on functional programming languages, which provide robust solutions to problems in modern development [1]. With an increasing emphasis on efficiency, compiler optimization becomes essential. Lambda Calculus represents functional programming languages, where the key challenge lies in deciphering program code to unveil reduction strategies, thereby improving compiler and interpreter performance [2]. Understanding how the program execution state varies based on the chosen execution strategy aids in selecting appropriate, resource-efficient, resilient interpreter and compiler methods. These strategies are pivotal in enhancing compilers and interpreters, thereby benefiting functional and object-oriented languages.

Exploring Lambda Calculus allows one to emulate interpreters and compilers as they seek optimal reduction strategies. Our methodology yields diverse Lambda terms, creating a solid foundation for testing various approaches to enhance reduction quality [2, 3]. The intricate task of determining whether to devise tailored strategies for individual terms or opt for a global approach, like the Rightmost Innermost method, provides valuable insights into the complexities of reduction. We contemplate employing sophisticated machine learning techniques to extract term data and uncover internal relationships, potentially enhancing the term reduction process.

Although, functional programming compilers have received comparatively less attention in terms of optimization efforts. Previous studies have explored areas like heap profiling [4] and hand-crafted logic optimization [5] for functional compilers, albeit on a smaller scale. Functional languages like Haskell and OCaml often employ specific reduction strategies using specialized mechanisms such as call-by-need and call-by-value.

Considering the landscape above, the challenge of optimizing functional programming compilers and interpreters becomes evident. While machine learning techniques have effectively optimized object-oriented compilers, applying them to functional code introduces unique complexities [6]. Therefore, this study aims to address this gap by extracting features from functional code and leveraging machine learning to enhance the quality of optimization strategies.

Paper structure. In Section 1, "The Current State of Research in the Lambda Term Normalization Process," we review existing research on Lambda Calculus, focusing on strategies for optimizing Lambda term normalization and the potential of machine learning techniques to refine these processes.

Section 2, "Motivation and Problem Statement", elaborates on the challenges in optimizing compilers and interpreters for functional programming languages, proposing the use of large language models (LLMs) and artificial neural networks (ANNs) to automate optimization processes based on program features.

In Section 3, "Objective and Approach", the paper outlines the specific objectives of using LLMs and ANNs to enhance feature extraction from Lambda terms, thus improving the understanding and performance of compilers and interpreters.

Section 4, "Materials and Methods of Research", details the methods used for generating and processing data, including the use of various LLMs to transform Lambda terms into meaningful vectors and the setup for predicting normalization strategies using ANN models.

Section 5, "Results", presents findings from the experiments, mainly focusing on the predictive performance of ANNs in determining optimal normalization strategies and comparing these to baseline models.

Section 6, "Discussion", reviews the findings' implications, the effectiveness of LLMs in feature extraction, and potential improvements in compiler and interpreter technologies for functional programming languages.

The paper ends with the Conclusion section, which summarizes the study's findings, highlights advances in understanding and optimizing Lambda term normalization, and suggests future research directions to refine these approaches further.

1. Current state of research in the Lambda term normalization process

Research on Lambda Calculus was performed using a Pure Lambda Calculus environment developed by the authors [2, 3]. This environment was used to optimize the Lambda terms normalization process. This study proposed a new method for analyzing the reduction step [3]. The proposed method considers that normalization (term reduction) steps can have different reduction complexities. For verification, this suggestion used base tree term characteristics and Machine Learning methods for data extraction. Such a suggestion allows for estimating each redex complexity and always choosing the least complex reduction. This will enable a greedy reduction strategy based on estimating redex reduction complexity.

Many studies have investigated another approach to analyzing Lambda terms. This approach does not involve building a new strategy but rather choosing the optimal strategy from a pool of strategies. In this case, the estimation approach compares the estimation of the count reduction steps or indicates if the strategy is better. Research [7] analyzed the LO strategy's influence on the depth increase of Lambda terms. Conversely, the work [8] showed that almost every simply typed Lambda term has a long beta-reduction sequence, which means that finding the worst possible reduction path for terms is possible. The use of the randomized strategy for term reduction and its influence on the beta reduction path was considered in the work [9]. Research [10] considered the consumption of computational resources of normalization strategies. None of them did not select term characteristics, which might indicate strategy preference. However, work [11] considered using simplified term strings as input on Artificial Neural Network (ANN) models to predict the number of the LO and RI steps. The predicted numbers can be used as strategy prediction indicators, which means that a smaller predicted number is a better strategy. Although using ANN is not computationally effective, analyzing trained ANN can answer questions about term features that indicate strategy priority. Using these features can be easily implemented in modern compilers. However, note that a significant disadvantage of this research is the use of simplified term strings, which lose information about term variables, impacting the reduction process.

In addition, promising research has been conducted on solving mathematical problems [12], code execution [13], and compilation improvement [14] using ANNs on Transformer models. Transformer models for processing natural, machine, or mathematical text information are called Large Language Models (LLM). LLM was used to analyze typed Lambda terms to detect term type, as shown in the article [15]. LLM has demonstrated the ability to analyze and understand Lambda terms to predict their type. Using string term representation while keeping its variable names and encoding each type of symbol was considered in the article [15]. The relatively small size of LLM is used in the article [15] allows the keeping of only 32 term variables and special symbols for application and abstraction. Moreover, training such as LLM still requires hours of work on modern graphical units.

2. Motivation and problem statement

Compilers and interpreters are based on unique optimization methods that decrease computational consumption during compilation, interpretation, and execution [14]. The main issue with such methods is improving their efficiency with many possible program variants. This research considered automatizing the selection of specific program features, influencing the computational efficiency of compilation, interpretation, or execution. It might help create a universal optimization technique that analyzes actual code and generates specific optimizations locally or analyze a considerable amount of code to generate general optimization.

The optimization searching method can be automated using ANNs to statistically analyze data [16]. The program contains sequences of variables, keywords, and operators. The most advanced way to process text information is using LLM [17]. Large enough LLMs can solve various text-related tasks [17]. One such task is to transform text into vectors of meaning, also called embeddings [18]. Embeddings are vectors of characteristics that can be used to solve specific tasks. Analyzing LLM, which generates such vectors and input data, might help identify text features that significantly impact the formation of these features. In other words, LLMs can extract specific characteristics that significantly impact the forming features in the program code. Such characteristics can influence program compilation, interpretation, or execution and can be used to choose dynamic optimization strategies for compilers or interpreters.

Testing on modern functional programming languages poses a significant challenge because of the extensive array of keywords, making it arduous to gather or generate sufficient training data. A plausible solution lies in opting for a programming language with simpler syntax, such as Lambda Calculus, which is characterized by only two operators: Application and Abstraction. However, despite its simplicity, Lambda Calculus is Turing complete [19], rendering it capable of emulating any computational process feasible in other programming languages. Within Lambda Terms, the selection rule governing a specific application, redex, dictates a reduction strategy (or execution sequence), as evidenced in prior studies [7, 8], showcasing its impact on reduction step counts.

Although Lambda Calculus is the most straightforward programming language regarding operator usage, gathering training data is quite resource-consuming. Thus, synthetic training data is an excellent way to cover as many term variants as possible. However, it should be noted that missing some term combinations with important feature indication reduction priority is still possible.

3. Objective and approach

This study improves feature extraction from Lambda terms to better understand the term normalization

process. Such term features can improve the performance of compilers and interpreters. This can be done by applying advanced machine learning techniques to represent Lambda calculus terms as embedding vectors containing specific features. Machine learning methods can also investigate these features by analyzing their relation to the actual number of reduction steps.

Considering all this, the following research solves the following problems:

1. Select a set of LLM models to process the Lambda terms and convert them into embeddings.
2. Configure hyperparameters of ANN models and train them to solve the problem of prediction term reduction steps for specific reduction strategies, as in the work [11].
3. Compare the results of using simplified term representation in the work [11] to achieve accuracy and conclude that such an approach is efficient.

This study deals with the artificially generated dataset of Lambda terms. Previous research converted this dataset into a simplified term representation [11]. The main idea was to save only tree structures to analyze redexes, not variables. Therefore, this research deals with the same dataset but keeps variable information. Keeping variable information in the text representing Lambda terms helps to better analyze terms and indicate specific relations lost in simplified representation. However, keeping variable information increases computational consumption significantly. Therefore, this research considered using general and specific LLMs to collect text embeddings, representing Lambda terms with variable information. Such embeddings can be used as input to some ANN models to solve the reduction step prediction task for a specific normalization strategy. These step predictions are keys to estimating the normalization strategy priority.

Therefore, this article's central hypothesis is that using LLMs to transform Lambda term expressions into embedding vectors can provide a sufficient way of analyzing their optimal reduction priority.

4. Materials and methods of the research

4.1. Machine learning models for text processing

The research considered Lambda term representation as it is in the text. There are numerous machine learning methods and models for processing text information. These methods and models can be split into simple methods (Support Vector Machines, Naive Bayes, Artificial Neural Networks, and others) and complex methods (deep learning architectures: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers) [20 – 22].

Fig. 1 [22] shows an example of the CNN model's typical usage for solving text classification problems. CNN models provide robust solutions for solving simple text problems, like classification. CNN can work as a word indicator, indicating word or phrase availability in text. However, it lacks context understanding and usually understands text content.

Fig. 2 shows another example of text-processing architectures [23]. It shows the units of RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU). The basic idea behind RNN, LSTM, and GRU is the same. Each unit can process each piece of text (usually symbols or words) and use its memory or output to store the entire text meaning in a vector representation called embedding. Such architecture provides a simple way to solve various text-related problems, from simple classification to text generation and translation. Although these architectures are a step forward compared to CNN, they have problems with too long sequences, especially RNN, and training problems due to gradient vanishing [23].

The state-of-the-art text processing models consider the Transformer architecture [20, 21]. The Transformer model architecture is shown in Fig. 3 [20]. The Transformer allows data processing in parallel, as in CNN architecture, but without input text sequence limitations. It can capture long-range dependencies using a self-attention mechanism. This model provides state-of-the-art performance in various text-related tasks [23].

The research compared the considered architectures' performance in estimating term reduction steps [11]. Although the LSTM model showed the best performance, it had sequence length limitations. In addition, simplified term representation without keeping variable information was considered in the research [11]. This limited the Transformer model in terms of its term analysis capabilities.

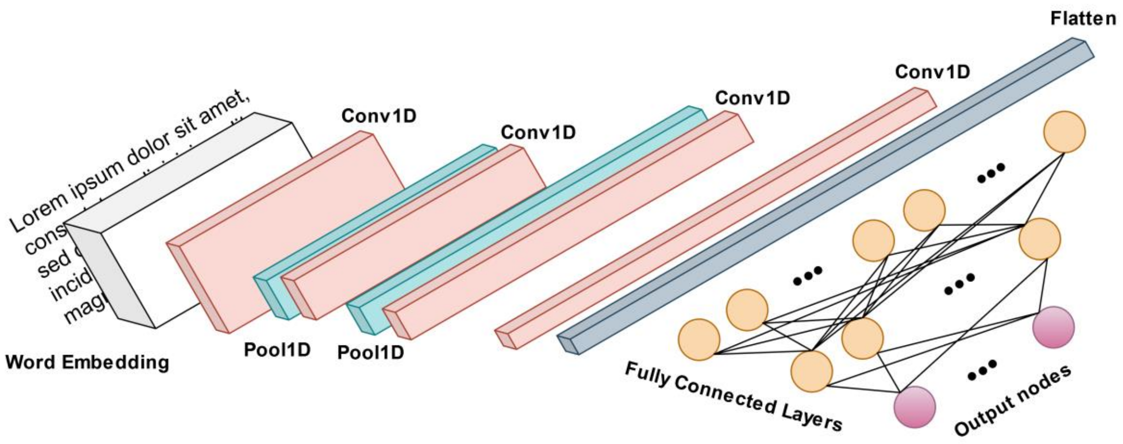

Fig. 1. CNN model architecture for solving the text classification problem

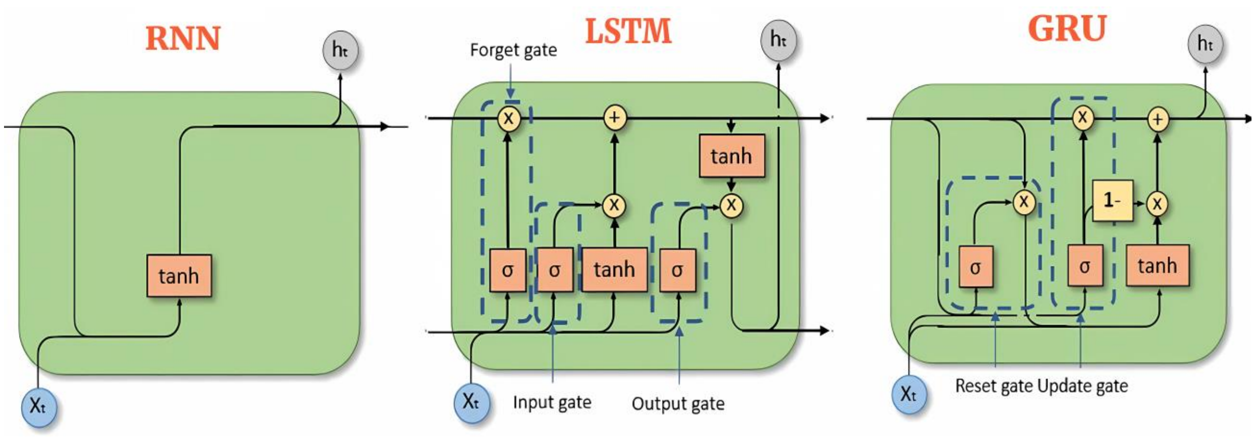

Fig 2. Diagrams of the RNN, LSTM, and GRU units

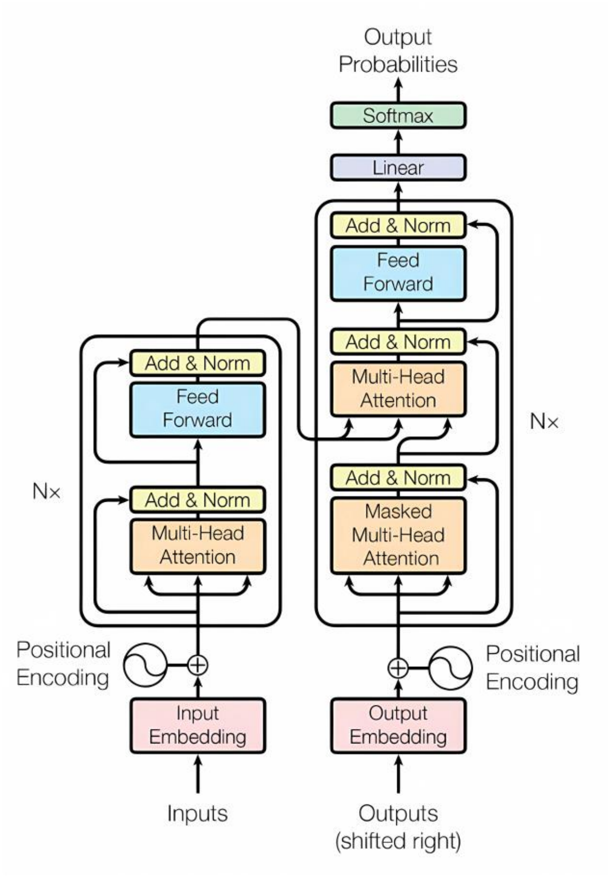

Fig 3. Architecture of the Transformer model

4.2. Large language models for embedding extraction

The latest sophisticated LLMs leverage the Transformer architecture [20, 21], which, owing to the characteristics of ANNs and Transformers, allows the use of intermediate layer outputs as feature vectors. Essentially, LLMs excel in translating textual data into vectors or matrices representing semantic information [20, 24]. Consequently, it becomes feasible to transform Lambda terms into meaningful vectors or matrices, facilitating subsequent analysis. Such vectors of meaning are called embeddings.

Research [6] analyzed popular LLMs related to code analysis and generation tasks. We concluded that the Microsoft CodeBERT model is the most suitable for transforming Lambda terms while keeping its variable information in embedding representation. Although that research focused on prioritizing normalization strategy, it did not consider general-purpose LLMs, which have much larger average weights and input tokens available. In addition, research [6] considers a reinforcement learning approach to analyze embedding space, while this work uses inform learning. The LLMs used for generating embeddings are listed below. The Microsoft CodeBERT is a relatively small model that can be processed locally. However, OpenAI models are proprietary and publicly unavailable, except for API calls, which provide model outputs.

This research aims to compare the results of relatively small and publicly available CodeBERT with general-purpose OpenAI models to show the limits of existing technologies and reliability for further research toward smaller LLMs. If CodeBERT shows lower results comparable to OpenAI models, it can be concluded that smaller LLMs cannot provide enough feature extraction capabilities.

CodeBERT and OpenAI can process Lambda terms as pure text, so data preprocessing is unnecessary. However, for CodeBERT, data postprocessing is required because its output contains the embedding vector vectors. The postprocessing for CodeBERT is explained and shown in the research [6], where average embedding vectors were taken. OpenAI provides embedding vectors without the need for postprocessing.

Comparison of LLMs for generating embeddings:

1. **Microsoft CodeBERT**, a bimodal pre-trained model designed for programming and natural language tasks, employs a Transformer architecture. It incorporates a hybrid objective function that includes a replaced token detection pre-training task. The authors curated datasets comprising code samples from Go, Java, JavaScript, PHP, Python, Ruby, and others for training the model [25]. The embedding vector size is 768.
2. **OpenAI text-embedding-ada-002**. The model supersedes five distinct models catering to text search, text similarity, and code search. It surpasses the previous flagship model, Davinci, in most tasks, all while being priced at a staggering 99.8% lower cost [26, 27]. The embedding vector size is 1536.
3. **OpenAI text-embedding-3-small**. The model is significantly upgraded upgrades over text-embedding-ada-002. It boasts improved performance, with MIRACL scores rising from 31.4% to 44.0% and MTEB scores increasing from 61.0% to 62.3%. Additionally, it offers a 5X price reduction [27, 28]. The embedding vector size is 1536.
4. **OpenAI text-embedding-3-large**. The model is significantly upgraded over text-embedding-ada-002. It boasts improved performance, with MIRACL scores rising from 31.4% to 44.0% and MTEB scores increasing from 61.0% to 62.3%. Additionally, it offers a 5X price reduction [27, 28]. The embedding vector size is 3072.

4.3. Neural network model for reduction steps prediction

The central assumption of this research was that embeddings generated with specific and general LLMs could be suitable for the detection normalization strategy.

A fully connected ANN model was considered to detect term strategy priority. However, simply telling whether the LO strategy is better or worse than the RI strategy is not enough for deep analysis. In this case, the solution can be using the ANN model with a prediction number of reduction steps. Showing the relation between the input embedding vector and output integer values is a problem here. Possible solutions are:

1. Solving the steps prediction problem as a regression problem. It requires accurate data tuning to limit possible predictions and ensure that they are evenly distributed.
2. Solving the steps prediction problem as a classification problem. This allows the model to be more concrete for various data samples. However, it requires removing some samples with too many reduction steps for both the LO and the RI strategies.
3. Solving the steps prediction problem as a bin problem allows the solving of classification problems for specific reduction steps, spreading some reduction steps into units but only a few steps. It creates some degree of freedom for the model but reduces its accuracy.

The training and testing datasets were split 80/20. The training procedure for fully connected ANN models requires 200 training steps. A testing set was used for validation, so the best model weights were saved for the best validation accuracy. Step decay was not considered because of the usage of the Adam optimizer. The categorical cross-entropy loss function was used because it is typical for the multiclass classification problem.

4.4. Experiments setup

Given the findings from research [11], which indicate that most terms are reduced within 0–30 steps, the decision was made to approach the problem as a classification task. So, the typical ANN model architecture is shown in Fig. 4. The ANN model contains a few intermediate layers with the ReLu activation function and 31 units of the output layer with the Softmax activation function for solving classification tasks. In addition, the ANN model contains a Dropout layer to prevent overfitting. Moreover, recognizing the limitations of accuracy metric usage for estimating, also Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics were evaluated [29]. Based on the hypothesis that terms with more reduction steps would result in higher error values, we examined MAE to the actual reduction step count.

Considering all the abovementioned points, the experiments will be processed in the following way:

1. Generate embeddings for each term in the previously generated dataset of terms using LLM models for generating embeddings.
2. Split each embedding dataset into training and testing sets in the same order. This step generates training and testing sets with the same terms and, for each term, the set number of the LO or the RI steps.
3. Predict the number of LO and RI steps using model predictions for the training and testing sets. Using predictions, build MAE graphics to estimate the possibility of the proposed LLM models representing term information as a vector. Compare the MAE result with the best result achieved in the work [11].

5. Results

Fig 5 visualizes training the proposed ANN model for predicting the number of the LO steps for the OpenAI text-embedding-3-small model embeddings. It shows the typical procedure for training all eight possible ANN models. In Fig 5, validation values stop progressing or worsen for loss and accuracy metrics. We consider saving model weights to maximize model performance for the best validation accuracy. Therefore, each trained model uses a different count of training steps but still provides the best performance. In addition, model weights are saved and can be used for further investigation.

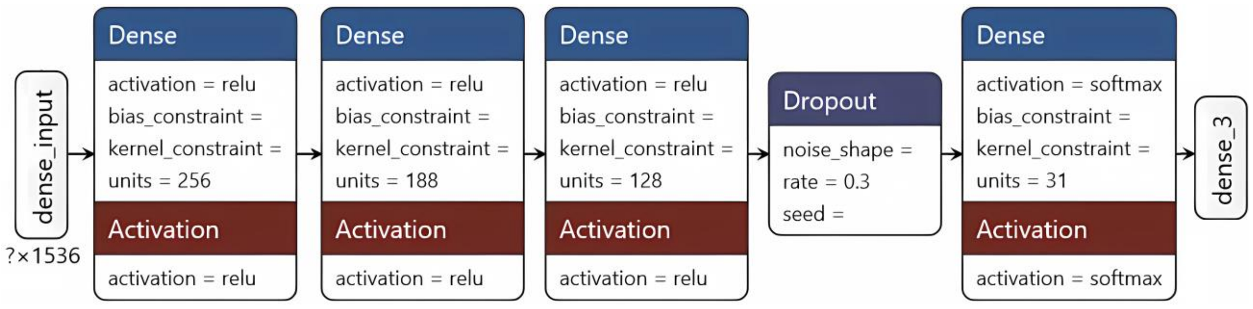

Fig. 4. Architecture of the fully connected ANN model used to estimate the count normalization steps with the selected reduction strategy

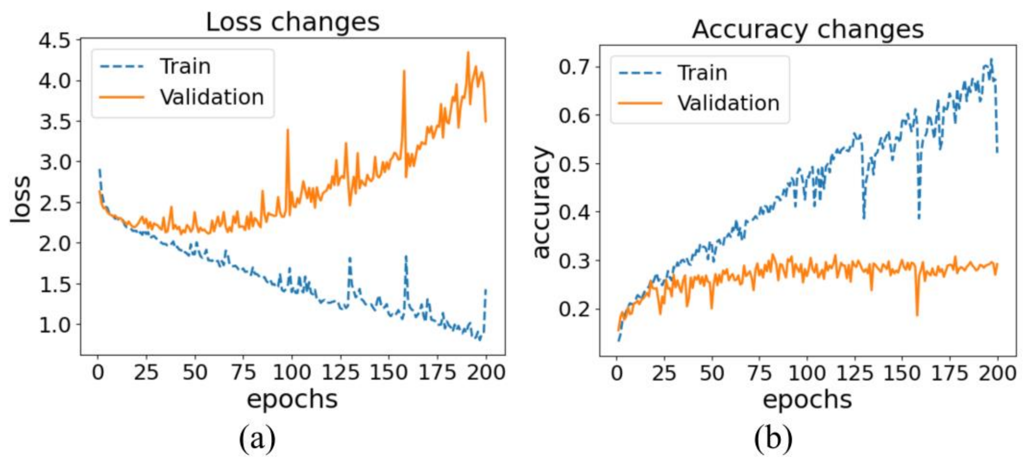

Fig 5. Training of the fully connected ANN model on OpenAI text-embedding-3-small model embeddings to predict the number of LO steps: (a) loss function; (b) accuracy progression

The results of training ANN models on the prediction number of LO reduction steps are shown in Table 1. In addition, the best results of the research [11] of prediction LO reduction steps by the convolution ANN model using simplified term representations are shown in Table 1. As it is represented for testing sets, embeddings can provide improvement up to 3 times. However, the testing set does not provide much improvement, although the mean deviation is almost 0.6 steps, which is a considerable result. In addition, RMSE provides information about deviation to the side of underprediction number of reduction steps for more complex terms.

The results of training ANN models on the prediction number of RI reduction steps are shown in Table 2. In addition, the best results of the research [11] of prediction RI reduction steps by the LSTM ANN model using simplified term representations are shown in Table 2. In this case, using embedding to predict the RI normalization steps does not provide many improvements compared with the best results of the research [11]. Although the MAE of text-embedding-3-large model embeddings shows much better results in the training set, it underperforms in the testing set. However, RMSE results for OpenAI indicate improved model performance with a minor possible maximum error.

Table 1

Result of training ANN model for predicting the number of the LO reduction steps using embeddings or simplified term representation

No	Embeddings source	MAE		RMSE	
		Train	Test	Train	Test
1	Microsoft CodeBERT	2.24	2.41	3.87	3.99
2	OpenAI text-embedding-ada-002	1.36	2.09	2.69	3.50
3	OpenAI text-embedding-3-small	1.07	2.12	2.21	3.61
4	OpenAI text-embedding-3-large	0.95	2.18	2.10	3.72
5	Best results with simplified terms with convolution model [11]	2.91	2.74	5.28	5.16

Table 2

Result of training ANN model for predicting the number of the RI reduction steps using embeddings or simplified term representation

No	Embeddings source	MAE		RMSE	
		Train	Test	Train	Test
1	Microsoft CodeBERT	1.38	1.74	2.60	2.87
2	OpenAI text-embedding-ada-002	0.95	1.53	1.74	2.41
3	OpenAI text-embedding-3-small	0.71	1.46	1.42	2.37
4	OpenAI text-embedding-3-large	0.27	1.51	0.69	2.35
5	Best results with simplified terms with LSTM model [11]	0.50	1.29	1.25	2.7

Fig 6 shows a more specific MAE analysis. Ten plots of the model performance depending on the expected number of reduction steps and the selected strategy are shown in Fig 6. Eight of these plots relate to the analysis of the possibility that LLM models produce embeddings usable in prioritizing normalization strategies. Fig 6, i and Fig 6, j show the best models for predicting LO and Ri steps from the research [11].

The results of the model testing, shown in Fig. 6, indicate that, in most cases, LLMs provide more suitable results than models trained on simplified terms representation (Fig 6, i and Fig. 6, j). The considered LLMs were not explicitly developed to analyze Lambda terms. However, overall MAE or RMSE performance was higher than that achieved on specially designed models in the work [11]. This result can be explained by considering variable information and more weights presented in pretrained LLMs for collecting embeddings.

6. Discussion

This article shows how advanced machine learning methods can be used to highlight hidden features in Lambda terms. These features can be used to estimate which normalization strategy is preferable for a particular Lambda term. Potentially, such features can be extracted in other more computationally effective ways, thus improving the efficiency of compilers and interpreters of functionally oriented programming languages. 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 Lambda term features were obtained using pretrained LLMs and full text to represent Lambda terms instead of simplified representation.

Results show that MAE and RMSE are lower for models trained on the RI (see Table 2) steps prediction than on the LO (see Table 1) steps prediction, which might indicate that detecting the priority of RI terms is more accessible than the LO. Also, it is interesting that

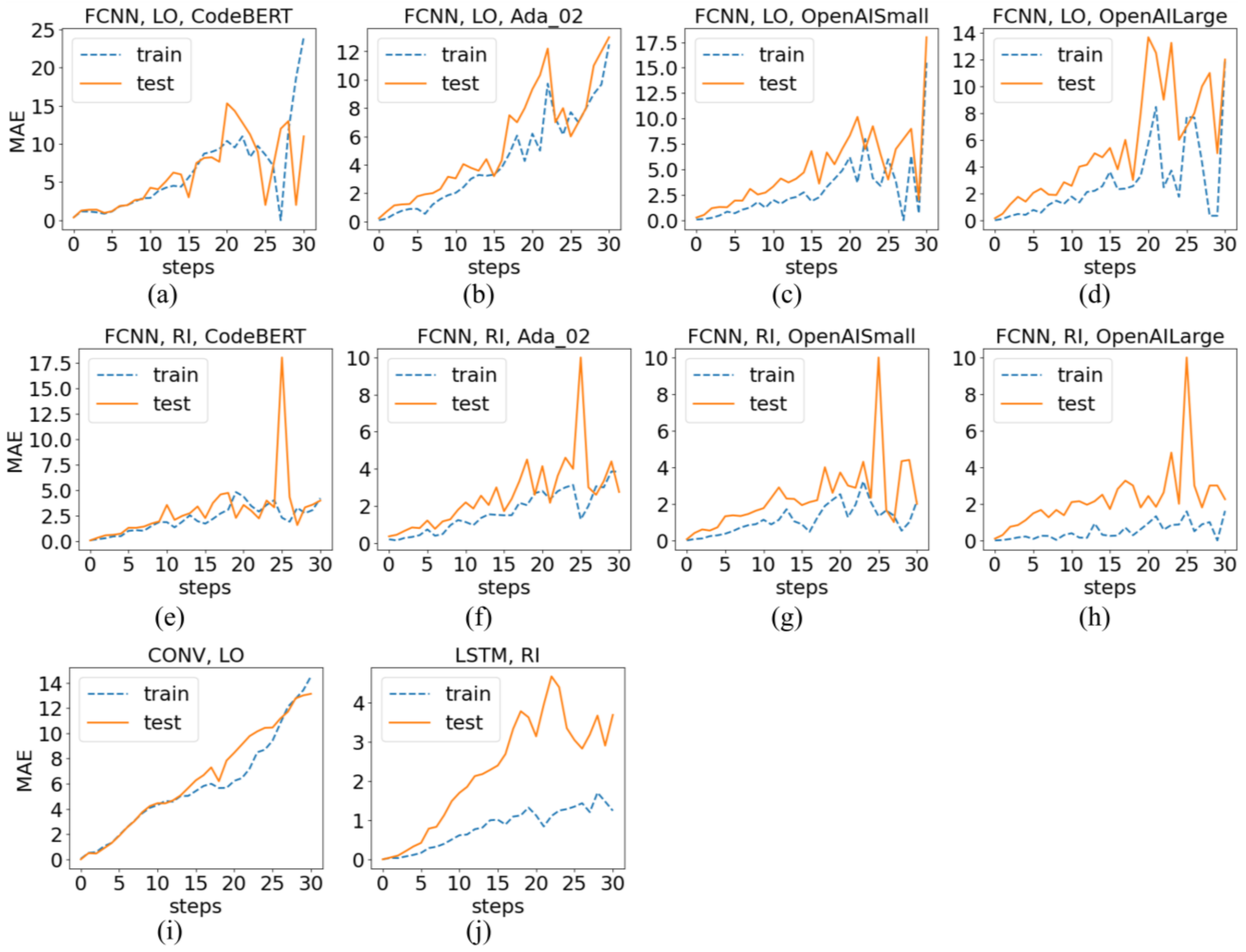

Fig. 6. MAE progression depending on the number of reduction steps for models trained on: (a) CodeBERT to LO steps; (b) text-embedding-ada-002 to LO steps; (c) text-embedding-3-small to LO steps; (d) text-embedding-3-large to LO steps; (e) CodeBERT to RI steps; (f) text-embedding-ada-002 to RI steps; (g) text-embedding-3-small to RI steps; (h) text-embedding-3-large to RI steps; (i) simplified terms to LO; (j) simplified terms to RI

general LLMs of OpenAI outperform the CodeBERT model for specific code-related tasks. This can be explained by the fact that OpenAI is more capable of processing logical information because of its enormous number of weights and broad data experience.

The limitation of this study is the use of the CodeBert model, which was initially trained in other programming languages (Go, Java, Python, and others), which can lead to incorrect representation of Lambda calculus terms in embedding vectors. Also, it is possible to say the same about using OpenAI LLMs to transform terms into embeddings, which have no training data and procedures. Another limitation of this study is the use of a comparable small number of Lambda terms and their artificial nature, which might only cover some possible variants of real ones.

Conclusion

Four LLM models were used to generate embeddings from the text representations of Lambda terms. Previously, artificially generated data of 4k Lambda terms were used. Generating embeddings was performed to analyze the possibility of using an LLM embedding model to extract term normalization features, which might help develop functional programming language compilers and interpreters.

Generated embeddings were used to create eight datasets for each considered embedding model and for the LO and RI term normalization strategies. These datasets contain the number of reduction steps for the selected strategy as a target variable. ANN models were designed and trained to solve a classification problem for predicting from 0 to 30 reduction steps. Such ANN configuration improves its performance in solving step prediction tasks.

Trained ANN models were used to collect MAE, RMSE, and step-depending MAE coefficients. These coefficients were compared with the best results achieved on the same task and dataset using simplified term representation. Results indicate improvements in the step prediction task; significant improvements were mainly achieved in predicting the number of LO steps. However, predictions of the number of RI were on the same low error rate as they were for the best result of the work [11], with slight improvement. Such results indicate that code and general LLMs can help extract information from Lambda terms and use this information to analyze strategy priority. Specially trained LLMs may lead to better data extraction. The overall conclusion is that the term feature extraction procedure using LLM is suitable and can be implemented in real Lambda code interpreters.

Future research directions. Given these limitations, further research can be conducted using the retrained LLM model for problems related to Lambda Calculus. Alternatively, increasing the number of terms in the database is possible by generating some and collecting real terms. Feature importance analysis can indicate important normalization term features and configure the Lambda Calculus interpreter to detect those features to select the appropriate strategy.

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship, or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted without any financial support.

Data Availability

This work has associated data in the data repository.

Use of Artificial Intelligence

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

References

1. Tanabe, Y., Lubis, L. A., Aotani, T., & Masuhara, H. A Functional Programming Language with Versions. *The Art, Science, and Engineering of Programming*, 2021, vol. 6, issue 1, article 5. DOI: 10.22152/programming-journal.org/2022/6/5.

2. Deineha, O., Donets, V., & Zholtkevych, G. On Randomization of Reduction Strategies for Typeless Lambda Calculus. *Communications in Computer and Information Science*, 2023, no. 1980, pp. 25–38. Available at: https://icteri.org/icteri-2023/proceedings/preview/01000021.pdf (accessed 08.03.2024).

3. Deineha, O., Donets, V., & Zholtkevych, G. Estimating Lambda-Term Reduction Complexity with Regression Methods. *International Conference "Information Technology and Interactions"*, 2023, no. 3624, pp. 147–156. Available at: https://ceur-ws.org/Vol-3624/Paper_13.pdf (accessed 08.03.2024).

4. Runciman, C., & Wakeling, D. Heap Profiling of a Lazy Functional Compiler. *Functional Programming*, 1992, pp 203–214. DOI: 10.1007/978-1-4471-3215-8_18.

5. Chlipala, A. An optimizing compiler for a purely functional web-application language. *Proceedings of the 20th ACM SIGPLAN International Conference on*

Functional Programming, 2015, pp. 10-21. DOI: 10.1145/2784731.2784741.

6. Deineha, O., Donets, V., & Zholtkevych, G. Unsupervised Data Extraction from Transformer Representation of Lambda-Terms. *Eastern European Journal of Enterprise Technology*, 2024. In press.

7. Clemens Grabmayer. Linear Depth Increase of Lambda Terms along Leftmost-Outermost Beta-Reduction. *ArXiv: Computer Science*, 2019. DOI: 10.48550/arXiv.1604.07030.

8. Asada, K., Kobayashi, N., Sin'ya, R., & Tsukada, T. Almost Every Simply Typed Lambda-Term Has a Long Beta-Reduction Sequence. *Logical Methods in Computer Science*, 2019, vol. 15, issue 1. DOI: 10.23638/LMCS-15(1:16)2019.

9. Lago, U. D., & Vanoni, G. On randomised strategies in the λ-calculus. *Theoretical Computer Science*, 2020, vol. 813, pp. 100-116. DOI: 10.1016/j.tcs.2019.09.033.

10. Xiaochu, Qi. Reduction Strategies in Lambda Term Normalization and their Effects on Heap Usage. *ArXiv: Computer Science*, 2004. Available at: https://arxiv.org/abs/cs/0405075 (accessed 08.03.2024).

11. Deineha, O., Donets, V., & Zholtkevych, G. Deep Learning Models for Estimating Number of Lambda-Term Reduction Steps. *ProfIT AI 2023: 3rd International Workshop of IT-professionals on Artificial Intelligence (ProfIT AI 2023)*, 2023, vol. 3624, pp. 147-156. Available at: https://ceur-ws.org/Vol-3641/paper12.pdf (accessed 08.03.2024).

12. Yang, Z., Ding, M., Lv, Q., Jiang, Z., He, Z., Guo, Y., Bai, J., & Tang, J. GPT Can Solve Mathematical Problems Without a Calculator. *ArXiv: Computer science, Machine Learning*, 2023. DOI: 10.48550/arXiv.2309.03241.

13. Liu, C., Lu, S., Chen, W., Jiang, D., Svyatkovskiy, A., Fu, S., Sundaresan, N., & Duan, N. Code Execution with Pre-trained Language Models. *Accepted to the Findings of ACL 2023*, 2023, pp. 4984-4999. DOI: 10.48550/arXiv.2305.05383.

14. Cummins, C., Seeker, V., Grubisic, D, Elhoushi, M., Liang, Y., Roziere, B., Gehring, J., Gloeckle, F., Hazelwood, K., Synnaeve, G., & Leather, H. Large Language Models for Compiler Optimization. *ArXiv: Computer science*, 2023. DOI: 10.48550/arXiv.2309.07062.

15. Miranda, B., Shinnar, A., Pestun, V., & Trager, B. Transformer Models for Type Inference in the Simply Typed Lambda Calculus: A Case Study in Deep Learning for Code. *Computer Science*, 2023. DOI: 10.48550/arXiv.2304.10500.

16. LeCun, Y., Bengio, Y., & Hinton, G. Deep Learning. *Nature*, 2021, no. 521, pp. 436-444. DOI: 10.1038/nature14539.

17. Li, Y., Zhang, Y., & Sun, L. MetaAgents: Simulating Interactions of Human Behaviors for LLM-based Task-oriented Coordination via Collaborative Generative Agents. *ArXiv*, 2023. DOI: 10.48550/arXiv.2310.06500.

18. Asudani, D. S., Nagwani, N. K., & Singh, P. Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial Intelligence*, 2023, vol. 56, pp. 10345-10425. DOI: 10.1007/s10462-023-10419-1.

19. Turing, A. M. Computability and λ-Definability. *The Journal of Symbolic Logic*, 1937, vol. 2, iss. 4, pp. 153-163. DOI: 10.2307/2268280.

20. Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. Attention is All you Need. *Neural Information Processing Systems*, 2017, vol. 30. DOI: 10.48550/arXiv.1706.03762.

21. Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J., & Wen, J. A Survey of Large Language Models. *ArXiv: Computer science*, 2023. DOI: 10.48550/arXiv.2303.18223.

22. Ormerod, M., Del Rincón, M. J., & Devereux, B. How is a “Kitchen Chair” like a “Farm Horse”? Exploring the Representation of Noun-Noun Compound Semantics in Transformer-based Language Models. *Computational Linguistics*, 2024, vol. 50, iss. 1, pp. 49-81. DOI: 10.1162/coli_a_00495.

23. Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. Text Classification Algorithms: A Survey. *Information*, 2019, vol. 10, iss. 4: article no. 150. DOI: 10.3390/info10040150.

24. *Compare the different Sequence models (RNN, LSTM, GRU, and Transformers). Machine Learning Resources, March 2024.* Available at: https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/ (accessed 08.03.2024).

25. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536-1547. DOI: 10.18653/v1/2020.findings-emnlp.139.

26. *New and improved embedding model. Blog, Dec. 2022.* Available at: https://openai.com/blog/new-and-improved-embedding-model (accessed 08.03.2024).

27. Nussbaum, Z., Morris, J.X., Duderstadt, B., & Mulyar, A. Nomic Embed: Training a Reproducible Long Context Text Embedder. *ArXiv: Computer science*, 2024. DOI: 10.48550/arXiv.2402.01613.

28. *New embedding models and API updates. Blog, Jan 2024.* Available at: https://openai.com/blog/new-embedding-models-and-api-updates (accessed 08.03.2024).

29. Krell, M. M., & Wehbe, B. A First Step Towards Distribution Invariant Regression Metrics. *ArXiv: Computer science,* 2020. DOI: 10.48550/arXiv.2009.05176.

Received 17.02.2024, Accepted 15.04.2024

КЕРОВАНЕ ВИЛУЧЕННЯ ДАНИХ З ТРАНСФОРМЕРНОЇ РЕПРЕЗЕНТАЦІЇ ЛЯМБДА-ТЕРМІВ

О. А. Дейнега

Об'єктом даного дослідження є процес оптимізації компіляторів, яких має ключове значення для розвитку сучасного програмного забезпечення, особливо коли мова йде про функціональні мови програмування, такі як лямбда-числення. Методи оптимізації безпосередньо впливають на швидкодію та ефективність роботи інтерпретаторів і компіляторів, визначаючи використання ресурсів та час виконання програм. Незважаючи на те, що компілятори для функціональних мов отримали менше уваги в контексті оптимізацій порівняно з об'єктно-орієнтованими мовами, унікальні виклики, що ставить перед ними лямбда-числення, потребують особливих рішень. Подолання цього розриву вимагає інноваційних підходів, таких як використання методів машинного навчання для вдосконалення стратегій оптимізації. У цьому дослідженні розглядаються особливості вилучення даних з лямбда-термів, пов'язаних із стратегіями редукції за допомогою машинного навчання. Попередні дослідження досліджували різні підходи, включаючи аналіз складності кроку скорочення та використання штучних нейронних мереж (ШНМ) аналізу послідовності зі спрощеним представленням термів. **Метою** цього дослідження є розробка методології для отримання вичерпних даних про терми, надаючи розуміння оптимальних стратегій нормалізації. Були поставлені **задачі** генерування вбудовування з лямбда-термів за допомогою великих мовних моделей (LLM), тренування моделі ШНМ для прогнозування кроків редукції та порівняння результатів зі спрощеним представленням термів. У дослідженні використовується складне поєднання алгоритмів машинного навчання та моделей глибинного навчання як **метод** аналізу та прогнозування оптимальних шляхів редукції в термах лямбда-числення для досягнення цих цілей. **Результати** показали покращення у визначенні кількості кроків скорочення за допомогою вбудовування. Результати цього дослідження мають суттєве значення для подальшого прогресу в оптимізації компілятора та інтерпретатора. Це дослідження прокладає шлях для майбутніх досліджень підвищення ефективності компілятора, демонструючи ефективність використання LLM для визначення пріоритетів стратегій нормалізації. Використання машинного навчання для оптимізації функціонального програмування відкриває можливості для стратегій динамічної оптимізації та комплексного аналізу функцій програми.

Ключові слова: лямбда-числення; функціональне програмування; оптимізація стратегії; велика мовна модель; вбудовування коду.

Дейнега Олександр Андрійович – аспірант, Харківський національний університет імені В. Н. Каразіна, Харків, Україна.

Oleksandr Deineha – PhD Student, V. N. Karazin Kharkiv National University, Kharkiv, Ukraine,
e-mail: oleksandr.deineha@karazin.ua, ORCID: 0000-0001-8024-8812, Scopus Author ID: 58865003800, https://scholar.google.com.ua/citations?user=XpjRe9gAAAAJ.