Estimating Lambda-Term Reduction Complexity with Regression Methods

Oleksandr Deineha, Volodymyr Donets and Grygoriy Zholtkevych

V.N. Karazin Kharkiv National University, 4 Svoboda Sqr, Kharkiv, 61022, Ukraine

Abstract

In this study, we dive deep into the Pure Lambda Calculus' evaluation process, specifically zooming in on term reduction steps. Commonly, these steps are seen as the same, but our research suggests otherwise - some steps are more complex than others. We investigated how specific term features play a role in computing efficiency, aiming to shed light on how different reduction strategies perform. Through testing, our Linear Regression and ANN regression models showed potential in predicting the time taken for a reduction step based on term details. Yet, when we used richer datasets to pin down complexity, our predictions didn't sharpen as much as hoped. This leads us to believe that term details, while useful, might not be the full puzzle pieces we need for spot-on strategy predictions. This research opens the door for further exploration into refining our models and unearthing other crucial factors that could shape term reduction time. Ultimately, our findings could help optimize programming tools like compilers and interpreters, steering them toward swifter operations.

Keywords [1]

pure lambda calculus, term reduction, cost of reduction, functional programming, computational complexity, estimating computational complexity, regression analysis.

1. Introduction

In Pure Lambda Calculus exists one of the fundamental problems of the evaluating program execution process [1]. The existing approach only analyzes term reduction steps as equal processes [1, 2]. The problem is that they are different, so we are trying to find a way to estimate that inequality.

This study aims to research the factors influencing time complexity in term reduction within the Pure Calculus environment. We focus on measuring different term features and their impact on computational efficiency. That approach must show that some reduction strategies are less effective in terms of computational resources but, at the same time, have smaller amounts of reduction steps.

Potentially, it may have an impact on improving programming language compilers and interpreters in a way to analyze programming code for choosing a faster execution strategy.

2. Background and related work

The idea that the impracticality of estimating term reduction strategy effectiveness by counting reduction steps as cost-equal processes has been known since early works in this area of math [1]. Also, many studies have explored various reduction strategies in functional programming but haven't focused on how time spent in term reduction varies with the topology of terms and choice of reduction strategy [1, 2]. But we must admit that there are works where authors tried to investigate methods for measuring computational reduction complexity by analyzing some reduction strategy and its memory consumption [3, 4] or measuring complexity via cost models [5]. In the work [6], the author tried to define some classes of terms by computational cost and define a type of reduction strategy. Also, other researchers proposed a weak invariant cost model for the lambda calculus, the model based on theoretical

Information Technology and Implementation (IT&I-2023), November 20-21, 2023, Kyiv, Ukraine
EMAIL: oleksandr.deineha@karazin.ua (A. 1); v.donets@karazin.ua (A. 2) g.zholtkevych@karazin.ua (A. 3)
ORCID: 0000-0001-8024-8812 (A. 1); 0000-0002-5963-9998 (A. 2); 0000-0002-7515-2143 (A. 3)

assumptions [7]. In previous articles [8, 9], we showed our Pure Lambda Calculus environment, which we developed using the Python programming language, in order to find an average optimal strategy for reducing lambda terms. The lambda environment is built on the idea of term tree representation, which was shown in many articles [2, 3, 5, 10], and we reduce all operations with terms to the idea of tree operations. So, simple operations like finding redexes, finding vertices, and finding height and width are the tree traversal operations, and operation reduction is actually a combination of the node removing and copying operations, which was shown in the paper [5]. Obviously, tree topology and its volume affect processing tree operations, but the lambda term in the tree representation is a more complicated construction, which may have many insert operations for one reduction.

We must admit that this task is practically impossible to solve due to the infinite amount of possible lambda terms [1], but it is possible to find a solution that will work in many practical applications.

3. Problem Statement

In our Pure Lambda Calculus environment we tried to find the best average strategy for reducing Lambda terms, and during experiments, we noticed that the time spent by the program on a one-step reduction of different terms differs and depends on the term topology and the selected redex. Thus, the necessity of a metric for defining the complexity of term reduction arose both before and after reduction.

After analyzing a certain amount of literature we can conclude that there is no good metric that could precisely measure the computational complexity of the term reduction strategy. There are no studies where authors tried to combine the lambda calculus environment for investigating computational measures based on the environment's computational costs. So, we need to find a way of defining this metric using knowledge about a term and redex. The metric can be used as a smart way to define the term reduction complexity and a way to define new computationally effective reduction strategies.

4. Hypothesis

We suggested that some combination of the parameters of the number of vertices, the number of redexes, the height and width of the term, as well as the depth of the redex can determine the time spent on one step of the reduction and, therefore, determine the complexity of the reduction process. In other words, we hypothesize that parameters for describing term topology can be used for accurate prediction time spent on reducing a specific redex, which shows the computational complexity of a term with specific redexes. Thereby, we can suggest two assumptions:

1. The term parameters before the reduction process with the redex data can define the computational complexity of the reduction process before performing one, or in other words, complexity prediction;
2. The second assumption requires term data before and after the reduction process with the redex data, which can define the computational complexity of the reduction process afterward, in other words, complexity determination.

5. Methodology

The central idea of Machine Learning methods is to automatically construct a complex function that best (according to the selected ML method) can approximate the real function describing the data. Because we assume time spent on one reduction step is a measure of computational complexity, we can use some method of Machine Learning for the prediction of time spent on one reduction step or solving a regression problem, in other words.

For regression analysis, we have to highlight the main regression methods:

- **Linear Regression**: one of the most important and widely used statistical techniques [11] for modeling the relationship between some feature data and target value, which is based on the equation $y = X\beta + \varepsilon$.

Advantages: it is relatively easy to train and get the final dependency expression, and this method could be relatively easily extended for use on non-linear convolution of input data. *Disadvantages*: the assumption that input and output data have a linear dependency, and sensitivity on outliers, especially on small datasets.

- **Decision Tree Regression**: the main idea of this method [12] is to split the data into subsets based on features and make predictions dependent on the average value of the sample in the leaf node. *Advantage*: interpretable, handling nonlinear and complex dependencies. *Disadvantage*: prone to overfitting, hard to configure optimal effectiveness, sensitive to noises in data.
- **Support Vector Regression** is an extension of the Support Vector Machine method for regression analysis. The method constructs a hyperplane in a high-dimensional input data space, which tends to get the largest distance to the nearest training data point for better generalization [13]. *Advantages*: versatility due to different kernel functions, effective in capturing complex relationships in high-dimensional data.
Disadvantage: sensitive to the choice of kernel.
- **K-Nearest Neighbors Regression**: uses a weighted average of the k nearest neighbors for estimating continuous variables [14].
Advantages: do not make strong assumptions about data distribution, can handle nonlinear relationships.
Disadvantages: it is difficult to obtain the objective function, sensitive to the choice of K and distance metric.
- **ANN Regression**: for an artificial neural network, we can set the output layer without activation function and it makes ANN a continuous value predictor. This is a widely used approach for obtaining continuous data without constraining the ANN model [15]. *Advantages*: ability to capture complex data relationships, high level of generalization, simple set of approaches to modify the effectiveness of the ANN.
Disadvantages: prone to overfitting, hard to tune many hyperparameters, sensitive to loss function choice.

As a metric for the estimating effectiveness of the regression model we are supposed to use default measures like [16]:

- **Mean Absolute Error (MAE)**: measures the average absolute difference between the expected values and predicted.
Advantages: easy to interpret, resistant to outliers because of equal weights to all errors. *Disadvantages*: can't define if it overestimates or underestimates, doesn't provide information about the error distribution.
- **Mean Squared Error (MSE)**: squares the difference between the expected and predicted values and takes the average of its sum.
Advantage: gives higher penalties for larger errors, which is important for use as a loss function. *Disadvantage*: sensitive to outliers and large errors can have a significant impact on the results. The squared units make interpretation less intuitive.
- **Root Mean Squared Error (RMSE)**: which is the square root of MSE. *Advantages*: provide a more interpretable measure compared to MSE, weighted penalties for larger error value.
Disadvantage: sensitive to outliers like MSE.
- **Mean Absolute Percentage Error (MAPE)**: measures the percentage of the difference between the expected and predicted values, averaged over all observation values. *Advantages*: it is easy to interpret and compare across different datasets, a percentage-based measure of accuracy.
Disadvantage: undefined when actual values are zero or close to zero, can be sensitive to outliers.

Over a list of the main regression analysis methods and metrics for measuring regression accuracy, we can use Linear Regression with complex data preprocessing and ANN Regression as methods that can help us to formulate dependencies between term characteristics and time spent on reduction steps. Other methods could be indicators of the best accuracy. Also, for ANN Regression, we use the MSE measure as a loss function and RMSE and MAE as more interpretable metrics for the cross-comparing model effectiveness.

6. Experiments and data collection

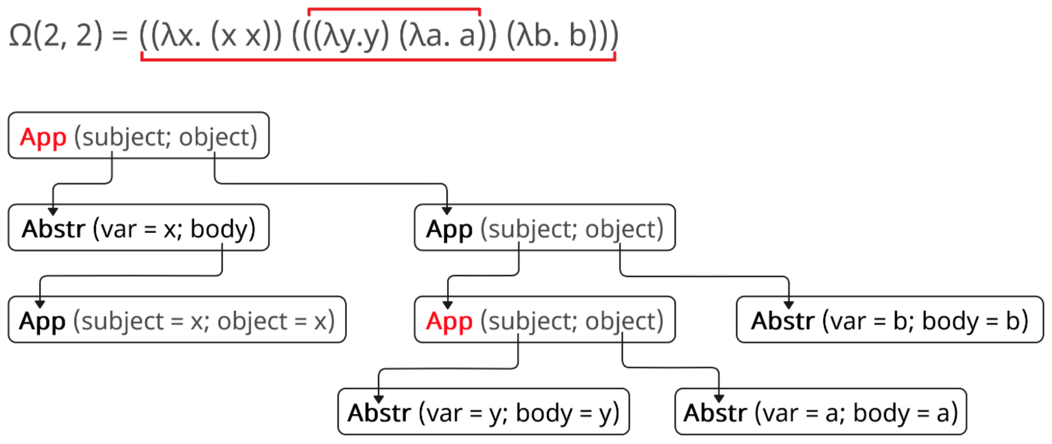
6.1. Data generation

In the Pure Lambda Calculus environment, we have a possibility to artificially generate terms. For the generating procedure, we can set min, and max counts vertices and redexes, after generating we have a filtering procedure that filters out terms with too long reduction trace with the selected strategy (by default, it is the leftmost outermost strategy) or with too big memory consumptions. The term generating procedure was inspired by the article [10]. For experiments, we generated 220 terms, which we normalized with the leftmost outermost and the rightmost innermost strategies, which gave us 3931 records with time in nanoseconds spent on each reduction. We also must admit that in this scenario time spent on reduction consists of time spent on searching for appropriate redex by selected strategy plus time spent on doing reduction. We use a Unix-based operating system for collecting time data because it gives a more clear representation of time spent on processes than other operating systems. That approach should give as accurate computational resource consumption as possible.

6.2. Data points and metrics

In Fig. 1, we can see that the term could be easily represented as a tree, so features that we collect about the term are actually parameters that describe a tree:

1. Height – the longest sequence from the tree root to the term tree (for the term in Fig. 1 the height value equals 4).
2. Width – count leaves of the term tree (for the example equals 4).
3. Redexes – count redexes in a term (in Fig. 1, you can find 2 redexes, marked as red Apps in the tree representation, and underlined in the formula representation).
4. Vertices – count vertices in the term tree, in terms of lambda calculus, are count all applications, abstractions, and atom variables (in Fig. 1, it equals 12).
5. Redex depth – length of sequence from the tree root to the selected redex application node (for the first redex, which is the root node in the tree, it equals 1, and for the second redex it equals 3).
6. Step time – time spent on finding appropriate redex by selected strategy plus time spent on reduction step by selected redex in nanoseconds.


Ω(2, 2) = ((λx. (x x)) (((λy.y) (λa. a)) (λb. b)))
App (subject; object)
Abstr (var = x; body)
App (subject; object)
App (subject = x; object = x)
App (subject; object)
Abstr (var = b; body = b)
Abstr (var = y; body = y)
Abstr (var = a; body = a)

b **Figure 1**: $\Omega(2,2)$ term with marked redexes and its tree representation

Also, we collect parameters that describe the term after the reduction procedure, which are the same except for redex depth, which isn't required now. We also got differences between parameter values before and after the reduction procedure.

7. Data analysis

7.1. Correlation matrices

In the domain of regression analysis, the quest for precision and accuracy demands a meticulous understanding of the relationship between variables in input data and the relationship between inputs and outputs, which will help us decide if further research makes sense. In the broadest sense, correlation may indicate any association type, but in statistical analysis, it usually refers to linear relationships, and this is an important drawback to be aware of.

Therefore, it became necessary to analyze the obtained data for correlations. Therefore, correlation matrices were obtained, you can find them in Fig, 2, it was shown that the term features data and the redex data have the highest correlation with the reduction time (0.56 - 0.77, shown in Fig. 2.a), the term features data after reduction have a slightly smaller correlation with reduction time (0.48 – 0.68, shown in Fig. 2.b) and given features difference before and after reduction have the smallest correlation (-0.033 – 0.13, shown in Fig. 2.c).

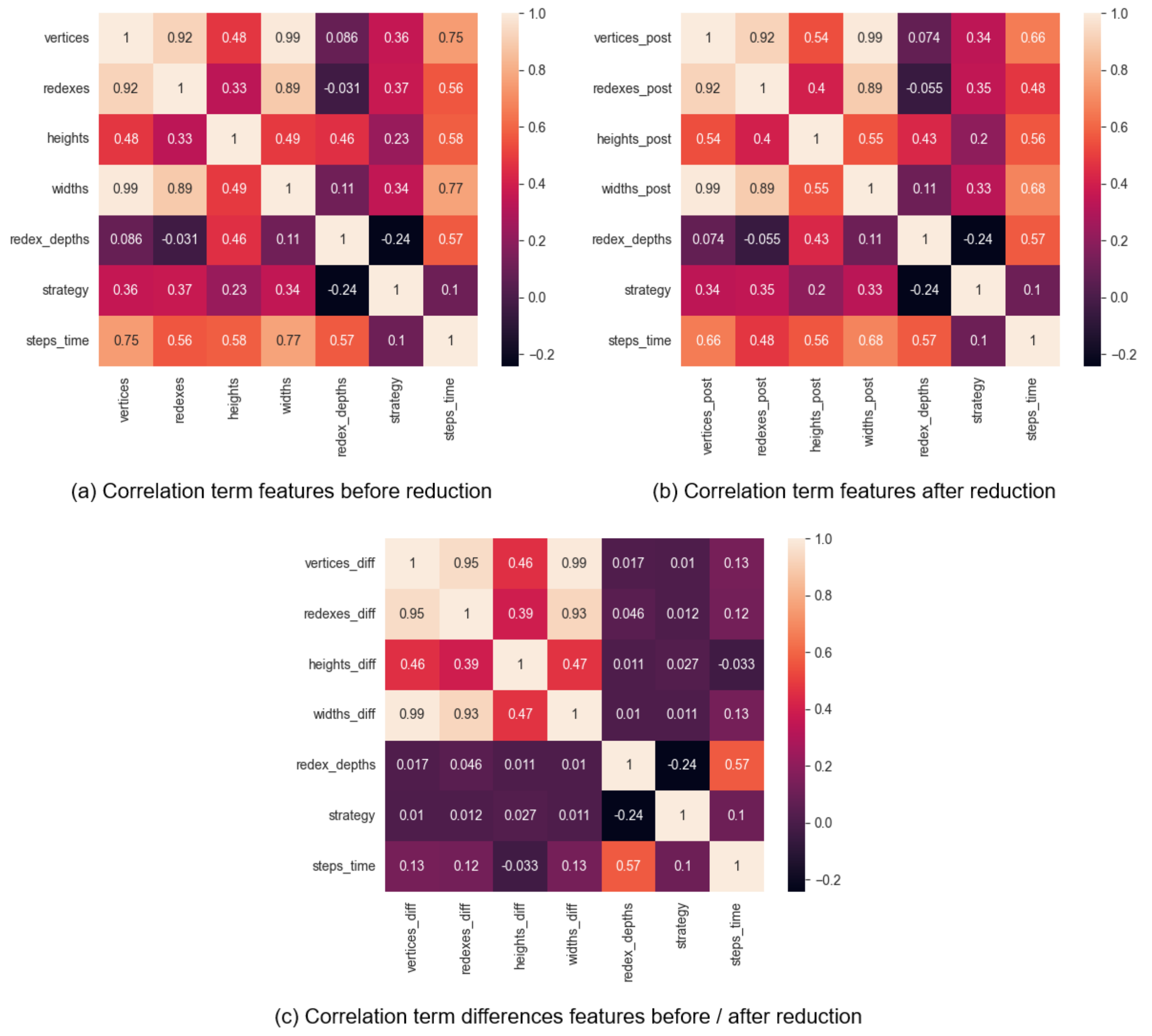


Figure 2: Correlation matrices for term parameters before and after reduction, and its differences: (a) correlation term features before reduction, (b) correlation term features after reduction, (c) correlation term differences features before and after reduction

We must admit that parameters like vertices, redexes, heights, widths, their post-reduction variant, and the difference variant have high correlation values, close to 1, which simply indicates that these parameters are related due to the term tree topology. Also, we should emphasize that the selected reduction strategy doesn't have correlation to the reduction step time and other term parameters. According to the results of the correlation analysis, we decided that solving the regression problem for these data and the target value made sense.

7.2. Assumptions

We previously defined two assumptions. Based on these assumptions, we can now define three datasets for training and testing:

1. For the complexity prediction, we can use the term data before the reduction process and the redex data. Input data: *widths*, *heights*, *vertices*, *redexes*, and *redex_depths*. Target data: *steps_time*.

2. For the complexity determination, we can use the term data before and after the reduction process and the redex data. Input data: *widths*, *heights*, *vertices*, *redexes*, *widths_post*, *heights_post*, *vertices_post*, *redexes_post*, and *redex_depths*. Target data: *steps_time*.

3. For the complexity determination, we can use the term data before and after the reduction process, the step difference term data, and the redex data. Input data: *widths*, *heights*, *vertices*, *redexes*, *widths_post*, *heights_post*, *vertices_post*, *redexes_post*, *widths_diff*, *heights_diff*, *vertices_diff*, *redexes_diff*, and *redex_depths*. Target data: *steps_time*.

7.3. Data preprocessing

The typical problem for any Machine Learning method is data distribution inputs and outputs: many methods require close distributions like (0; 1), (-1; 1), etc., which guarantee as fast method convergence as possible. Hence, we decided to analyze the term features distributions and the target data distribution. The results of the data distribution analysis are shown in Fig. 3, and we can see two problems: the first problem with collected data is too wide distributions (unscaled in other words) and the second problem for some features is non-normal data distribution, which might impair the learning process. A non-normal distribution of data is characteristic of width (Fig. 3.a and Fig. 3.e), vertices (Fig. 3.c and Fig. 3.g), redexes (Fig. 3.d and Fig. 3.h) before and after the reduction step. All parameters, which are detailed in Figure 3, are presented in their unscaled form.

To increase model regression performance we decided to take a logarithm of target value (step time). For all input fields, we apply Yeo-Jonson power transformation [17], because it can take zero values in data, process data logarithmically, normalize data distribution, and automatically scale data. After applying the Yeo-Jonson power transformation for input data and taking the logarithm of the target value, we got new data distributions which are shown in Fig. 4. It’s easy to see that new data distributions for widths (Fig. 4.a and Fig. 4.e), vertices (Fig. 4.c and Fig. 4.g), and redexes (Fig. 4.d and Fig. 4.h) before and after reduction step, also, the target variable step time (Fig. 4.n) became close to normal distribution. Other features: heights (Fig. 4.b and Fig. 4.f), and all differences (Fig. 4.i, Fig. 4.j, Fig. 4.k, and Fig. 4.l) did not change their distribution law. The transformation ensures setting compact value distribution for all features and the target variable. In total, all these changes will improve model quality and speed up regression model convergence.

Before testing regression methods, we should consider that usual practice in Machine Learning is dividing a dataset into tree sets for training, validation and testing. These sets are required for model training episodes, tuning model hyperparameters, and final model testing in accordance, which allows us to reveal the real quality of learning on previously unseen data. Due to relatively small amounts of data and small models, we decided to use the test set as the validation. So, we divided our term records in all datasets into two subsets namely training and testing ones in proportion to 70% / 30% in accordance, so we got 2782 samples in the train set and 1149 samples in the test set. Also, we should highlight that terms data in three datasets are identical, and train-test splitting happens in the same order. So, model performance depends not on the random data splitting but mostly on itself.

8. Regression models

8.1. Standard approaches

Further, standard approaches for solving regression problems were tested on the 1-st dataset (complexity prediction problem). Results are represented in Table 1, where you can find results for the following methods: 1. Linear Regression, 2. Decision Tree Regression, 3. Support Vector Regression, 4. K-Nearest Neighbors Regression and 5. ANN Regression. Methods 2 and 4 showed the lowest error values (RMSE = 0.0497 – 0.2105) on the train data, but the error value on the test data was relatively high (RMSE = 0.30 – 0.4), indicating the inability of these models to generalize the data. Methods 1, 3, and 5 showed a higher error rate during training (RMSE = 0.27 – 0.29), but on test data, the error was identical (RMSE = 0.28), which indicates the sufficient ability of the algorithms to generalize the data. We decided to continue testing methods 1 and 5 due to the high generalizability. Linear Regression was also practically proven that the linear model is quite sufficient for estimating the reduction time, and it’s easy to get the final complexity estimation equation. ANNs are effective in terms of modifiability and their ability to approximate complex functions within data.

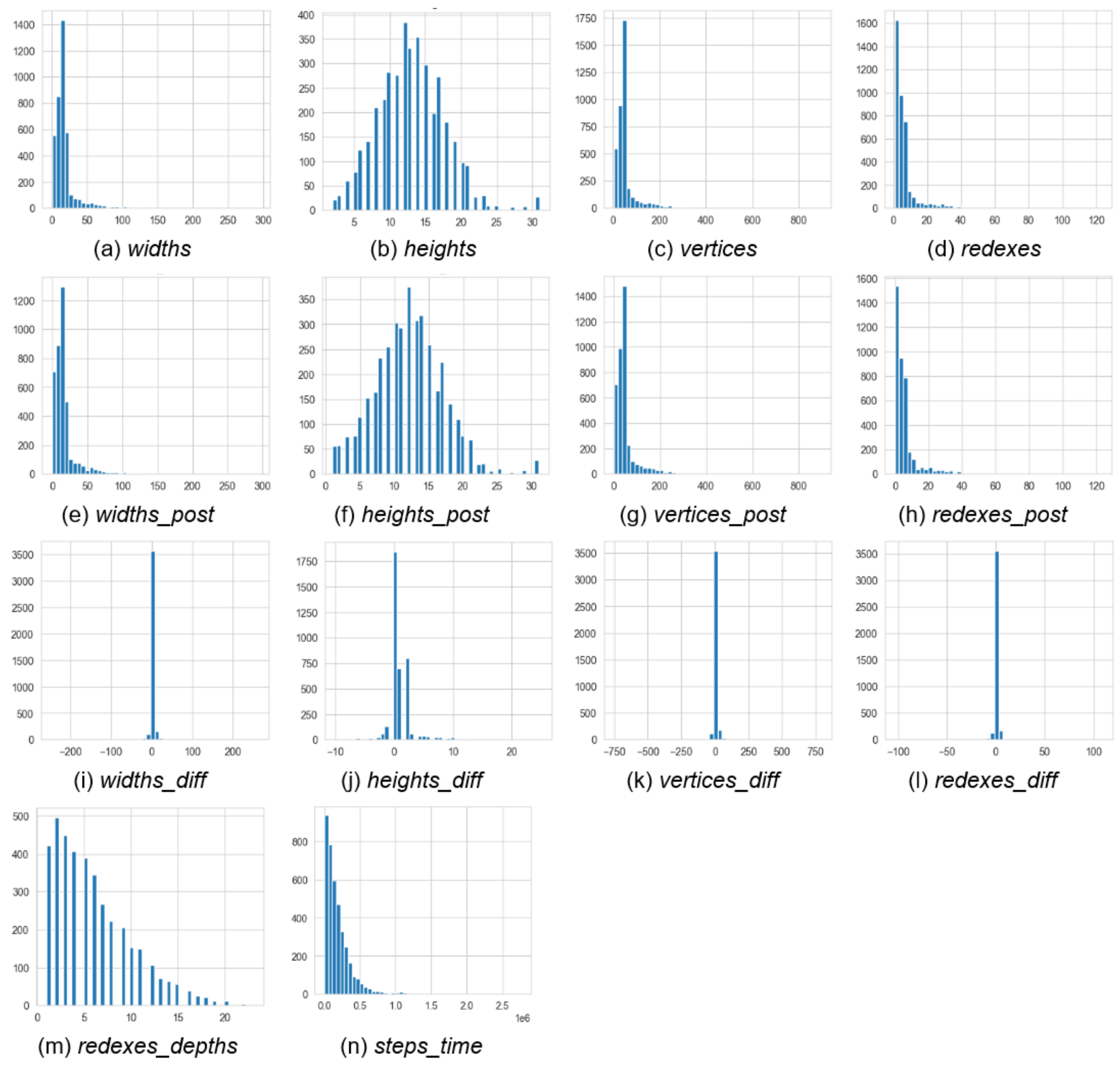


Figure 3: Data distributions for all collected term features: (a) widths, (b) heights, (c) vertices, and (d) redexes before reduction; (e) widths, (f) heights, (g) vertices, and (h) redexes after reduction; (i) widths, (j) heights, (k) vertices, and (l) redexes after and before reduction features difference; (m) redexes depth as redexes feature; and target variable (n) steps time

8.2. Model performance

As you can see in Table 1, the Linear Regression and the ANN Regression models showed high generalizability with RMSE of 0.28-0.29 on the test set with the 1st dataset. So we decided to test the second assumption about complexity definition (on the 2nd and the 3rd datasets), where we achieved RMSE of 0.27-0.29, which didn't seem as much improvement withholding more data as expected.

Also, for the 1st dataset, you can see in Fig. 5, drawings for ANN predictions and expected values. These plots are achieved by plotting a dot in the space of predicted and expected values, where closer values to the diagonal (marked as a gray line) mean a better model approximation. Here, dots over the diagonal indicate overestimation, and dots under the diagonal indicate underestimation. Also, these plots for the 2nd and the 3rd datasets are approximately the same as for ANN Regression and for Linear Regression, which means that even more complex models cannot get better results. In Table 2 we can find other metrics like MSE, MAE, and MAPE for estimating efficiency of Linear Regression and ANN Regression models on the 1st, 2nd, and 3rd datasets. We conclude that using other metrics did not give many benefits in distinguishing the quality of models, but some metrics like MAE give an average error value, and MAPE gives an average percent of error, which might help in understanding plots in Fig. 5.

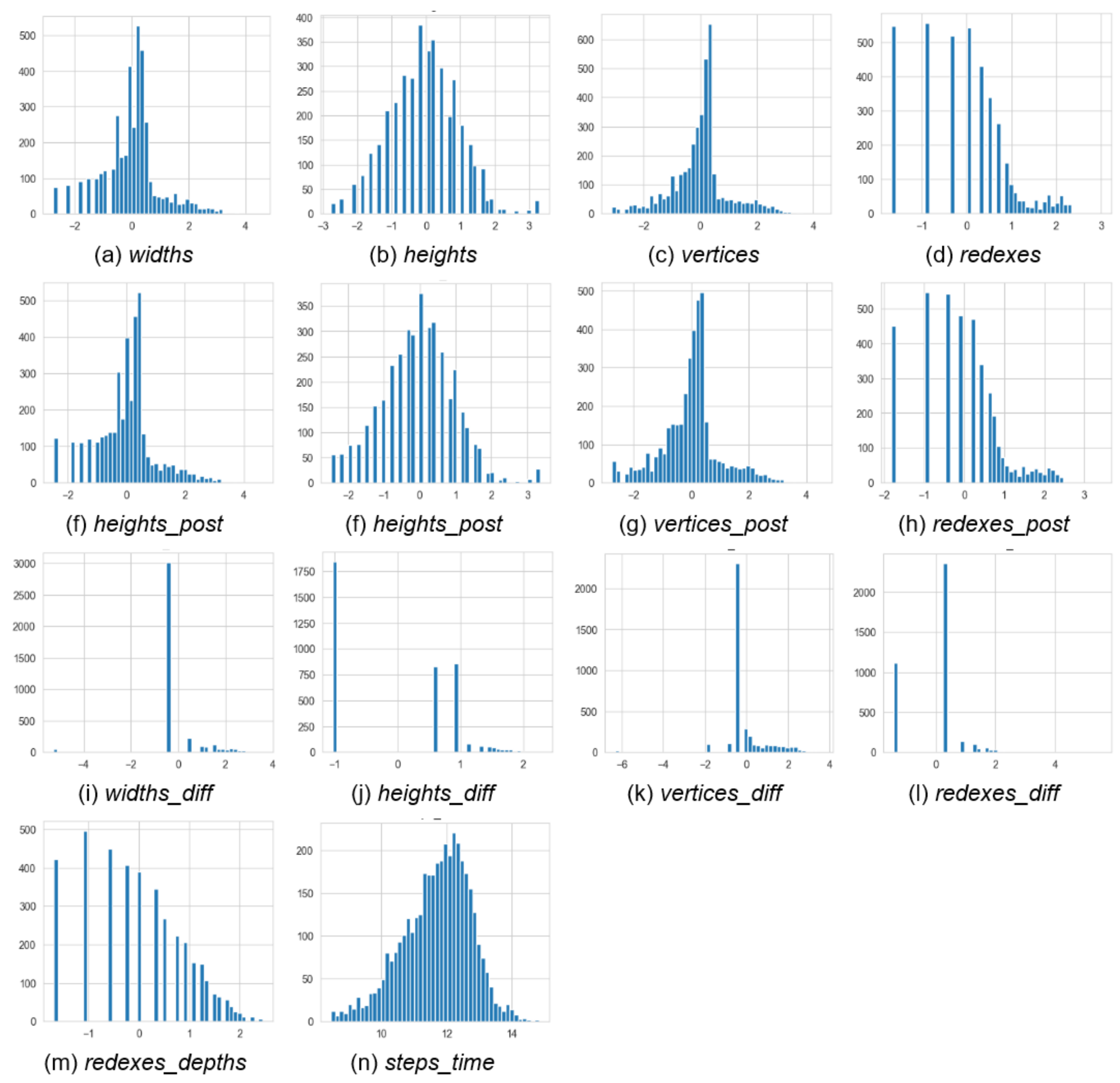


Figure 4: Data distributions after applying Yeo-Jonson normalization for all collected term features: (a) widths, (b) heights, (c) vertices, and (d) redexes before reduction; (e) widths, (f) heights, (g) vertices, and (h) redexes after reduction; (i) widths, (j) heights, (k) vertices, and (l) redexes after and before reduction difference; (m) redexes depth as redexes feature; and taking logarithm of target variable: (n) steps time

Table 1
Best values of Root Mean Squared Errors, achieved for different models, studied on three formulated datasets

Regression Method	1st dataset		2nd dataset		3rd dataset	
	train	test	train	test	train	test
1. Linear Regression	0.2953	0.2818	0.2989	0.2938	0.2968	0.2873
2. Decision Tree Regression	0.0497	0.4052	-	-	-	-
3. Support Vector Regression	0.2744	0.2892	-	-	-	-
4. K-Nearest Neighbors Regression	0.2105	0.3217	-	-	-	-
5. ANN Regression	0.2905	0.2845	0.2805	0.2858	0.2788	0.2759

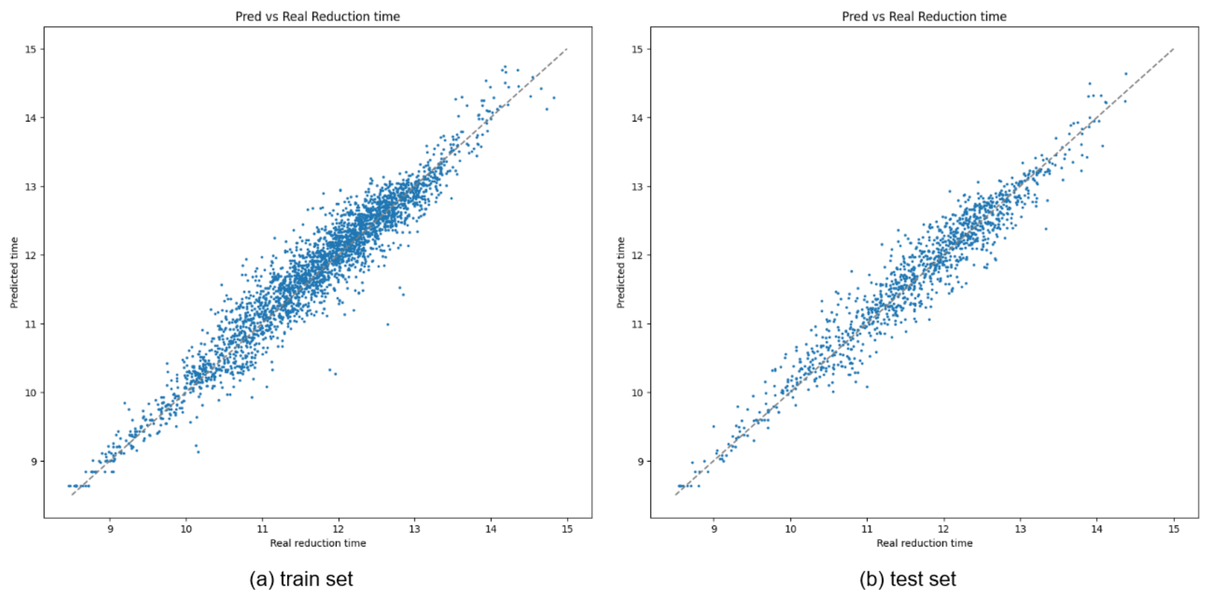


Figure 5: Dots plot real values and prediction for estimating ANN model efficiency on the 1st dataset: (a) achieved for the train set, (b) achieved for the test set

Table 2
MSE, MAE and MAPE for Linear Regression models and ANN Regression models for the 1st, 2nd, and 3rd datasets separated by train and test sets

Regression Method	1st dataset		2nd dataset		3rd dataset	
	train	test	train	test	train	test
1. Lin. Regr. (MSE)	0.0871	0.0788	0.0879	0.0895	0.0837	0.0923
5. ANN Regr. (MSE)	0.0858	0.0766	0.0788	0.0814	0.0766	0.0786
1. Lin. Regr. (MAE)	0.2276	0.2196	0.2284	0.2326	0.2244	0.2370
5. ANN Regr. (MAE)	0.22	0.2116	0.2158	0.2206	0.2141	0.2146
1. Lin. Regr. (MAPE)	1.958	1.894	1.9641	2.013	1.93	2.047
5. ANN Regr. (MAPE)	1.898	1.827	1.872	1.925	1.846	1.856

9. Discussion

Performance testing of our model indicates that both the Linear Regression and ANN regression models effectively predict the time spent on a reduction step based on term parameters. So we can conclude that the first assumption about complexity prediction is relevant. However, the second assumption about complexity determination with more detailed datasets didn't show much effectiveness improvements despite available data about the term state after the reduction procedure and differences in these states. In particular, we can admit that the effect on the ANN model on which we can increase model volume, but without much impact on MSE. So, we can conclude that term parameters have relations to term reduction complexity, but these parameters aren't sufficient, which is a weak sign, that might help with estimating the best reduction strategy but without prominent accuracy.

10. Conclusion

We hypothesized that parameters that describe term topology can be used for accurate prediction time spent on reducing a specific redex, which shows the computational complexity of a term with specific redexes. For this, we have proposed two assumptions: complexity prediction and complexity determination, and we formulated three datasets on data obtained by normalizing artificially generated terms. The first conclusion: we showed on two datasets for complexity determination, we didn't achieve much improvement in RMSE value even with increasing model capacity, which indicates that state term after the reduction process and difference data don't offer much information about the reduction process. The second conclusion: we achieved a stable level of RMSE on train and test data for the 1-st dataset for complexity prediction, which means that with the Linear Regression model or with the ANN

Regression model we can get a weak term reduction estimator, which doesn’t require big computational resources, but doesn’t provide quite accurate estimation.

For further research, we could focus on enhancing the regression models or investigating new parameters that might affect term reduction time. For investigating new parameters, we could use parameters that describe redex in more detail, like count bound variables in the object body (which indicates count inserts in the term tree), parameters for redex subject, etc. We should also consider improving the accuracy of our time measurements. Our experiments were conducted on a Unix-based operating system using the Python interpreter, which does not guarantee precise time accuracy.

11. References

[1] Mariangiola Dezani-Ciancaglini, Simona Ronchi Della Rocca, and Lorenza Saitta. Complexity of lambda-term reductions. RAIRO Theor. Informatics Appl. 13 (1979): 257-287. doi:10.1051/ita/1979130302571.

[2] Kazuyuki Asada, Naoki Kobayashi, Ryoma Sin'ya, and Takeshi Tsukada. Almost Every Simply Typed Lambda-Term Has a Long Beta-Reduction Sequence. Logical Methods in Computer Science, February 2019, Volume 15. doi:10.23638/LMCS-15(1:16)2019.

[3] Clemens Grabmayer. Linear Depth Increase of Lambda Terms along Leftmost-Outermost Beta-Reduction. November 2019. URL: https://doi.org/10.48550/arXiv.1604.07030.

[4] Xiaochu Qi. Reduction Strategies in Lambda Term Normalization and their Effects on Heap Usage. 2004. URL: https://arxiv.org/abs/cs/0405075.

[5] Ugo Dal Lago, and Simone Martini. On Constructor Rewrite Systems and the Lambda Calculus. Logical Methods in Computer Science, August 2012, Volume 8. URL: https://doi.org/10.2168/LMCS-8(3:12)2012.

[6] Maciej Bendkowski. Towards the Average-Case Analysis of Substitution Resolution in Lambda-Calculus. International Conference on Formal Structures for Computation and Deduction (2018). URL: https://arxiv.org/pdf/1812.04452.pdf.

[7] Ugo Dal Lago and Simone Martini. An Invariant Cost Model for the Lambda Calculus. 2005. URL: https://arxiv.org/pdf/cs/0511045.pdf.

[8] Vladyslav Shramenko, Victoriya Kuznietcova, and Grygoriy Zholtkevych. Studying Mixed Normalization Strategies of Lambda Terms. Proceedings of the 2nd International Workshop of IT-professionals on Artificial Intelligence, Vol. 3348 (2022): 57-68. URL: https://ceur-ws.org/Vol-3348/paper5.pdf.

[9] Oleksandr Deineha, Volodymyr Donets, and Grygoriy Zholtkevych. On Randomization of Reduction Strategies for Typeless Lambda Calculus. Proceedings of conference ICTERY 2023, in press. URL: https://icteri.org/icteri-2023/proceedings/preview/01000021.pdf.

[10] Paul Tarau. On lambda-term skeletons , with applications to all-term and random-term generation of simply-typed closed lambda terms. 2017. URL: http://www.cse.unt.edu/~tarau/research/2017/repel.pdf.

[11] D. Maulud, and A. M. Abdulazeez. A Review on Linear Regression Comprehensive in Machine Learning. JASTT, vol. 1, no. 4, pp. 140-147, Dec. 2020. doi:10.38094/jastt1457.

[12] Chang Liu, Zhenhua Hu, Yan Li, Shaojun Liu. Forecasting copper prices by decision tree learning. Resources Policy, Volume 52, 2017, Pages 427-434. ISSN 0301-4207. URL: https://doi.org/10.1016/j.resourpol.2017.05.007.

[13] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. Statistics and Computing 14, 199–222 (2004). URL: https://doi.org/10.1023/B:STCO.0000035301.49549.88.

[14] T. Cover, and P. Hart. Nearest neighbor pattern classification. In IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, January 1967. doi:10.1109/TIT.1967.1053964.

[15] Malte Jahn. Artificial neural network regression models: Predicting GDP growth. HWWI Research Paper No. 185, 2018. URL: https://www.econstor.eu/handle/10419/182108.

[16] M. M. Krell, and Bilal Wehbe. A First Step Towards Distribution Invariant Regression Metrics. 2020. URL: https://doi.org/10.48550/arXiv.2009.05176.

[17] E. Tieppo, J. P. Barddal, and J. C. Nievola. Improving Data Stream Classification using Incremental Yeo-Johnson Power Transformation. 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Prague, Czech Republic, 2022, pp. 3286-3292, doi:10.1109/SMC53654.2022.9945302.