

Deep Learning Models for Estimating Number of Lambda-Term Reduction Steps

Oleksandr Deineha, Volodymyr Donets and Grygoriy Zholtkevych

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

Abstract

This research delves into the utilization of execution strategy-dependent program state information for the enhancement of compilers and interpreters across functional and object-oriented programming languages. Focusing on Lambda Calculus as a versatile representation, we employ Machine Learning models to ascertain reduction step counts for both the Leftmost Outermost and Rightmost Innermost strategies. Leveraging Convolution-based, LSTM-based, and Transformer-based models, we estimate reduction steps within a defined range. Employing a one-hot-encoded simplified term representation streamlines input dimensions, though at the expense of variable information. Our experiments reveal that this simplified representation suffices for estimating reduction counts within a significant range. Remarkably, predictions for the Rightmost Innermost strategy outperform those for the Leftmost Innermost strategy by nearly 25%. In conclusion, employing trained models to determine reduction step counts for specific terms shows great promise in automating the search for optimal reduction strategies tailored to individual terms. Our findings underscore the viability of simplified term representations for predicting reduction steps within a specified strategy, albeit with considerations for depth elimination. Experiments highlight the challenge of handling lengthy sequences and complex terms in models.

Keywords ¹

Pure Lambda Calculus, Deep Learning, Large Language Models, Convolutional NN, LSTM, Transformer, computational optimization.

1. Introduction

Information about program execution state depending on execution strategy helps in choosing reasonable, effective, in terms of computational resource amount, and robust execution strategies that would be implemented in improvements for compilers and interpreters, not only for functional but also for object-oriented programming languages.

This work is a part of our series of works, where we focus on researching average optimal reduction procedures for Lambda Calculus terms. We used Lambda Calculus as a simple representation for any programming language, which in fact could implement any of them [1]. In this study, we consider using Machine Learning models to determine the number of reduction steps for the Leftmost Outermost (LO) and Rightmost Innermost (RI) strategies. We chose three Machine Learning models (Convolution-based ANN, LSTM-based ANN, and Transformer-based ANN) for the estimation count reduction steps from 0 to 30. As input, we use one-hot-encoded simplified term representation, which allows us to significantly shrink input dimensions into the model while saving terms tree structure, but we lose information about term variables. Losing this type of information introduces some sort of uncertainty to Lambda terms, but we are going to determine how much impact it has.


Results of our experiments show that simplified term representation is enough for estimating the number of reductions with sufficient diapason. Interestingly, predicting count reduction steps

1ProfIT AI 2023: 3rd International Workshop of IT-professionals on Artificial Intelligence (ProfIT AI 2023), November 20–22, 2023, Waterloo, Canada

✉ oleksandr.deineha@karazin.ua (O. Deineha); v.donets@karazin.ua (V. Donets); g.zholtkevych@karazin.ua (G. Zholkevych)

ORCID 0000-0001-8024-8812 (O. Deineha); 0000-0002-5963-9998 (V. Donets); 0000-0002-7515-2143 (G. Zholkevych)

© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

for the RI strategy was almost 25% more precise than predicting count reduction steps for the LO strategy.

In addition, we must admit the computational contradiction: is it more effective to just keep running Lambda term reduction in parallel for different strategies, and choose more preferable in some defined sense or convert a Lambda term to the simplified representation and feed to ANNs, and choose the most preferable strategy.

2. Background and related work

In previous works, we showed our Pure Lambda Calculus environment [2, 3], which we used for the Lambda Calculus normalization process research. In [3], we focused on investing in a new approach for estimating the reduction step complexity by estimating the computational resources required for this step using the basic parameters that characterize the term tree and its redex.

The investigation of Lambda Calculus term reduction has been the main topic of many studies. Some studies have analyzed the influence of reduction strategies on count term reductions and detailed analysis of some standard strategies [4, 5, 6]. In the research [7], the author considers the computational aspects of reduction strategies. However, none of them try to extract some term features that signal the preference for some strategies over others.

In addition, promising results were achieved in the case of solving mathematical problems [8], compile optimization, code execution [9], and code compilation optimization [10] with Large Language Models based on Transformer models. Type Inference in the Simply Typed Lambda Calculus with Transformer model was considered [11], although this article was about predicting a term type it still required a deep term understanding by the model, which is a key point in our work. The authors of [11] considered full-term representation via token sequences by encoding special symbols for Applications, Abstractions, and 32 possible term variables.

3. Problem Statement

In modern compilers, interpreters implement specific methods for optimizing the code execution process [10]. The problem with this approach is choosing the right optimization method for an infinite number of program variants. We investigated the automatization of the process by choosing the appropriate optimizer.

As a popular method of automatization, we stopped on Artificial Neural Networks (ANN) methods, which are de facto automatic statistical collectors [12]. The program is a sequence of keywords, operators, and variables, so we consider state-of-art methods [13, 14, 15] for processing sequences: Convolutional Neural networks, Long-Short Term Memory (LSTM) networks, and Transformer networks.

Another problem with testing exactly on any modern programming language is the large number of possible keywords, and as a result, it is difficult to collect or generate training data. Thus, the obvious solution is to use a programming language with simple syntax, such as Lambda Calculus, where only one operator Application exists; however, Lambda Calculus is Turing sufficient [1], which means that it can imitate any computational process available for other programming languages. In Lambda Terms, a choosing rule of a specific application called redex defines a reduction strategy (or, in other words, execution sequence), and as shown in articles [2, 4, 5], can have an impact on count reduction steps.

The main advantages of synthetic training data are that it is cheaper to obtain, covers as many data combinations as possible, and does not require special tuning for exact feature extraction. This allows the training of robust and widely tasked models. However, it might not cover real task combinations and introduce unnecessary complexity into models.

Combining these, we aim to solve the following problems:

1. Configure the hyperparameters of Machine Learning models and train them for as low as possible error rates for prediction count reduction steps for a specific strategy.

2. Generate a sufficient amount of Lambda terms for training robust models, which can operate with real-term data with the same error rate.

4. Hypothesis

We deal with Lambda terms data, and the main issue with them is an infinite number of variables, which can be presented in Lambda terms. A possible solution is to limit the maximum number of variables, such as 64, 128, and 512, to save variable information. On the other hand, using variable information significantly increases the model input size, and as a result, increases the total weight requirements for ANNs, it also leads to the necessity to carefully configure training data in order of proper model coverage.

Considering all this we decided to simplify term representation for models in the following way: mark all Abstraction and Atom variables as 'x', so we save a term as a tree without any additional information about inner relationships, losing information about detailed redexes representation.

Therefore, we hypothesize that simplified term representation is sufficient for estimating count reduction steps for the LO and RI strategies using modern Machine Learning models for sequence processing.

5. Methodology

We assume that a simplified term representation is sufficient to estimate the count reduction steps. Our problem here is to show the relationship between the model's inputs and outputs and real values, and it has three solutions:

1. Solving this as a regression problem, we cannot make the model as precise as possible, and we cannot collect terms with too large count reduction steps to save the normal distribution of expected and output data.
2. Solving this as a classification problem allows us to make the model more precise. However, we should set a strict limit on count reduction steps for selected strategies, and it also imposes restrictions on the nature of the terms entering the training data.
3. Solving this as a bin problem allows us to combine a regression problem and a classification problem and obtain some bins for defined term distributions such as 0-4, 5-10, up to infinity. This solution does not guarantee a precise solution, but it gives us the possibility to analyze any kind of term.

Taking into account our findings (as illustrated in Figure 5) that the majority of the generated terms are reduced within 0 to 30 steps, we opted to approach the problem as a classification task. In addition, we must admit that the accuracy metric is not suitable for the evaluation number of reduction steps prediction quality, because low accuracy does not mean that the count step error should be high; therefore, we are also considering the following regression metrics [16]: Mean Absolute Error (MAE), which measures the average absolute difference between the actual and predicted reduction steps, and Root Mean Squared Error (RMSE), which is the root of the squared difference between the actual and predicted reduction steps. Building on the hypothesis that terms requiring more reduction steps would exhibit a greater error value, we analyzed the MAE concerning the actual count of reduction steps.

6. Models for sequence processing

To process sequential data, we can consider the following methods:

1. Hidden Markov models (HMMs) are a type of probabilistic model that can be used to model sequential data [17]. HMMs are often used for tasks, such as speech recognition and machine translation.
Pros: relatively simple to train and deploy; able to learn long-range dependencies in sequential data.

- Cons: difficult to design for specific tasks; computationally expensive on large datasets.
2. Support vector machines (SVMs): SVMs are a type of machine learning algorithm that can be used for various tasks, including classification and regression [18]. SVMs with Sequence Kernels are suitable for sequence-processing tasks, such as text classification and sentiment analysis.
Pros: effective for the variance of sequence processing tasks, including text classification and efficient training and deployment.
Cons: difficult to tune for specific tasks; prone to overfitting; computationally expensive to train.
 3. Rule-based systems are a type of expert systems that use a set of rules to make decisions [19]. Rule-based systems can be used for a variety of tasks, including sequence processing tasks such as text filtering and machine translation.
Pros: easy to interpret and can be very accurate for tasks where the rules can be determined.
Cons: difficult and time-consuming to develop and maintain, and scale to large datasets, and difficult to develop systems with good generalization capabilities.
 4. Artificial neural networks (ANNs) are powerful tools for processing sequential data such as text, audio, and video. They show a state-of-the-art level of productivity for many sequence-processing tasks [13, 14, 15]. ANNs can learn to extract patterns from sequential data and use these patterns to make predictions or generate new sequences.
Pros: very flexible to variety sequences; can learn long-term dependencies in data; able to generalize unseen data.
Cons: computationally expensive to train; difficult to interpret; hard to tune hyperparameters for proper generalizing and prevent overfitting.

Because many ANN models show state-of-the-art productivity for many sequence processing tasks (especially tasks related to text processing), we decided to consider the three most advanced models for this purpose:

1. Convolutional neural networks (CNNs) are a type of ANN that is particularly well suited for processing sequential data. CNNs use a series of convolutional layers to extract features from data. Convolutional layers work by sliding a filter over the data and calculating the dot product between the filter and the data. This process is repeated multiple times using different filters to extract different types of features from the data [13].
Pros: It is good at extracting spatial features from data, relatively efficient to train and deploy, and highly parallelizable.
Cons: difficult to design and tune for specific tasks; prone to overfitting; can lose parallelizability due to the nature of sequence tasks.
2. Long short-term memory (LSTM) networks are another type of ANN that is well suited for processing sequential data. LSTMs have a special architecture that allows them to learn long-term dependencies in data. LSTMs are often used in such a kind of task as machine translation and speech recognition [14].
Pros: very good at learning long-range dependencies in sequential data; relatively robust to noise in the data; shown state-of-the-art effectiveness on a wide variety of text processing tasks.
Cons: computationally expensive to train, especially on long sequences; difficult to tune for specific tasks; and cannot be processed in parallel because of the nature of the LSTM node.
3. Transformers are a newer type of ANN that has recently achieved state-of-the-art results on a variety of sequence-processing tasks, including machine translation and text summarization. Transformers use a self-attention mechanism to learn long-term dependencies in data [15].
Pros: achieve state-of-the-art results on a variety of sequence processing tasks, including those related to programming code and logical tasks; can learn long-range dependencies at once owing to the self-attention mechanism; relatively efficient to train and deploy, especially on large datasets.

Cons: difficult to tune for specific tasks; hard restrictions on input sequence length.

7. Selected Deep Learning models

We decided to compare completely different ANN models and determine whether our hypothesis could be proven using these models. Let us consider the tuning of the three ANN models for this task, its overall architecture, and its hyperparameter configurations.

The base principle for building Deep Learning ANN models involves segmenting the ANN architecture into distinct layers: the input layer, which receives the initial data; the feature extraction layers, which are some specific neuron layers that identify patterns and features within the data; the fully-connected layers, which synthesize the extracted features into a form suitable for making predictions or decisions; and finally, the output layer, which presents the final results of the model's computations [12]. Using this principle, all our models contain the required parts and mostly differ in the feature extraction part, which is based on the selected model architecture. By configuring the feature extraction part, one turns the complexity and volume of the feature vector. Similarly, by configuring the fully connected part, we can set both the output type and problem type to solve the model, also this part is important because of its capability to analyze extracted features.

Consider our tuned convolutional-based model shown in Fig 1. There are two options for defining the entire model for processing sequences: the first is to use truncating and paddings to set the input sequence length, as shown in our practice, this option struggles due to a variety of term lengths, but trains much faster because data can be split in batches; the second is to use the entire sequence, but in this case, the feature extraction part always gives different output shapes to solve this problem, we use GlobalAveragePooling1D, which is shown in Fig 1. Our convolutional-based model has four Conv1D layers, each of which has a stride value of two, imitating the pooling layer. Thus, the basic idea of this solution is to extract features from N 1-d vectors representing terms, distributing them to a 128-dim feature vector with increasing feature complexity. Subsequently, we applied a 256-dim dense layer for collecting and transforming as much feature information as possible to classify the Softmax layer for 31 units, which represent count reduction steps from 0 to 30.

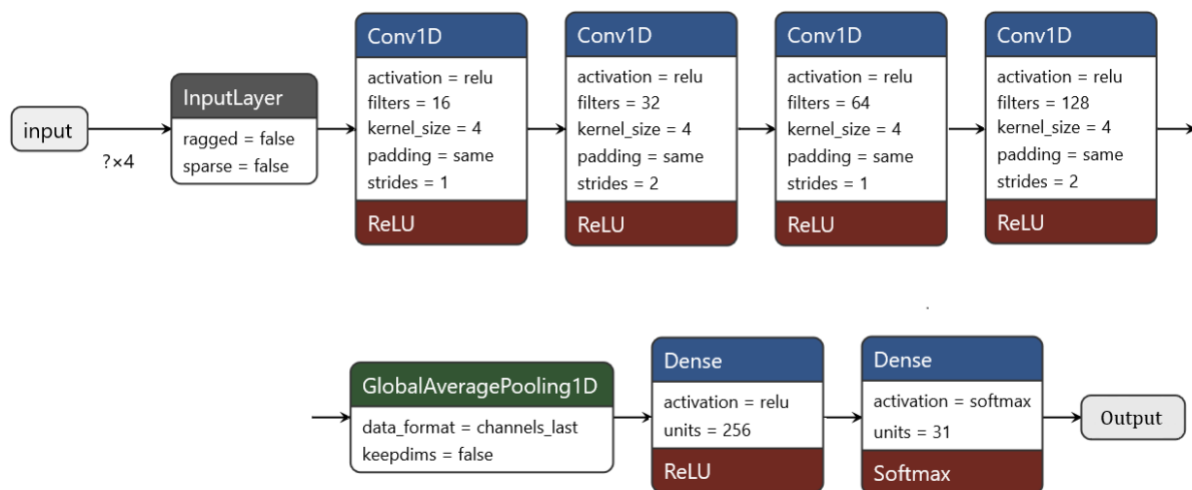


Figure 1: Architecture of the convolutional-based model used to estimate the count term reduction steps for the selected strategy

Fig 2 shows the LSTM-based model used in our experiments. As the convolution-based model LSTM-based models for sequence processing have two variants of implementation, we choose the second with allows different sequence lengths, which also decreases the required weights because only the last LSTM layer output is in the fully connected layer. Experiments show that the LSTM layer with 256 nodes is sufficient for extracting and saving required features. Applying

the same principles as the convolutional-based model led to the use of the same fully connected part.

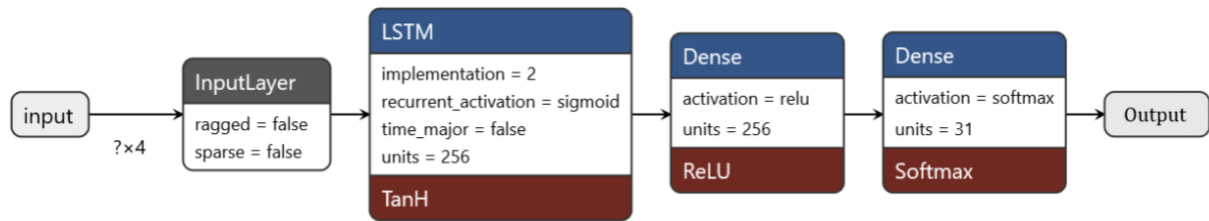


Figure 2: Architecture of the LSTM-based model used to estimate the count term reduction steps for the selected strategy

In Fig 3 you can find a Transformer-based model based on the Bert layer [15], which is an encoder-only or autoencoding model that can process limited sequences only, based on our sequence length distribution and classical Bert model [15], we set the input sequence length limit to 512 tokens. Consider the transformer architecture in Fig 3 in more detail. It has two input vectors: the input vector of IDs that are our trunked or padded input vector and the attention mask, which is simply a boolean vector, which indicates how the input IDs are filled. Our Bert layer consists of 2 hidden layers with 84 units each, the intermediate layer contains 64 units, and this module contains 6 self-attention heads which are kind of independent feature extractors. The output of the Bert layer is a sparse matrix 512 x 84 to prevent overfitting and minimize required computational resources we apply AveragePooling1D with pooling size and strides 50 which gave us a matrix 10 x 84 or after applying Flatten layer gives 840-dim feature vector, and Dropout with rate 0.1. As the fully connected we leave the Dense layer with 31 units.

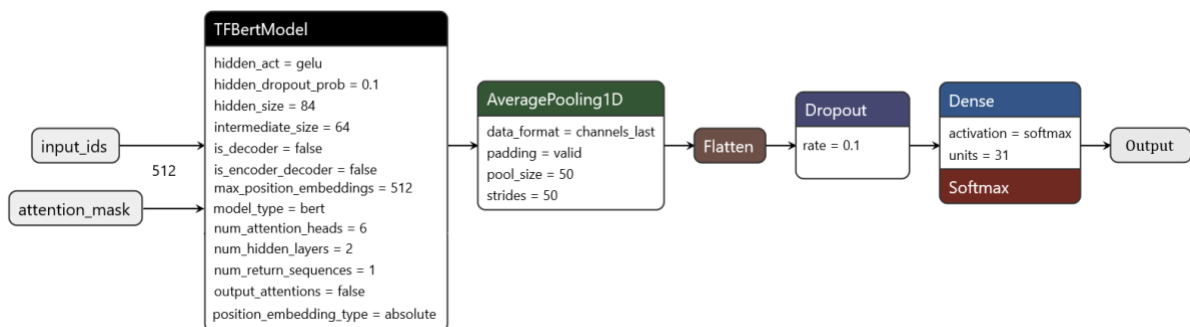


Figure 3: Architecture of the Transformer-based model used to estimate the count term reduction steps for the selected strategy

The final model complexities and trainable weights are listed in Table 1. The LSTM-based model has the largest number of weights. The Convolution-based model has evenly distributed weights between the feature extractor and the fully connected parts, which is explained by the necessity of analyzing the collected features and the specificity of the convolutional feature extractor part, which does not require as many weights as fully connected layers. The smallest fully connected part has a transformer-based model owing to the high feature complexity of the Transformer-based feature extractor.

Table 1
Comparison of model weights

	Convolution-based	LSTM-based	Transformer-based
Feature extractor weights	43504	43504	43504
Fully connected weights	267264	267264	267264
All trainable weights	130832	130832	130832

8. Experiments and data collection

8.1. Data generation

For the term generation procedure, we use a recursive random-based algorithm, which allows us to evenly distribute Variables, Applications, and Abstraction across the term body, which allows us to collect as many possible term variants for the defined distribution of variables. The algorithm consists of the following steps:

1. Set probabilities for generation Abstraction, Application, and Atom terms.
2. Set a list of variables used for the generation of terms.
3. Set the maximum limit of vertices in the generated term.
4. Considering probabilities, perform one of the three actions:
 - a. Generate an Atom term based on a random variable from the list of variables.
 - b. Generate an Abstraction term based on a head, which is a random variable from the list of variables, and a body, which is a randomly generated term with options 1 and 2 and a decreased number of vertices.
 - c. Generate an Application term based on the subject and object, which are randomly generated with terms with options 1 and 2, and a decreased number of vertices.
5. Decrease the limit of vertices on one and return to 4 until the limit of the vertices reaches 0.

After generating a batch of terms, we filter them out on actual count vertices, similar terms, and reduction filtering; for example, in the LO strategy, we can filter out terms represented by quite long sequences appearing while reducing.

8.2. Datasets and metrics

To increase the collection of terms and obtain terms with different counts of steps for the selected strategies, we normalize the collected terms with the LO strategy. For each term, we obtain its simplified form and count steps for the LO and the RI strategies. After filtering out similar terms (in simplified terms) and formulating datasets for each strategy, we cut out terms that have 31 or more steps on the reduction left. To increase the complexity of the test/validation set, we introduced more filters into the term generation procedure, which gave us an independent dataset.

Table 2

Generated dataset sizes with simplified terms and its LO and RI reduction steps

		LO datasets	RI datasets
Train set	Samples	38272 (~91%)	34851 (~91%)
	Tokens	~4.5M	~3.8M
Test	/ Samples	3691 (~9%)	3431 (~9%)
Validation set	Tokens	~405k	~340k
Total		41963	38282

It should be noted that generating and collecting current count training samples (shown in Table 2) was not random. We focused on the Chinchilla low recommendation for training the transformer model, which states that for training a transformer model, we should have the number of training tokens at least 20x from count trainable weights [21]. So as shown in Table 1 our current Transformer-based model has 156k trainable weights, so we need at least 3M training tokens, and as shown in Table 2 we have a minimum of 3.8M training tokens which should be sufficient for training current models.

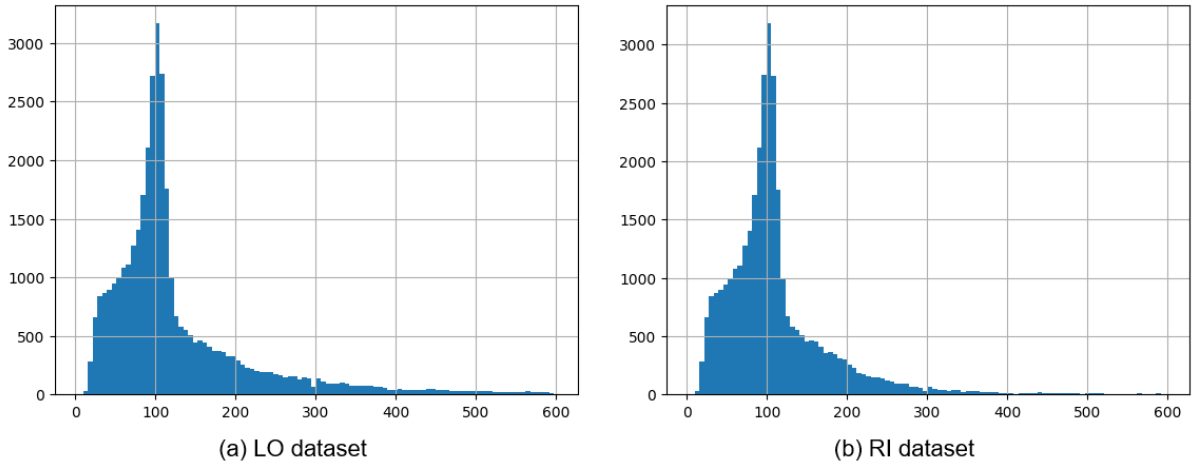


Figure 4: Term sequences length distribution for (a) LO dataset, (b) RI dataset

Consider analyzing the sequence distribution of collected terms, although datasets for LO and RI steps are built on the same terms, they can still have different reduction lengths, which impact on filtered out records in both datasets. The length sequences for LO terms and RI terms are shown in Fig 4, as you can see these plots are approximately the same, but LO terms have longer terms (see Fig 4(a)). In addition, we must admit that choosing 31 classes as model outputs stipulates count reduction step distributions, as shown in Fig 5. For distribution LO steps, as you can see in Fig 5(a) most terms are in boundaries 30 steps, and the same is true for distribution RI steps shown in Fig 5(b).

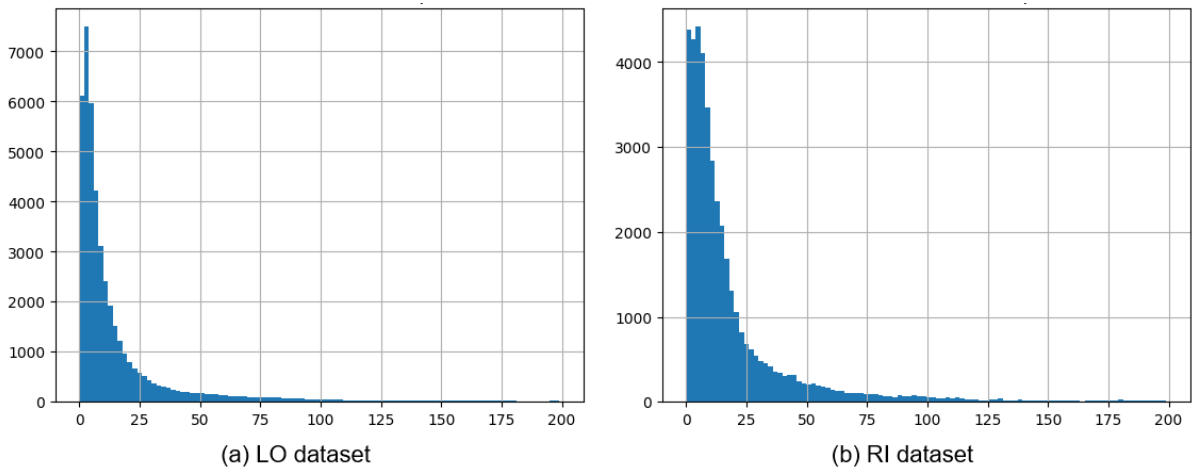


Figure 5: Reduction steps number distribution for (a) LO dataset and (b) RI dataset

9. Model Training

We decided to solve this problem as a classification problem. The classical strategy for training a Machine Learning model for this problem is to select the categorical cross entropy as the loss function. We also determine a training procedure by training the model with checkpoints to save the best by the validation accuracy model to a file, so that we can train the model to an overfitting state of still having the most robust solution. It is also necessary to emphasize that we use the Adam optimizer, which allows automatic adjustment of the learning rate and takes into account the gradient momentum [21]; practically, it is the best for training models for sequential processing, especially transformers [11, 15].

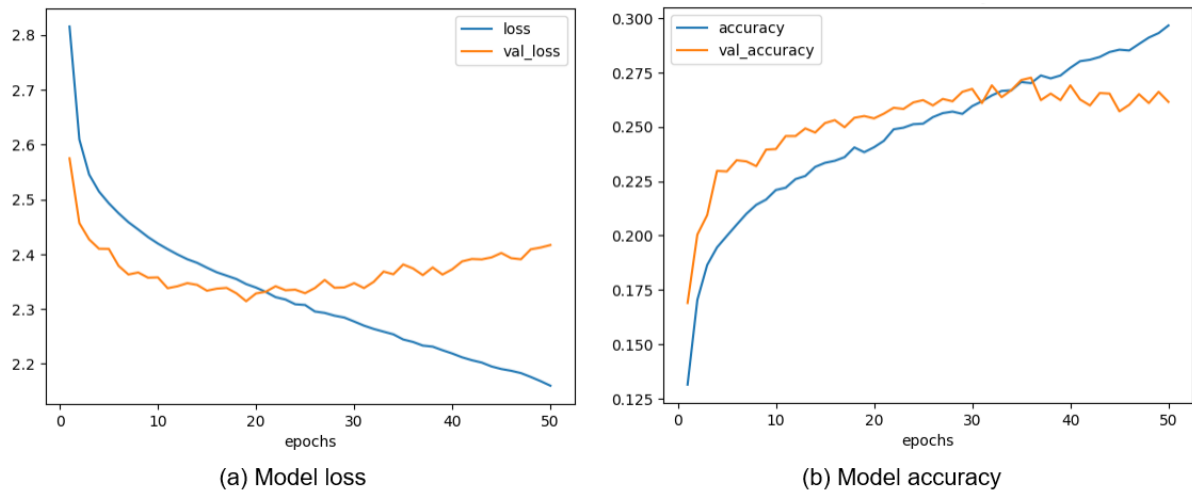


Figure 6: Transformer-based model training curves on the LO dataset: (a) loss curve, (b) accuracy curve

In total we trained 6 models: two models of each selected type were trained on LO and RI datasets. Typical curves for the training and validation sets representing the learning process are shown in Fig 6(a) for loss value and accuracy in Fig 6(b). The curves shown in Fig 6 represent the training of the Transformer-based model for the prediction of count LO steps. In these plots (both loss and accuracy), we should consider moments when the training and validation curves begin to diverge, which means that the model tends to overfit and cannot learn any more common features for the train and validation datasets; therefore, we should stop model training from here or save current model weights. Therefore, as a training strategy, we choose to set more count training epochs than required and validation accuracy tracking: saving model weights that provide the best validation accuracy. The saved models were used for further testing.

10. Model performance and generalization

As mentioned earlier, we evaluated model performance not only using accuracy metric, which is typical for classification problems but also using Mean Absolute Error and Root Mean Squared error. The results of the training Convolution-based, LSTM-based, and Transformer-based models on the LO-specific data are shown in Table 3. The overall performance did not seem high, and despite the high accuracy of the LSTM-based model, it still had the same MAE and RMSE levels, indicating that the model had many overestimations. In addition, we must admit that the high difference between MAE and RMSE values for all models indicates that for some cases, models made big mistakes like a term had 30 reduction steps, but the models predicted 0. In addition, the inherent performance of the Transformer-base model can be explained by the fact that it has limited input sequence lengths and cannot provide a full sequence view for some samples.

Table 3:
Result of training selected models on the LO dataset

	Accuracy		MAE		RMSE	
	train	test	train	test	train	test
Convolution-based	33.6%	31.7%	2.86	2.75	4.84	4.7
LSTM-based	40.0%	37.8%	2.91	2.74	5.28	5.16
Transformer-based	29.7%	27.2%	3.1	2.36	5.31	5.55

The results of training the same models on RI-specific data are shown in Table 4. Comparing the results achieved on identical models on the LO and RI datasets shows that models trained on the RI dataset have better performance, which can be explained by the fact that, on average, LO term data have more tokens than RI term data (shown in Fig 4). The LSTM-based model trained on RI data exhibited the best performance and lowest error rate. In addition, we should emphasize that the models trained on RI had the lowest overall MAE and RMSE values, which indicates a low step prediction error.

Table 4:
Result of training selected models on the RI dataset

	Accuracy		MAE		RMSE	
	train	test	train	test	train	test
Convolution-based	48.7%	47%	1.33	1.44	2.59	2.76
LSTM-based	68.8%	54.3%	0.509	1.29	1.25	2.7
Transformer-based	33.4%	28.6%	1.77	2.25	2.98	3.63

A detailed performance comparison is presented in Fig 7, here are plots with comparison MAE values depending on the actual reduction step count (term complexity, in other words). These plots are achieved in the following way: all data is split into 31 sets by actual reduction count and for each set tested on a corresponding model and using real and predicted values calculated MAE, that gives us one dot on the plot with coordinates: (step, MAE). This approach allows the analysis of model performance on different term complexities.

As shown in Fig 7 a simpler term means a lower error, and the maximum error value is higher for LO-data trained models. For all plots except Fig 7(e), the training error is close to the test error, which indicates that our strategy for preventing overfitting works well on these datasets.

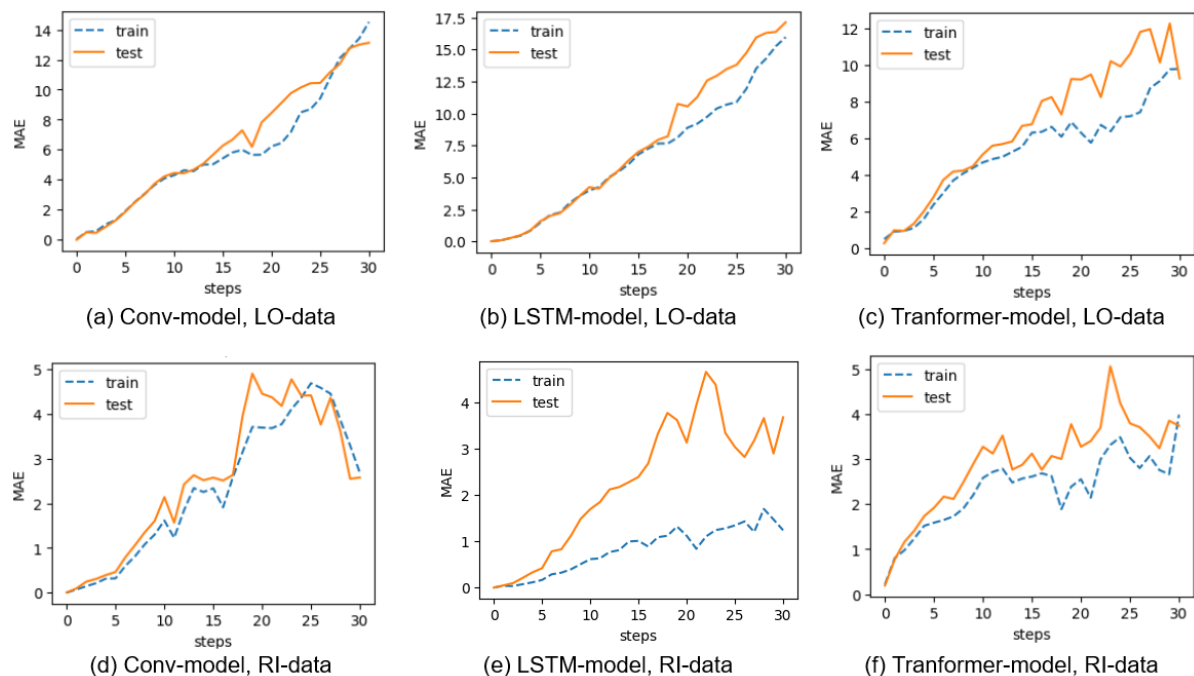


Figure 7: MAE depending on term complexity for: (a) Convolution-model and LO-data, (b) LSTM-model and LO-data, (c) Transformer-model and LO-data, (d) Convolution-model and RI-data, (e) LSTM-model and RI -data, (f) Transformer-model and RI -data

11. Conclusions and Discussion

Using trained models to define the number of reduction steps for a given term is a very promising approach for automating the search for the optimal reduction strategy for each term independently. In combination with our approach for estimating the computational resources required for one reduction step [3], it may decrease the time for the entire term normalization process. With these results, we can expand the scope of programming language optimization.

Our findings show that a simplified term representation is sufficient for predicting count reduction steps for a given strategy, but for eliminated depth. The experiments demonstrate that models struggle to process long sequences, and terms that are more complex in terms of the number of reduction steps tend to yield less reliable predictions. Furthermore, it is crucial to consider the removal of information regarding term variables, as this has a substantial effect on the behavior of term reduction.

12. Future work

The high error rate for models trained on the LO dataset with longer token sequences and error growth due to increasing term complexity indicate that using these models for determining the optimal choice between the Leftmost Outermost and the Rightmost Innermost strategies is not sufficiently robust. Therefore, in future research, we can consider term representation with variables or term combinatorial representation [22], which eliminates the use of variables in some cases.

However, training independent models for each reduction strategy and choosing an optimal strategy from a given strategy batch is not an effective solution. We can consider training a single model that provides the answer to which strategy is the most efficient. Furthermore, it would be beneficial to create generative models that identify the optimal redex for each term reduction step, which can help develop a balanced strategy in terms of reducing complexity and count reduction steps.

References

- [1] Alan M. Turing. Computability and λ -Definability. *The Journal of Symbolic Logic*. 2 (December 1937): 153–163. DOI:10.2307/2268280.
- [2] 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>.
- [3] Oleksandr Deineha, Volodymyr Donets, and Grygoriy Zholtkevych. Estimating Lambda-Term Reduction Complexity with Regression Methods. *CEUR Workshop Proceedings 2023*, in press.
- [4] Clemens Grabmayer. Linear Depth Increase of Lambda Terms along Leftmost-Outermost Beta-Reduction. November 2019. URL: <https://doi.org/10.48550/arXiv.1604.07030>.
- [5] 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.
- [6] Ugo Dal Lago, Gabriele Vanoni. On randomised strategies in the λ -calculus. *Theoretical Computer Science*, Volume 813, Pages 100-116 (2020). DOI:10.1016/j.tcs.2019.09.033.
- [7] Xiaochu Qi. Reduction Strategies in Lambda Term Normalization and their Effects on Heap Usage. 2004. URL: <https://arxiv.org/abs/cs/0405075>.
- [8] Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, Jie Tang. GPT Can Solve Mathematical Problems Without a Calculator. *Machine Learning* (September 2023). URL: <https://arxiv.org/abs/2309.03241>.
- [9] Chenxiao Liu¹, Shuai Lu, Weizhu Chen, Daxin Jiang, Alexey Svyatkovskiy, Shengyu Fu, Neel Sundaresan, Nan Duan. Code Execution with Pre-trained Language Models. Accepted to the Findings of ACL 2023. URL: <https://arxiv.org/abs/2305.05383>.

- [10] Chris Cummins, Volker Seeker, Dejan Grubisic, Mostafa Elhoushi, Youwei Liang, Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Kim Hazelwood, Gabriel Synnaeve, Hugh Leather. Large Language Models for Compiler Optimization. (September 2023). URL: <https://arxiv.org/abs/2309.07062>.
- [11] Brando Miranda, Avi Shinnar, Vasily Pestun, Barry Trager. Transformer Models for Type Inference in the Simply Typed Lambda Calculus: A Case Study in Deep Learning for Code. Computer Science (March 2023). URL: <https://arxiv.org/abs/2304.10500>.
- [12] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep Learning. *Nature* 521: 436-444 (2015). DOI:10.1038/nature14539.
- [13] Liang Yao, Chengsheng Mao and Yuan Luo. Graph Convolutional Networks for Text Classification. AAAI Conference on Artificial Intelligence (September 2018). DOI:10.1609/aaai.v33i01.33017370.
- [14] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, SangUk Kang, and Jong Wook Kim. Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism. *Applied Sciences* (2020). DOI:10.3390/app10175841.
- [15] Jimmy J. Lin, Rodrigo Nogueira, and Andrew Yates. Pretrained Transformers for Text Ranking: BERT and Beyond. Proceedings of the 14th ACM International Conference on Web Search and Data Mining (October 2020). DOI:10.1145/3437963.3441667.
- [16] M. M. Krell, Bilal Wehbe. A First Step Towards Distribution Invariant Regression Metrics. 2020. URL: <https://arxiv.org/abs/2009.05176>.
- [17] Yoshua Bengio, Paolo Frasconi. Input-output HMMs for sequence processing. *IEEE transactions on neural networks* 7 5: 1231-49 (1996). DOI:10.1109/72.536317.
- [18] William M. Campbell, Elliot Singer, Pedro A. Torres-Carrasquillo, and Douglas A. Reynolds. Language recognition with support vector machines. *The Speaker and Language Recognition Workshop* (2004).
- [19] Sin-Wai Chan. *Routledge Encyclopedia of Translation Technology*. Routledge. p. 454 (November 2014). ISBN 978-1-317-60815-8.
- [20] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and L. Sifre. Training Compute-Optimal Large Language Models. (2022). URL: <https://arxiv.org/abs/2203.15556>.
- [21] Sebastian Ruder. An overview of gradient descent optimization algorithms. *Vestnik kompiuternykh i informatsionnykh tekhnologii* (2016). URL: <https://arxiv.org/abs/1609.04747>.
- [22] Hendrik Pieter Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Vol. 103 (1984). ISBN 0-444-87508-5.