Харківський національний університет імені В. Н. Каразіна
Міністерство освіти і науки України

Кваліфікаційна наукова праця
На правах рукопису

ДЕЙНЕГА ОЛЕКСАНДР АНДРІЙОВИЧ

УДК 004.4`6:004.4`4

ДИСЕРТАЦІЯ

ОПТИМІЗАЦІЯ ФУНКЦІОНАЛЬНИХ МОВ ПРОГРАМУВАННЯ НА ОСНОВІ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність 122 – Комп’ютерні науки
Галузь знань 12 – Інформаційні технології

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
_____ **О. А. Дейнега**

Науковий керівник: Жолткевич Григорій Миколайович доктор технічних наук, професор

Харків – 2024

АНОТАЦІЯ

Дейнега О. А. Оптимізація функціональних мов програмування на основі методів штучного інтелекту. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 122 – Комп'ютерні науки (Галузь знань 12 – Інформаційні технології). – Харківський національний університет імені В. Н. Каразіна, Міністерства освіти і науки України, Харків, 2024.

Дисертація присвячена оптимізації функціональних мов програмування, на основі методів штучного інтелекту, що є складною та важливою задачею з багатьма проблемами та викликами. В дисертації розглянуто лямбда-числення як приклад відносно простої репрезентації функціональних мов програмування, що дозволяє показати процеси компіляції та інтерпретації функціональних мов програмування шляхом редукції лямбда-термів.

У **першому розділі** описано теоретичну частину дослідження. Представлено функціональні мови програмування, як інструмент верифікації програмного забезпечення. Описано переваги функціональних програм, такі як простота тестування та надійність коду, а також їх недоліки, основним з яких є низька продуктивність. Далі постає описання лямбда-числення, як одного з найпростіших варіантів представлення функціональних мов програмування. Пояснюється можливість переходу від роботи з функціональними мовами програмування до лямбда-числення. Далі представлені підходи для оптимізації лямбда-числення, основним із яких є удосконалення стратегій редукції лямбда-термів.

Далі текст заглиблюється в зв'язок між лямбда-численням і верифікацією програм в контексті паралельного програмування. Підкреслюється, як лямбда-числення служить основою для аналізу та розуміння поведінки паралельних програм. Використовуючи властивості лямбда-числення, розробники можуть застосовувати формальні методи перевірки, щоб переконатися, що паралельні

програми виконуються правильно та відповідають призначеним специфікаціям. Це включає перевірку таких властивостей, як безпека, живучість і правильність у різних сценаріях виконання.

Описано важливість формальної перевірки для паралельних програм, особливо з огляду на потенційні складності та проблеми, пов’язані з одночасним виконанням. Використовуючи формальні методи, засновані на лямбда-численні, розробники можуть отримати впевненість у надійності та правильності свого паралельного програмного забезпечення.

Далі у першому розділі розглянуті бібліотеки Python для роботи з лямбда численням, виведені їх недоліки та для усунення виявлених недоліків розроблено власну бібліотеку Lambda Calculus Environment. Показано класову архітектуру розробленої бібліотеки та зазначено функції, що виконують зазначені класи. Також зазначено вимоги яким задовольняє розроблена бібліотека і вимоги до функціонування розробленого програмного забезпечення. Згідного вимог до розробленого програмного забезпечення визначено багатоетапну процедуру верифікації. Процедура визначає певний набір лямбда-термів із ключовими умовами які мають бути виконані й враховані при виконанні на них доступних операції. Також процедура визначає перевірку виконання кожного із зазначених пунктів при додаванні нового функціоналу, та перевизначення множини лямбда-термів для тестування.

У **другому розділі** представлений підхід до оптимізації стратегій редукції, що базується на змішуванні стратегій та використанні рандомізованих стратегій. Ідея даного підходу відноситься до теорії ігор, де в деяких випадках використання стратегій в чистому вигляді не може призвести до перемоги. Описані результати, що показують ефективність даного підходу, та можливість заміни чистих стратегій змішаними, що дозволяють зберегти існуючу продуктивність, проте підвищити загальну вірогідність успішного редукування термів.

Далі у розділі була розглянута концепція обчислювальної нерівнозначності редексів лямбда-термів, що є ключовими точками у виборі стратегії редукції.

Нерівнозначність була оцінена з використанням методів машинного навчання для вирішення задачі регресії. Ціллю регресії була оцінка часу виконання операції редукції для даного редексу по параметрам терму, що відображають його деревну структуру. В результаті було отримано відхилення від очікуваного логарифму часу в 0.28 для регресійної моделі на базі штучної нейронної мережі та в 0.28 для лінійної регресії, варто зазначити також незначне падіння точності на тестовому наборі, що свідчило про достатню спроможність зазначених методів вилучати необхідні характеристики для оцінки часу редукції. Застосування методів дерев рішень та опорних векторів для вирішення цієї задачі також не показали високих результатів точності.

Далі у розділі розглядається використання даних про стан терму до і після процесу редукції для оцінки часу, проте без значних покращень у підвищенні точності оцінки часу редукції. Показано, що подальші дослідження цих аспектів можуть збільшити точність оцінки обчислювальних витрат. Також показано, що точна оцінка обчислювальних витрат може допомогти розробити жадібну стратегію з мінімізацію витраченого часу на процес редукції проте без урахування кількості кроків редукції.

У **третьому розділі** була перевірена можливість оцінки кількості кроків редукції лямбда-термів за заданою стратегією із застосуванням методів глибинного навчання. Для цього було використано спрощене текстове представлення лямбда термів із видаленням інформації про змінні. Аналіз проводився з використанням методів глибинного навчання для аналізу послідовностей. Показано, що точних результатів оцінки можливо досягти при визначенні 0-2 кроків редукції проте із збільшенням очікуваних кроків редукції зростала помилка в оцінці. Це свідчило про втрату важливої інформації при використанні спрощеного представлення лямбда-термів та недостатньої спроможності використаних моделей до глибинного аналізу термів.

Далі було досліджено можливість використання вбудовувань для репрезентації різниці в редукції лямбда термів різними стратегіями. Для цього було

розглянуто чотири моделі LLM для генерації вбудовувань з текстових представлень лямбда-термів. Це дозволяє проаналізувати можливість використання вбудовувань, отриманих з LLM, для вилучення характеристик, що впливають на редукцію термів та в перспективі можуть допомогти розробити компілятори та інтерпретатори функціональних мов програмування.

Згенеровані вбудовування були використані для створення восьми наборів даних для кожної розглянутої моделі LLM та для стратегій редукції термів LO та RI. Ці набори даних містять вбудовування по обраній LLM та кількість кроків редукції для кожного з розглянутих термів за обраною стратегією редукції. Для оцінки якості репрезентації інформації у вбудовуваннях були використані моделі ШНМ, що вирішували проблему класифікації відносно кроків редукції від 0 до 30.

Далі навчені моделі ШНМ були оцінені з показниками для оцінки точності регресії MAE, RMSE. Ці коефіцієнти було порівняно з найкращими результатами, досягнутими для того самого завдання та набору даних зі спрощеним представлення термів. Результати вказують на покращення прогнозування кількості кроків, особливо значних покращень було досягнуто в прогнозуванні кількості кроків LO, що збільшило точність на 23% для показника MAE та на 30% для показника RMSE. Проте прогнози щодо кількості кроків для стратегії RI мали незмінно низький рівень помилок із незначними покращеннями показників MAE та RMSE. Такі результати вказують на те, що код і загальні LLM можуть допомогти отримати інформацію з лямбда-термів і використовувати цю інформацію для вибору оптимальної стратегії редукції. Спеціально навчені LLM можуть ще більше підвищити точність вилучення даних із лямбда-термів. Отже, метод вилучення ознак із використанням LLM може бути реалізований в реальних інтерпретаторах функціональних мов програмування, а вилучені дані можуть бути використані для оптимізації.

У **четвертому розділі** лямбда-терми були перетворені в усереднені вектори вбудовувань розміром 768, що були отримані в результаті застосування попередньонавченої моделі ШНМ для задач пов’язаних із аналізом програмного

коду Microsoft CodeBERT та подальшої обробки виходів середніх рівнів цієї моделі за принципом об’єднання слів у значущі вектори Word2Vec. Завдяки аналізу PCA та t-SNE візуалізацій цих усереднених векторних вбудовувань виявлено, що представлення лямбда-термів у цих усереднених вбудовуваннях можливо візуально чітко розрізнити. Це дозволило підтвердити гіпотезу можливості виділення значущих кластерів в цьому наборі вбудовувань. Також виходи CodeBERT моделі були оброблені із застосуванням автокодувальника проте це не дало бажаної точності візуального розділення даних.

Тому далі було досліджене формування кластерів даних із застосуванням методу DBSCAN, що використовує як евклідову, так і косинусну метрику, окрім методу агломеративної кластеризації з використанням евклідової, косинусної, L1 і L2 метрики. Ці зусилля з кластеризації підкреслили ефективність моделі CodeBERT у вилученні значущих характеристик із лямбда-термів. Незважаючи на це, універсальність Microsoft CodeBERT, навченого різними мовами програмування, вносить рівень складності в досягнення точного представлення термів лямбда-числення в матрицях вбудовувань. Ця складність поширюється на процес перетворення цих матриць у зрозумілі усереднені вбудовування або вектори прихованого простору, особливо при використанні автокодувальників.

Далі була оцінена інформативність змінних усереднених вбудовувань із застосуванням моделі ШНМ навченої на результатах кластеризації та градієнтного методу оцінки інформативності моделі ШНМ. Цей аналіз дозволяє краще зрозуміти вплив певних змінних на результати кластеризації, пропонуючи пояснення основного значення цих змінних у контексті лямбда-термів і мінливості результатів кластеризації.

Крім того, далі було запроваджено коефіцієнт перекриття, що полегшило оцінку взаємозалежності між кластерами та застосованими стратегіями. Ця оцінка виявила відсутність кореляції між попередньо визначеними пріоритетами стратегії та фактично досягнутою дискримінацією термів, що вказує на потенційну потребу

в тонкому налаштуванні моделі CodeBERT та вказує на необхідність розгляду альтернативних моделей, більш придатних для аналізу даних у цій області.

Також було продовжено ідею трансформації лямбда-термів у вектори вбудовувань з використанням моделей OpenAI з розміром векторів 1536, та 3072. Дані вектори були так само проаналізовані з застосуванням методів PCA та t-SNE для візуалізації цих векторів. Так само було виявлено можливість візуального розділення цих даних. Далі також було розглянуто формування кластерів даних із застосуванням методу DBSCAN з евклідовою метрикою. Також підкреслено здатність моделей OpenAI Embeddings вилучати важливі характеристики з лямбда-термів. Однак, навчання OpenAI Embeddings моделей проводилось не на представленнях лямбда-термів, а здебільшого на людському тексті та коді, що ускладнило точне зображення термів лямбда-числення в матрицях вбудовування.

В наступній частині четвертого розділу представлено підхід для використання LLM безпосередньо для проведення процесу редукції лямбда-термів. В даному підході на вхід моделі подаються лямбда-терм, а на виході очікується наступний крок редукції лямбда терму за обраною стратегію. Результати показали, що використання LLM для вирішення цієї задачі не є достатньо ефективним.

В останньому розділі дисертації представлено можливий варіант імплементації описаних методів для використання у компіляторах для підвищення їх продуктивності. Описано можливі ризики, що мають бути враховані при використанні даного підходу. Також у даній частині представлено варіант використання великих мовних моделей у якості засобу верифікації лямбда-термів, та функціональних програм в цілому. Оскільки великі мовні моделі мають великий потенціал з точки зору аналізу коду та можуть бути розвинені для забезпечення його надійності.

Сукупність результатів, викладених у дисертації, разом із підтвердженою науковою та практичною актуальністю демонструють досягнення поставленої мети щодо оптимізації функціональних мов програмування на базі методів штучного інтелекту. Цей успіх пояснюється впровадженням сучасних моделей і методів

машинного навчання та штучного інтелекту, що показали високі результати у вирішенні задач даного класу. Крім того, наукові та практичні результати, представлені в дисертації, у поєднанні з перевіркою їх достовірності та значущості, демонструють, що проблема оптимізації функціональних програм за допомогою методів штучного інтелекту була ефективно вирішена, і поставлена мета була досягнута.

Ключові слова: *штучний інтелект, великі мовні моделі, кластеризація, глибинне навчання, інформативність характеристик, машинне навчання, нейронні мережі, профілювання процесу редукції, графове представлення, моделювання лямбда-числення, функціональні мови програмування, автоматизація вибору стратегії редукції, симуляція процесу редукції, верифікація програмного забезпечення.*

ABSTRACT

Deineha O. A. Optimization of functional programming languages based on artificial intelligence methods. – Qualification scholarly paper: a manuscript.

The dissertation submitted for obtaining the Doctor of Philosophy degree in Technical Science: Speciality 122 – Computer science. V. N Karazin Kharkiv National University, Ministry of Education and Science of Ukraine, Kharkiv, 2023.

The dissertation is devoted to the optimization of functional programming languages based on artificial intelligence methods, which is a complex and important task with many problems and challenges. The thesis examines lambda calculus as an example of a relatively simple representation of functional programming languages, which allows us to show the processes of compilation and interpretation of functional programming languages by reducing lambda terms.

The **first chapter** describes the theoretical part of the research. Functional programming languages are presented as an effective tool for solving many problems including program verification. The advantages of functional programs, such as ease of testing and code reliability, are described, as well as their disadvantages, the main of which is low performance. Next comes the description of lambda calculus, as one of the simplest options for representing functional programming languages. The possibility of transition from working with functional programming languages to lambda calculus is explained. Next, approaches for optimizing the lambda calculus are presented, the main of which is the improvement of strategies for the reduction of lambda terms.

Later in this section, the text delves into the connection between lambda calculus and program verification in the context of parallel programming. It highlights how lambda calculus serves as a foundation for analyzing and understanding the behavior of parallel programs. By leveraging the properties of lambda calculus, developers can apply formal verification techniques to ensure that parallel programs execute correctly and adhere to their intended specifications. This includes verifying properties such as safety, liveness, and correctness in various execution scenarios.

The section emphasizes the importance of formal verification for parallel programs, especially given the potential complexities and challenges associated with concurrent execution. By using formal methods based on lambda calculus, developers can gain confidence in the reliability and correctness of their parallel software.

Further, in the first chapter, the Python libraries for working with lambda calculus are considered, their shortcomings are deduced, and Lambda Calculus Environment library was developed to eliminate the identified shortcomings. The class architecture of the developed library is shown and the functions performed by the specified classes are indicated. The requirements that the developed library satisfies are also indicated and the functioning requirements for the developed software are also indicated. According to the functioning requirements, a multi-stage verification procedure is defined. The procedure defines a certain set of lambda terms with key conditions that must be fulfilled and considered when performing available operations on them. Also, the procedure determines the verification of each execution of each of the specified points when adding new functionality, and the redefinition of a lambda terms set for testing.

The **second chapter** presents an approach to optimization of reduction strategies based on mixing strategies and using randomized strategies. The idea of this approach refers to the theory of games, where in some cases the use of strategies in their pure form cannot lead to victory. The results are described, showing the effectiveness of this approach and the possibility of replacing pure strategies with mixed ones, which allow maintaining the existing performance, but increasing the overall probability of successful term reduction.

Further in the section, the concept of computational inequality of lambda-term redexes, which are key points in choosing a reduction strategy, was considered. The disparity was estimated using machine learning techniques to solve the regression problem. The purpose of the regression was to estimate the time of the reduction operation for a given redex based on the parameters of the term reflecting its tree structure. As a result, a deviation from the expected logarithm of time was obtained in 0.28 for the

regression model based on an artificial neural network and 0.28 for linear regression, it is also worth noting a slight drop in accuracy on the test set, which indicated the sufficient ability of the specified methods to extract the necessary characteristics for estimating the reduction time. The use of decision tree and support vector methods to solve this problem also did not show high accuracy results.

Next, the section examines the use of term state data before and after the reduction process to estimate the time, but without significant improvements in the accuracy of the reduction time estimate. It is shown that further studies of these aspects can further increase the accuracy of the estimation of computational costs. It is also shown that an accurate estimate of the computational cost can help to develop a greedy strategy to minimize the time spent on the reduction process, however, regardless of the reduction steps number.

In the **third chapter**, the possibility of estimating the number of lambda term reduction steps according to a given strategy using deep learning methods was tested. For this, a simplified textual representation of lambda terms was used, with information about variables removed. The analysis was carried out using deep learning methods for sequence analysis. It is shown that accurate estimation results can be achieved when determining 0–2 reduction steps, however, with the increase of the expected reduction steps, the error in the estimation increased. This indicated the loss of important information when using a simplified representation of lambda terms and the insufficient capacity of the used models for in-depth analysis of terms.

Next, the possibility of using embeddings to represent the difference in the reduction of lambda terms by different strategies was investigated. For this, four LLM models were considered for generating embeddings from text representations of lambda terms. This allows us to analyze the possibility of using the embeddings obtained from LLM to extract characteristics that affect the reduction of terms and in the future can help to develop compilers and interpreters of functional programming languages.

The generated embeddings were used to generate eight datasets for each LLM model considered and for the LO and RI term reduction strategies. These datasets contain

observations on the selected LLM and the number of reduction steps for each of the considered terms on the selected reduction strategy. ANN models were used to assess the quality of information representation in embeddings, which solved the classification problem with respect to reduction steps from 0 to 30.

Next, the trained ANN models were evaluated with indicators for assessing the accuracy of regression MAE, RMSE. These coefficients were compared with the best results achieved for the same task and data set with a simplified term representation. The results indicate an improvement in step count prediction, particularly significant improvements were achieved in LO step count prediction, which increased the accurate by 23% for the MAE indicator and by 30% for the RMSE indicator. However, the number of step predictions for the RI strategy had consistently low error rates, with slight improvements in MAE and RMSE. Such results indicate that the code and general LLMs can help extract information from lambda terms and use this information to select an optimal reduction strategy. Specially trained LLMs can further improve the accuracy of data extraction from lambda terms. Therefore, the feature extraction method using LLM can be implemented in real interpreters of functional programming languages, and the extracted data can be used for optimization.

In the **fourth chapter**, lambda terms were transformed into averaged embedding vectors of size 768, which were obtained as a result of applying a pre-trained ANN model for tasks related to the analysis of Microsoft CodeBERT software code and further processing of the outputs of the average levels of this model according to the principle of combining words in significant Word2Vec vectors. Through PCA and t-SNE analysis of visualizations of these averaged vector embeddings, it is found that the representation of lambda terms in these averaged embeddings can be visually clearly distinguished. This made it possible to confirm the hypothesis of the possibility of identifying significant clusters in this set of embeddings. Also, the outputs of the CodeBERT model were processed using an autoencoder, which did not give the desired accuracy of visual data separation.

Therefore, the formation of data clusters using the DBSCAN method using both Euclidean and cosine metrics, in addition to the agglomerative clustering method using Euclidean, cosine, L1 and L2 metrics, was further investigated. This clustering effort highlighted the effectiveness of the CodeBERT model in extracting meaningful features from lambda terms. Despite this, the versatility of Microsoft CodeBERT, trained in various programming languages, introduces a level of complexity in achieving an accurate representation of lambda calculus terms in embedding matrices. This complexity extends to the process of transforming these matrices into understandable averaged embeddings or hidden space vectors, especially when using autoencoders.

Next, the informativeness of the averaged embedding variables was evaluated using the ANN model trained on the results of clustering and the gradient method of assessing the informativeness of the ANN model. This analysis provides a better understanding of the effect of certain variables on the clustering results by offering an explanation of the underlying meaning of these variables in the context of lambda terms and the variability of the clustering results.

In addition, an overlap coefficient was further introduced, which facilitated the assessment of interdependence between clusters and applied strategies. This evaluation found no correlation between the predefined strategy priorities and actual term discrimination achieved, indicating a potential need for fine-tuning of the CodeBERT model and indicating the need to consider alternative models more suitable for data analysis in this area.

The idea of transforming lambda terms into embedding vectors was also continued using OpenAI models with vector sizes of 1536 and 3072. These vectors were also analyzed using PCA and t-SNE methods to visualize these vectors. The possibility of visual separation of these data was also discovered. Further, the formation of data clusters using the DBSCAN method with the Euclidean metric was also considered. The ability of OpenAI Embeddings models to extract important features from lambda terms is also highlighted. However, OpenAI Embeddings models were not trained on representations

of lambda terms, but mostly on human text and code, which made it difficult to accurately represent lambda terms in embedding matrices.

The next part of the fourth chapter presents an approach for using the LLM directly to perform the lambda term reduction process. In this approach, lambda terms are fed to the input of the model, and the next step of lambda term reduction according to the chosen strategy is expected at the output. The results showed that using LLM to solve this problem is not efficient enough.

The last chapter of the dissertation presents a possible implementation option. Possible risks that must be taken into account when using this approach are described. Also, this part presents the option of using large language models with full verification of lambda terms and functional programs in general. Because large language models have great potential in terms of code analysis and can be developed to ensure its reliability.

The set of results presented in the dissertation, together with the confirmed scientific and practical relevance, demonstrate the achievement of the set goal of optimizing functional programming languages based on artificial intelligence methods. The success is explained by the implementation of modern models and methods of machine learning and AI, which has shown its success in solving the tasks of this class. In addition, the scientific and practical results presented in the thesis, combined with the verification of their reliability and significance, demonstrate that the problem of optimizing functional programs using artificial intelligence methods was effectively solved, and the goal was achieved.

Keywords: artificial intelligence, LLMs, clustering, deep learning, informativeness of features, machine learning, neural networks, profiling of reduction process, graph representation, lambda-calculus simulation, functional programming languages, automation of reduction strategy selection, simulation of reduction process, verification of programs.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях, що входять до міжнародних наукометричних баз

1. Deineha, O., Donets, V., & Zholtkevych, G. (2024). The approach development of data extraction from lambda terms. *Eastern-European Journal of Enterprise Technologies.*

(*Особистий внесок здобувача: розробка основних ідей підходу екстракції даних з лямбда термів, проведення експериментів, написання частини тексту та його переклад.*

Особистий внесок Volodymyr Donets проведення експериментів та аналіз підходу екстракції даних з лямбда термів, написання частини тексту та його переклад.

Особистий внесок Grygoriy Zholtkevych розробка основних ідей підходу та концепції дослідження, аналіз отриманих результатів.)

2. Deineha, O. (2024). Supervised data extraction from transformer representation of Lambda-terms. *Radioelectronic and Computer Systems, 2024*(2), 19-29.

(*Особистий внесок: розробка ідей підходу екстракції даних з лямбда термів, дослідження існуючих моделей трансформерів, проведення експериментів, аналіз отриманих результатів, написання тексту та його переклад*)

Статті у наукових фахових виданнях України

3. Deineha O. The Clustering of Lambda Terms by Using Embeddings. Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». 2023. вип. 59. С.16-23.

(*Особистий внесок здобувача: аналіз підходів кластеризації лямбда термів за допомогою вбудовувань, проведення експериментів, дослідження типів вбудовувань, написання тексту та його переклад*).

4. Deineha, O. (2024). Lambda calculus term reduction: Evaluating LLMS' predictive capabilities. *Information Technology and Society*, 1(12), 51-55.

(*Особистий внесок: розробка та дослідження підходів оцінки прогнозивних здатностей LLM в розрізі роботи за лямбда-численням, аналіз існуючих моделей, написання тексту та його переклад*).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Deineha, O., Donets, V., Zholtkevych, G. (2023). On Randomization of Reduction Strategies for Typeless Lambda Calculus. In: Antoniou, G., et al. Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI 2023. Communications in Computer and Information Science, vol 1980. Springer, Cham. (Scopus)

6. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Estimating Lambda-Term Reduction Complexity with Regression Methods. International Conference "Information Technology and Interactions". (Scopus)

7. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Deep Learning Models for Estimating Number of Lambda-Term Reduction Steps. International Workshop of IT-professionals on Artificial Intelligence. (Scopus)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, Інтерфейс програмування застосунків (прикладний програмний інтерфейс)

BERT – Bidirectional Encoder Representations from Transformers, тип ШНМ

CBN – Call By Name, виклик за ім’ям

CBV – Call By Value, виклик за значенням

CHI – Calinski-Harabasz Index, індекс Калінського-Харабаша

CNN – Convolutional Neural Network, згорткові нейронні мережі, клас глибоких штучних нейронних мереж

DBI – Davies–Bouldin Index, індекс Девіса-Болдіна

DBSCAN – Density-based spatial clustering of applications with noise, алгоритм кластеризації заснований на щільності

GMM – Gaussian Mixture Model, метод кластеризації

GRU – Gated Recurrent Unit, тип ШНМ

HAC – Hierarchical Agglomerative Clustering, метод кластеризації

HMM – Hidden Markov Models, приховані моделі Маркова

IoT – Internet of Thing

JVM – Java Virtual Machine

kNN – k-Nearest Neighbors

LI – Leftmost Innermost

LLM – Large Language Model, велика мовна модель

LLVM – Low Level Virtual Machine, віртуальна машина низького рівня

LO – Leftmost Outermost, найбільш лівий найбільш зовнішній

LSTM – Long short-term memory, довга короткочасна пам’ять, архітектура рекурентних нейронних мереж

MAE – Mean Absolute Error, середня абсолютна похибка

MIRACL – Multilingual Information Retrieval Across a Continuum of Languages

ML – Machine Learning, машинне навчання

MSE – Mean Square Error, середньоквадратична помилка

MTEB – Massive Text Embedding Benchmark, тест вбудовування масивних об'ємів тексту

NLP – Natural Language Processing, розділ науки присвячений обробці природного мовлення

PCA – Principal Component Analysis, Метод головних компонентів, способів зменшення розмірності даних

RBF – Radial Basis Function

RI – Rightmost Innermost, найбільш правий найбільш внутрішній

RMSE – Root Mean Square Error, корінь середньоквадратичної помилки

RNN – Recurrent Neural Network, Рекурентні нейронні мережі, клас штучних нейронних мереж

RO – Rightmost Innermost

SVM – Support Vector Machine, метод опорних векторів

SVR – Support Vector Regression, регресія опорних векторів

t-SNE – t-distributed Stochastic Neighbor Embedding, T-розподілене вкладення стохастичної близькості, метод візуалізації даних

UR – Uniformly Random, стратегія редукції лямбда-термів

WCSS – Within cluster sum of squares, внутрішньо кластерна сума квадратів

ШІ – Штучний Інтелект

ШНМ – Штучна Нейронна Мережа

ВСТУП

Обґрунтування вибору теми дослідження. Вибір теми дослідження в галузі функціонального програмування та оптимізації стратегій редукції в безтиповому лямбда-численні зумовлений як теоретичною значущістю, так і практичним значенням у сучасному ландшафті розробки програмного забезпечення.

Верифікація програмного забезпечення. Постійно зростаюча складність програмних систем призвела до зростаючої потреби в надійному коді, який можна перевірити. У той час як традиційні імперативні мови програмування домінували в галузі протягом десятиліть, функціональне програмування набуло значної популярності у сфері верифікації програм завдяки своїм притаманним математичним властивостям.

Розвиток функціонального програмування як інструменту для верифікації програм насамперед пов’язаний з його формальними основами в математиці та логіці. Багато мов функціонального програмування, таких як Haskell, OCaml і Scala, ґрунтуються на лямбда-численні, формальній системі визначення функцій, яка, природно, піддається формальним системам міркувань і доказів.

Функціональні мови програмування невід'ємно пов'язані з процесом верифікації програм через їхню основу в математичних функціях і формальній логіці. На відміну від імперативних мов, де зміни стану та побічні ефекти можуть ускладнити розуміння поведінки програми, функціональні мови підкреслюють незмінність та оцінку виразів, засновану виключно на вхідних значеннях. Таке узгодження з формальними системами, такими як лямбда-числення, полегшує перевірку правильності програми за допомогою математичних доказів. Оскільки функціональні програми часто складаються з чистих функцій, вони природно піддаються формальним методам верифікації, включаючи індуктивні докази, теорію типів і рівняння. Крім того, сильний акцент у функціональних мовах на рекурсії, функціях вищого порядку та системах типів забезпечує більш передбачувану та аналізовану поведінку, що додатково полегшує автоматизовані

процеси перевірки. Декларативний характер функціонального програмування створює чіткий шлях для встановлення правильності програми, що робить його цінною парадигмою в розробці систем, де надійність і математична точність є критичними.

Вирішення фундаментального виклику функціонального програмування. Функціональне програмування стає все більш важливим у сучасній розробці програмного забезпечення, відомого своїм внеском у підвищення надійності програмного забезпечення, верифікації, валідації та масштабованості. Однак воно стикається зі значними проблемами, особливо з точки зору низької продуктивності та неефективного використання пам’яті.

Розвиток розуміння лямбда-числення. Дослідження зосереджено на методах оцінки складності редукції лямбда-термів з точки зору часу кожного з кроків, що традиційно розглядаються однаковими, та загальної кількості кроків. Це дослідження сприяє глибшому розумінню процесу редукції в лямбда-численні, що є основою функціонального програмування.

Інтеграція машинного навчання в оптимізацію функціональних програм та компіляторів. Це дослідження поєднує передові статистичні методи та методи машинного навчання, такі як лінійна регресія та моделі регресії ШНМ, щоб передбачити час редукції на основі характеристик термів. Інтеграція машинного навчання пропонує новий підхід до оптимізації компіляторів та інтерпретаторів функціональних мов програмування, що робить їх більш продуктивними та ресурсоефективними.

Внесок в еволюцію мов програмування. Оскільки індустрія програмного забезпечення все більше використовує парадигми функціонального програмування, це дослідження надає критичні висновки та інструменти для вирішення деяких ключових проблем. Покращуючи обчислювальну продуктивність функціональних програм за допомогою оптимізованих стратегій редукції, дослідження робить внесок у ширшу область розробки та оптимізації мов програмування.

Нові підходи до скорочення обчислень. Дослідження різноманітних характеристик, що впливають на час редукції терму, сучасними методами машинного навчання, є новим підходом в цій галузі. Ці методи відкривають можливості для автоматизованої оптимізації стратегій редукції, потенційно призводячи до значного прогресу функціональних програм.

Підсумовуючи, вибір теми дослідження обумовлений її потенціалом значного впливу на сферу функціонального програмування. Дослідження не тільки розглядає ключові проблеми в парадигмі, але й впроваджує нові підходи, які можуть призвести до значного покращення ефективності та результативності функціональних мов програмування та їх компіляторів.

Отже, все це визначає актуальність рішення **науково-прикладної задачі** удосконалення існуючих або розробки нових методів машинного навчання для екстракції характеристик із програмного коду функціональних мов програмування, що впливають на обчислювальні витрати на їх компіляцію чи інтерпретацію.

Вирішити поставлену задачу стає можливим із вибором в якості функціональної мови програмування лямбда-числення, що є найпростішим їх представником з обмеженнями по існуючим операціям, проте без обмежень в реалізації можливих програм.

Мета дослідження – підвищення продуктивності функціональних програм за критерієм часу виконання за рахунок прогнозованого удосконалення процесів редукції лямбда-термів на основі методів штучного інтелекту.

Завдання дослідження.

1. Розробка експериментального середовища лямбда-числення, що дозволяє генерувати терми та імітувати процес редукції, що представляє собою виконання функціональних програм.

2. Розробка та аналіз продуктивності параметричної рандомізованої стратегії в безтиповому лямбда-численні для можливості налаштування порядку процесу редукції під конкретні підмножини термів.

3. Оцінка складності одного кроку редукції з точки зору часу виконання для подальшої можливості вибору найпростіших кроків редукції під час виконання функціональних програм.

4. Кластерний аналіз термів, для виявлення існуючих підмножин термів зі схожими характеристиками, що дозволить подальшу роботу з окремими кластерами.

5. Оцінка кількості кроків редукції терму за обраною стратегією для подальшої можливості вибору найбільш продуктивної для конкретного терму.

6. Оцінка прогнозивних здатностей LLM в розрізі передбачення наступного кроку редукції, для розуміння їх потенціалу для подальших досліджень.

Об'єкт дослідження – процес редукції лямбда-термів, що забезпечує інтерпретацію та компіляцію функціональних мов програмування. Характеристиками цього процесу є кількість кроків та час виконання.

Предмет дослідження – методи наближеної оптимізації функціональних мов програмування за рахунок підвищення продуктивності процесів редукції на основі використання методів штучного інтелекту, а саме штучних нейронних мереж для обробки текстової інформації, методів кластерного аналізу, методів аналізу чутливості.

Методи дослідження. У роботі використовувався комплексний підхід із використанням широкого спектру архітектур моделей штучного інтелекту методів імітаційного та математичного моделювання, теорії графів, та методів кластерного аналізу для вирішення проблем редукції термів у безтиповому лямбда-численні.

Дослідження включає застосування передових архітектур штучних нейронних мереж як LSTM, CNN і Transformer, кожна з яких має унікальні можливості для аналізу і прогнозування в контексті лямбда-числення, зокрема в оптимізації процесів редукції.

Наукова новизна отриманих результатів полягає в наступному.

1. **Вперше розроблено** параметричну випадкову однокрокову стратегію редукції, що відрізняється від існуючих можливістю налаштування ймовірності вибору редексу для скорочення. Варіація параметрів дозволяє керувати процесом редукції лямбда-термів з метою підвищення його продуктивності для конкретних підмножин термів.

2. **Удосконалено** метод оцінки складності кроку редукції лямбда терму, що відрізняється від існуючих використанням методів машинного навчання. Це дозволило підвищити точність прогнозування обчислювальної складності кроку редукції довільного лямбда-терму.

3. **Удосконалено** експериментальне середовище емуляції лямбда-числення завдяки впровадженню процедури багатоетапної верифікації розробленого програмного забезпечення та автоматичної генерації лямбда-термів. Це дозволило забезпечити коректну реалізацію цього програмного забезпечення та створювати набори даних лямбда-термів для забезпечення теоретично необхідного покриття контрольованими конфігураціями лямбда-термів.

4. **Дістали подальшого розвитку** методи представлення лямбда-термів у вигляді вбудовувань (embeddings), що на відміну від існуючих представлень забезпечує використання штучних нейронних мереж архітектури типу Transformer. Це дозволило збільшити глибину аналізу коду лямбда-термів та підвищити точність екстрації характеристик, що впливають на процес редукції.

5. **Дістав подальшого розвитку** комплексний метод оптимізації стратегії редукції, що на відміну від існуючих поєднує моделі штучного інтелекту для оцінки часу (в кількості кроків) редукції з процедурою вибору стратегії нормалізації терму. Цей комплексний підхід є новим і в має потенціал скоротити загальний час нормалізації, що суттєво сприяє оптимізації програм, реалізованих функціональними мовами програмування.

Особистий внесок здобувача. Дослідження, зосереджене на оптимізації процесу редукції функціональних мов програмування, що представлені лямбда-численням, початково концептуалізовано та керовано науковим керівником здобувача. Однак, в подальшому дослідження було виконане здобувачем самостійно. Всі результати наведені в дисертації були обґрунтовані особистими дослідженнями автора. Також окремі положення були аргументовані з використанням робіт інших науковців та мають посилання в тексті дисертації. В індивідуальних наукових роботах були розглянуті тільки авторські розробки та ідеї.

Розробка ключових ідей та реалізація. Незважаючи на те, що головна тема дослідження була окреслена науковим керівником, здобувач особисто концептуалізував (доповнив) і розвинув основні ідеї, що лежать в основі впровадження дослідження.

Моделі штучного інтелекту. Значною частиною внеску здобувача було проектування та розробка підходів використання моделей штучного інтелекту, включаючи моделі лінійної регресії та глибинних нейронних мереж. Здобувач самостійно взявся за завдання побудови підходів використання цих моделей, налаштування параметрів і адаптації, для вирішення задач даного дослідження.

Кодування та виконання експериментів. Кодування і виконання експериментів проводилися в рамках командної роботи. У цьому аспекті здобувач відігравав центральну роль, включаючи практичне та теоретичне проектування експериментів, аналіз даних та інтерпретацію результатів. Здобувач активно співпрацював з членами команди, щоб забезпечити надійність та ефективність процесів дослідження.

Постійне обговорення та вдосконалення. Впродовж дослідження здобувач брав участь у постійних дискусіях з науковим керівником. Цей спільний діалог був вирішальним для вдосконалення підходу та узгодження його з теоретичною структурою, встановленою науковим керівником.

Автор дисертаційної роботи активно брав участь у наукових дискусіях, написанні наукових статей за темою дисертації та доповідях результатів на міжнародних наукових конференціях.

В роботі [1, 2] показано результати розробки основних ідей підходу екстракції даних з лямбда термів із застосуванням методів машинного навчання. В роботі [3] проаналізовано підходи до кластеризації вбудовувань лямбда термів, що були отримані із застосуванням штучних нейронних мереж на базі архітектури Transformer. В роботі [4] автором було показано основні підходи до оцінки прогнозивних здатностей великих мовних моделей щодо вибору оптимальної стратегії редукції лямбда термів. В наукових працях [5–7] автором було показано результати розробки випадкової стратегії редукції лямбда-термів та результати оцінки обчислювальної складності редукції лямбда-термів після аналізу деревної структури лямбда-термів.

Підсумовуючи, особистий внесок здобувача в це дослідження був значним, особливо в області генерації ідей, розробки моделей, алгоритмічного кодування та виконання експериментів. Здобувач відігравав важливу роль у просуванні досліджень у напрямку досягнення поставлених цілей, кульмінацією яких стали цікаві ідеї та внесок у сферу функціонального програмування та оптимізації функціональних мов програмування.

Зв’язок роботи з науковими програмами, темами, планами.

Дисертаційна робота виконана відповідно до:

1. Закону України No 2623-III «Про пріоритетні напрями розвитку науки і техніки» редакція від 20.02.2021 р., зокрема за напрямом «Інформаційні та комунікаційні технології»;

2. Закону України No 3715-VI «Про пріоритетні напрями інноваційної діяльності в Україні» редакція від 05.12.2012 р., зокрема за напрямом «Розвиток сучасних інформаційних, комунікаційних технологій, робототехніки»;

Наукові дослідження, викладені в дисертації, виконані згідно з напрямом наукової роботи кафедри Теоритичної та прикладної інформатики Харківського національного університету ім. В.Н. Каразіна.

Практичне значення отриманих результатів.

Дослідження, проведені в рамках цієї роботи, хоча переважно теоретичні, пропонують значні практичні наслідки у сфері функціонального програмування та оптимізації компіляторів. Практичне значення дисертації визначення далі.

Підвищення ефективності мов функціонального програмування. Розробка параметричної однокрокової випадкової стратегії для редукції в лямбда-численні безпосередньо впливає на ефективність функціональних програм. Відходячи від детермінованих алгоритмів до стратегій, заснованих на випадковому пошуку, дослідження створює основу для компіляторів, які можуть динамічно обирати оптимальні стратегії редукції, тим самим покращуючи продуктивність.

Покращення використання обчислювальних ресурсів. Пропонуючи методи прогнозування часу редукції певних редексів та загальної кількості кроків редукції, дослідження допомагає оптимізувати розподіл обчислювальних ресурсів. Це особливо корисно в середовищах, де ефективність використання ресурсів має вирішальне значення, наприклад у платформах хмарних обчислень.

Оптимізація в функціональних мовах програмування. Отримані результати розширюють сферу оптимізації в функціональних мовах програмування за межі традиційних методів. Завдяки інтеграції моделей машинного навчання для прогнозування кількості кроків редукції та оцінки вимог до обчислювальних ресурсів дослідження пропонує нові підходи до оптимізації всього процесу нормалізації термів у функціональних мовах програмування.

Основа майбутніх досліджень і розробок. Незважаючи на деякі обмеження в точності моделі, дослідження забезпечує основу для майбутніх досліджень. Ідентифікація характеристик, що впливають на час редукції терму можуть вплинути на визначення більш ефективних стратегій оптимізації.

Ефективність використання ресурсів у редукції лямбда-термів. Здатність передбачити кількість кроків редукції для лямбда-термів за допомогою навчених моделей має практичне значення для автоматизації пошуку оптимальних стратегій редукції. Цей підхід не тільки дозволяє скоротити обчислювальні ресурси, але й час, необхідний для процесів нормалізації термів.

Застосування в складних програмних системах. Результати дослідження можуть бути застосовані в розробці складних програмних систем, де ефективність і швидкість мають першорядне значення. Розуміння поведінки редукції термів і оптимізації стратегії може бути особливо корисним у системах, які широко використовують парадигми функціонального програмування.

Внесок у теоретичну та прикладну інформатику. Окрім функціонального програмування, дослідження встановлює зв'язки між різними галузями, такими як теорія лямбда-числення, машинне навчання та розробка програмного забезпечення. Цей міждисциплінарний підхід збагачує як теоретичну базу, так і практичне застосування в інформатиці.

Підсумовуючи, хоча дослідження зберігає теоретичну спрямованість, його результати мають значні практичні застосування, зокрема у підвищенні ефективності та результативності компіляторів функціонального програмування та оптимізації продуктивності програмного забезпечення за допомогою передових стратегій редукції.

Апробація результатів дисертації. Основні теоретичні положення, висновки і пропозиції, які містяться в дисертації, обговорювалися та були схвалені на засіданнях кафедри теоретичної та прикладної інформатики факультету математики та інформатики Харківського національного університету імені В.Н. Каразіна. Ключові положення дослідження оприлюднені у доповідях на науково-технічних конференціях всеукраїнського та міжнародного рівнів:

– Міжнародній науково-технічній конференції ICTERI 2023 (Україна, м. Івано-Франківськ, Прикарпатський національний університет імені Василя Стефаника, 2023 р.).

– Міжнародній науково-технічній конференції «Information Technology and implementation» IT&I-2023 (Україна, м. Київ, Київський національний університет імені Тараса Шевченка, 2023 р.)

– Третій міжнародній науково-технічній конференції «International Workshop of IT-professionals on Artificial Intelligence» ProfIT AI 2023 (Канада, м. Ватерлоо, Університет Ватерлоо, 2023 р).

Публікації. Основні теоретичні положення і висновки дисертації викладені у 7 наукових працях, з яких 2 статті у наукових фахових виданнях, що входять до міжнародних наукометричних баз [1, 2] та 2 у наукових фахових виданнях України [3,4], та 3 тез наукових доповідей [5–7].

Структура та обсяг дисертації. Дисертаційна робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел і одного додатку. Загальний обсяг дисертації становить 174 сторінок: у тому числі анотації на 13 сторінках, зміст на 2 сторінках, основний текст на 143 сторінках, список використаних джерел із 105 найменувань на 11 сторінках та один додаток на 2 сторінках. Робота містить 16 таблиць, 30 рисунків, з яких 1 на окремій 1 сторінці.

РОЗДІЛ 1.
БЕЗТИПОВЕ ЛЯМБДА ЧИСЛЕННЯ

1.1. Функціональне програмування

В останні роки індустрія програмного забезпечення стала свідком значного зсуву в бік прийняття парадигми функціонального програмування. Ця тенденція, ретельно проаналізована в основоположних роботах, таких як [1–7], що відображає оцінку унікальних переваг, які пропонує функціональне програмування в процесах розробки програмного забезпечення. Функціональні мови програмування забезпечують основу для верифікації програм завдяки їх узгодженню з математичною логікою, формальними системами та передбачуваною поведінкою. Функціональні мови програмування слугують інструментами для перевірки програмного забезпечення. Ці переваги, підкреслені ключовими дослідниками в цій галузі [2, 3, 7], що включають підвищену простоту структури коду, покращену можливість тестування компонентів програмного забезпечення, покращену можливість повторного використання коду та масштабованість проектування програмних систем. Ці особливості разом сприяють розробці надійних та ефективних програмних рішень, що задовольняють постійно зростаючій складності сучасного програмного забезпечення [2, 7].

У фінансовому секторі функціональне програмування цінується за його сильний фокус на незмінності та чистих функціях, що може призвести до більш передбачуваних, відмовостійких систем. Такі мови, як Haskell і Scala, зазвичай використовуються для розробки фінансових моделей, систем управління ризиками та торгових платформ, які потребують високої точності та надійності [8–11].

Функціональні мови програмування застосовуються в різних областях та галузях завдяки їх здатності до управління складними системами, забезпечуючи при цьому високу надійність і зручність обслуговування [4–7]. Функціональне програмування відоме тим, що створює код, який легше тестувати, налагоджувати та підтримувати. Воно часто використовується в корпоративних середовищах, де

великі програмні системи вимагають високої стабільності, надійності та простоти обслуговування. Функціональні мови, як-от Haskell і Erlang, використовуються для розробки як невеликих інструментів, так і великомасштабних програм [7].

У фінансовому секторі функціональне програмування цінується за його сильний фокус на незмінності та чистих функціях, що може призвести до більш передбачуваних, відмовостійких систем. Такі мови, як Haskell і Scala, зазвичай використовуються для розробки фінансових моделей, систем управління ризиками та торгових платформ, які потребують високої точності та надійності [8–11].

Erlang, функціональна мова, розроблена Ericsson для використання в телекомунікаційних системах, є прикладом корисності функціонального програмування в цій галузі. Здатність обробляти численні одночасні дії робить її ідеальним для таких додатків, як сервери та бази даних, які потребують високої доступності та сталої продуктивності [12].

Методи функціонального програмування все частіше використовуються у веб-розробці для керування складним станом і побічними ефектами, які спостерігаються в сучасних веб-додатках. Такі мови, як Elixir (створена на віртуальній машині Erlang) і Scala, використовуються для створення масштабованих веб-сервісів, які можна відносно легко підтримувати. Функціональні концепції також поширені у фреймворках JavaScript, таких як React, де принципи функціонального програмування допомагають більш передбачувано керувати станом інтерфейсу користувача [13].

Функціональне програмування також підходить для завдань із інтенсивним використанням даних завдяки потужній підтримці незмінних структур і виразному синтаксису для обробки списків та інших колекцій. Такі мови, як Scala, особливо популярні в додатках для великих даних, головним чином завдяки їхній сумісності з Apache Spark, швидкою інфраструктурою для кластерних обчислень загального призначення [14].

Слід також зазначити, що функціональне програмування може спростити реалізацію алгоритмів, які використовуються в ШІ (штучному інтелекті) та

машинному навчанні. Математична чистота функціональних мов добре узгоджується з потребою алгоритмів у точності та продуктивності. Clojure, наприклад, використовується завдяки його потужним можливостям маніпулювання даними та перевагам у продуктивності, що випливають із JVM (Java Virtual Machine) [15].

Однією із частих областей застосування функціональних мов програмування є паралельне обчислення. Природна сумісність функціонального програмування з моделями одночасного та паралельного виконання робить його ідеальним для додатків, які потребують високого рівня паралелізму. Це пояснюється тим, що незмінність усуває проблеми, пов’язані зі спільним змінним станом, поширеним джерелом помилок у паралельних системах. Haskell і Erlang, наприклад, пропонують вбудовану підтримку паралелізму [7].

В сучасному світі пристроїв функціональне програмування може внести значний внесок у область IoT (Internet of Things), дозволяючи розробляти надійні, масштабовані та зручні в обслуговуванні системи, які можуть обробляти велику кількість з’єднань і одночасних завдань, таких як мережеві датчики та телекомунікаційне обладнання. Erlang і Elixir є відомими виборами в цій галузі завдяки їхній здатності обробляти багато з’єднань одночасно [16].

Функціональні мови програмування використовуються в різних сферах, де переваги чистих функцій, незмінності та експресивного паралелізму є особливо цінними. Ця зміна парадигми сприяє створенню більш надійного, керованого та стійкого до помилок коду, що робить його дуже корисним для сучасних складних систем у різних галузях.

Незважаючи на переваги парадигми функціонального програмування можливо виділити значні недоліки такого підходу. В роботі [3] визначені основні недоліки, які перешкоджають більш широкому застосуванню парадигми. З них можна виділити проблеми з неефективним використанням пам’яті та нижчою продуктивністю порівняно з імперативними парадигмами.

Крім того, функціональне програмування обмежене нестачею фреймворків та інструментів, яких більше в імперативних середовищах. Ця прогалина в наявних засобах розробки ще більше ускладняється малою спільнотою експертів і користувачів, що обмежує доступність і еволюцію даного підходу [2, 17, 18].

Нижче детально наведено ***проблеми функціональних*** мов програмування на вирішенні яких ***сфокусоване дане дослідження***:

I. ***Низька продуктивність обчислення*** – це є найбільш гострою проблемою функціонального програмування. Обчислювальний процес у функціональній парадигмі характеризується застосуванням трьох ключових механізмів: альфа-конверсії, бета-редукції та ета-конверсії [2, 17].

Кумулятивні витрати на обчислення програми, таким чином, є сукупністю витрат, пов’язаних з кожним із цих підпроцесів, разом із додатковими витратами, такими як розпізнавання редексу. Оцінка внеску кожного підпроцесу в загальну вартість обчислення визначається різними моделями для оцінки продуктивності обчислення функціональних програм.

II. ***Визначення стратегії редукції.*** Неоднозначність процесу редукції потребує розробки стратегій, які можуть ефективно подолати цю складність. Однак визначення ефективної стратегії редукції є тонкою проблемою. Як показано в статті [17], не існує універсальної оптимальної стратегії редукції, що ускладнює завдання оптимізації функціональних програм для кращої продуктивності.

Робота з існуючими функціональними мовами програмування може бути складною з точки зору оптимізації, тому для цього дослідження було обрано лямбда-числення як найпростіший варіант для представлення функціональних мов програмування загального застосування.

Лямбда-числення та функціональне програмування глибоко взаємопов’язані між собою, причому лямбда-числення є основоположною теоретичною основою для розробки та розуміння мов функціонального програмування [19].

Кажучи про спільну теоретичну основу, лямбда-числення, розроблене Алонзо Черчем у 1930-х роках, є формальною системою, призначеною для

дослідження визначення функції, застосування функції та рекурсії. Воно забезпечує теоретичну базу для опису функцій та їх обчислень. Ця система є суто функціональною, тобто вона розглядає функції як громадян першого класу, даний концепт є центральним для функціонального програмування [19, 20].

Мови функціонального програмування знаходяться під сильним впливом концепцій і структур, отриманих з лямбда-числення. Такі мови включають Haskell, Scheme, OCaml, Erlang та багато інших. У цих мовах функції можна передавати як аргументи, повертати з інших функцій і призначати змінним, що безпосередньо відображає те, як лямбда-числення розглядає функції як дані.

Незмінність. Спільною ключовою концепцією для лямбда числення та функціональних мов програмування є незмінність. У лямбда-численні після визначення функції її зв’язок вводу-виводу не змінюється, що відображає незмінність даних у функціональному програмуванні, де об’єкти даних не можна змінювати після їх створення [19, 20].

Функції першого порядку. Лямбда-числення допускає функції, які працюють над іншими функціями та повертають їх. Ця концепція є основною у функціональному програмуванні, уможливлюючи потужні методи, такі як відображення, зменшення та фільтрування, які є фундаментальними для обробки списків і масивів [19, 20].

Чистота функцій. Функції лямбда-числення є чистими функціями, тобто вони завжди дають однаковий результат для того самого входу без побічних ефектів. Функціональне програмування також наголошує на чистоті, щоб забезпечити передбачуваність і полегшити налагодження [19, 20].

Рекурсія. Лямбда-числення покладається на рекурсію для повторюваних або циклічних операцій, оскільки воно за своєю суттю не має циклічних конструкцій або ітераційних операторів. Це схоже у функціональному програмуванні, де рекурсія часто використовується замість циклів для повторюваних операцій. Крім того, лямбда-числення представило концепцію кодування Черча, яке представляє дані й оператори, використовуючи лише функції. Функціональні мови

програмування часто використовують подібні кодування та абстрактні представлення даних [19, 20].

Обчислення виразів. Стратегії оцінки, отримані з лямбда-числення, такі як звичайний порядок (Call by Name) і аплікативний порядок (Call by Value), мають безпосередній вплив на функціональне програмування [19, 20]. Наприклад, Haskell за замовчуванням використовує відкладене обчислення, відкладаючи обчислення до необхідного часу, тоді як такі мови, як ML і Scala, проводять обчислення аргументів в першу чергу.

Таким чином, лямбда-числення не тільки надихнуло багато фундаментальних принципів функціонального програмування, але й продовжує впливати на його еволюцію. Його абстрактний підхід до функцій і обчислень забезпечує багату теоретичну базу, яка підвищує глибину та можливості функціональних мов програмування. Саме тому подальші експерименти та дослідження були проведені в рамках роботи з лямбда-численням.

1.2. Лямбда-числення

Лямбда-числення – це проста нотація для функцій та застосувань. Основна ідея застосування функцій до аргументів (application) та створення функцій за допомогою абстракцій (abstraction). Синтаксис базового лямбда-числення є доволі прозорим, роблячи його елегантним рішенням з фокусом нотації на репрезентації функцій. Функції та аргументи постійно перекликаються один з одним. В результаті була отримана нонекзистенціальна теорія функцій як правил для обчислення, на противагу екзистенціальній теорії, що розглядає функції як набори упорядкованих пар. Незважаючи на простий синтаксис, виразність та гнучкість лямбда-числення робить його ефективним інструментом для логіки та математики [19–21].

Лямбда-числення є елегантною нотацією для роботи із застосуванням функцій до аргументів. Розглянемо приклад математичного поліному $x^3 - 5x + 8$ у якості прикладу. Чому дорівнює значення зазначеного виразу при аргументі $x = 3$? Обчислення цього є досить очевидним, необхідно просто замість x підставити

3, що призведе до отримання виразу $3^3 - 5 * 3 + 8$, що в подальшому обчислюється до значення 20. Представлення даного поліному у термінах лямбда-числення можливе завдяки використанню символу λ, та матиме вигляд:

$$x^3 - 5x + 8 \quad (1.1).$$

Оператор лямбда (λ) дозволяє абстрагуватися (to abstract) над змінною x. Інтуїтивно можна прочитати $(\lambda x.(x^3 - 5x + 8))a$ як вираз, що очікує на значення a для змінної x. Провівши таку заміну (як було у прикладі з числом 3), значення виразу стає $a^3 - 5a + 8$. Символ λ сам по собі не має значення; він просто зв'язує змінну x, що вказана після символу λ, та дозволяє відокремити цю змінну від інших змінних x поза лямбда виразом. Термінологічно лямбда-числення полягає в застосування (або аплікації) виразу до його аргументу та отримати значення такого застосування. Вираз (Ma) представляє собою застосування (application) функції M до аргументу a. Подовжуючи з даним прикладом ми отримуємо:

$$(\lambda x.(x^3 - 5x + 8))\ 3 \ \triangleright\ 3^3 - 5 * 3 + 8 = 27 - 15 + 8 = 20 \quad (1.2)$$

Перший крок обчислення – це заміна 3 замість усіх входжень x у виразі $x^3 - 5x + 8$, це є переходом від **терму абстракції** до іншого терму за допомогою операції підстановки. Решта рівностей – є простими обчисленнями з цілими числами. Вираз 1.2 показує центральний принцип лямбда-числення, що називається **β-редукцією** (або **β-*reduction*** в оригіналі) або **β-конверсія** (або **β-*conversion*** в оригіналі) [21–23]:

$$(\lambda x.M)\ N \ \triangleright\ M[x := N] \quad (1.3)$$

Розуміння того, що можливо редукувати, або скоротити ($\triangleright$) терм аплікації $(\lambda x.M)\ N$ терму абстракції (ліву частину виразу, $\lambda x.M$) до чогось (праву частину

виразу, *N*) простою підстановкою *N* замість входжень *x* в *M* (це те, що нотація '$M[x:=N]$' представляє собою). β-редукція або β-конверсія – це основа лямбда-числення.

1.2.1. Синтаксис лямбда числення

Оригінальний синтаксис лямбда-числення можна визначити за його алфавітом та використаними правилами, що показано у наступному визначенні.

Визначення. Алфавіт лямбда-числення складається з символів: "(", ")", ".", "λ" та нескінченної кількості змінних. Тоді клас лямбда-терму можна визначити наступним чином [23].

1) Кожна змінна – це лямбда-терм.

2) Якщо *M* та *N* це лямбда-терми, тоді $(M\ N)$ також лямбда-терм, що також називається термом аплікацією.

3) Якщо *M* це лямбда-терм та *x* – це змінна, тоді $(\lambda x.M)$ – це лямбда-терм, що також називається термом абстракцією.

Опущення символів дужок, як це притаманно формальним мовам із символами групування, також можливе, але за умови якщо це безпечно (тобто, коли їх можна повторно ввести лише одним способом) [21, 23]. Зіставлення більше ніж двох лямбда-термів є неможливим. Для уникнення перенавантаження виразів лямбда-термів береться правило асоціації зліва.

Правило (асоціації зліва). Згідно з цим правилом [21, 23], якщо є деякий терм $M_1\ M_2\ M_3\ \ldots\ M_n$ з опущеними дужками, то можна відновити усі дужки єдиним способом застосовуючи правило асоціації зліва, згідно з яким M_1 та M_2 можна об'єднати в $(M_1\ M_2)M_3\ \ldots\ M_n$, далі об'єднати з M_3 і отримати $((M_1\ M_2)M_3)\ \ldots\ M_n$. Що в підсумку веде до єдиного способу відновлення всіх відсутніх дужок: $(((M_1\ M_2)M_3)\ \ldots\ M_n)$.

Таким чином правило асоціації зліва дає можливість читання всіх послідовностей лямбда-термів довжина яких більша за 2 з опущеними дужками єдиним чином [21, 23].

1.2.2. Змінні, зв'язані та вільні

Функція символу 'λ' в термі абстракції ($\lambda x.M$) полягає в тому, що вона зв'язує появу змінної одразу після її появи в термі M. Таким чином 'λ' є аналогом універсального та екзистенціального кванторів $\forall$ та $\exists$ логіки першого порядку [21, 23]. Аналогічно можна визначити поняття вільної та зв'язаної змінної.

Визначення (вільної та зв'язаної змінних). Синтаксичні функції FV (вільна змінна, або free variable) та BV (зв'язана змінна, або bound variable) можна визначити за структурною індукцією лямбда термів [21, 23]:

1) $FV(x) = \{x\}, BV(x) = \emptyset$
2) $FV(M\ N) = FV(M) \cup FV(N), BV(x) = BV(M) \cup BV(N)$
3) $FV(\lambda x.M) = FV(M) - \{x\}, BV(\lambda x.M) = BV(M)\ \cup\ \{x\}$

Якщо $FM(M)\ =\ \emptyset$, тоді M називають комбінатором.

Це в загальному підтверджує факт того, що 'λ' зв'язує змінні (тобто робить їх не вільними). Це є типовим для інших предметів із концептом логіки першого порядку, де потрібно зважати на можливі синтаксичні складності при виконанні операції підстановки [22]. Невимушену підстановку можна захистити узгодивши факт зацікавленості не в самих термах, а в певному класі еквівалентності термів. Тепер стає можливим визначити підстановку й конвенцію для уникнення складнощів перестановки.

Визначення (підстановка). Визначити підстановку можна наступним чином $M[x :=\ N]$ як підстановку N у всі вільні входження x в M. Точне визначення [23]

за рекурсією на множині лямбда-термів полягає в наступному: для всіх термів *A*, *B* та *M*, та для всіх змінних x та y визначаються наступні правила:

1) $x[x:=M] \equiv M$

2) $y[x:=M] \equiv y$ (*y* відрізняється від *x*)

3) $(A\ B)[x:=M] \equiv A[x:=M]\ B[x:=M]$

4) $(\lambda x.A)[x:=M] \equiv \lambda x.A$

5) $(\lambda y.A)[x:=M] \equiv \lambda y.(A[x:=M])$ (*y* відрізняється від *x*)

Положення (1) визначення підставки вказує, що у разі заміни *M* на *x* і терм в якому відбувається заміна є *x*, тоді результатом буде просто *M*. Положення(2) вказує на те, що нічого не відбувається, коли терм представляє змінна, відмінна від *x*, але необхідно замінити *x*. Положення (3) зазначає, що підставка безумовно розподіляється по терму. Положення (4) і (5) стосуються абстрактних термінів і паралельних їм положень (2) і (1): якщо зв'язана змінна *z* абстрактного терму $(\lambda z.A)$ ідентична змінній *x*, для якої необхідно зробити заміну, тоді жодна підстановка не виконується (тобто підстановка зупиняється). Це узгоджується з наміром, що $M[x:=N]$ має позначати заміну *N* для вільних входжень *x* в *M*. Якщо *M* є абстрактним термом $(\lambda x.A)$, зв'язаною змінною якого є *x*, тоді *x* не входить вільно в *M*, тому нічого не потрібно робити. Це пояснює положення (4). Положення (5) вказує та те, що за умови якщо зв'язана змінна абстрактного терму відрізняється від *x*, то принаймні *x* має «шанс» вільно з'явитися в абстрактному термі, і заміна продовжується в тілі терму абстракції [21].

Визначення (заміна зв'язаних змінних, α-конвертованості) полягає, що терм *N*, отриманий з терму *M* згідно з заміною зв'язаної змінної якщо, приблизно, будь-який абстрактний терм $(\lambda x.A)$ всередині *M* був замінений на $(\lambda y.(A[x:=y]))$. Тоді терми *M* та *N* є α-конвертовані, якщо є послідовність замін зв'язаних змінних, що починається на термі *M* та закінчується на *N* [21, 23].

Аксіома β-конвертованості (із застереженням про заборону захоплення) полягає в $(\lambda x.M)\ N\ \rhd\ M[x :=\ N]$, зазначає, що жодна змінна, що була вільною в N не стає зв'язаною в M після підстановки [21, 23].

Існує необхідність тримати вільні змінні такими, навіть за умови, коли виникає можливість зв'язування змінної через підстановку необхідно виконати декілька α-перетворень, для уникнення зв'язування вільних змінних. Якщо враховувати це далі операції в лямбда-численні не призводять до синтаксичних складнощів. Так, наприклад, неможливо застосувати функцію λx. λy. $(3y\ (x-15))$ для аргументу $3y$ адже підстановка $3y$ для x, та y в $3y$ буде зв'язано оператором зв'язування змінної λy. Така підстановка може призвести до функції відмінної від запланованої. Але, якщо застосовуючи правило α-конвертованості до функції $\lambda x.\lambda y.(3y\ (x-15))$ отримаємо $\lambda x.\lambda b.(3b\ (x-15))$, то застосування аргументу $2y$ до цієї функції стає можливим. Тому, неможливо використати правило β-конвертованості у виразі $(\lambda x.\lambda y.(3y\ (x-15)))5y\ \rhd\ \lambda y.(15y(x-15))$, але є можливим використати правило β-конвертованості у виправленому виразі $(\lambda x.\lambda y.(3y(x-15)))2y\ \rhd\ \lambda z.(15y(x-15))$.

Цей приклад допомагає побачити, чому застереження до β-конвертованості є важливим. Застереження нічим не відрізняється від того, що використовується у формулюванні аксіоми обчислення предикатів, а саме $\forall x\phi\ \rightarrow\ \phi_x^\tau$, не надає жодної змінної, що є вільною в термі τ до того як виконана заміна, але стає зв'язаною після [24].

Синтаксис λ-числення є доволі гнучким. Присутня можливість формулювання будь-яких термів, навіть самозастосування, такого як $(x\ x)$. Такі терми видаються сумнівними, можна припустити, що використання таких термів може призвести до непослідовності, проте в будь-якому випадку можна знайти інструмент, що дозволяє позбутися таких термів. Якщо розглядати функції та множини впорядкованих пар певного типу, то x в $(x\ x)$ буде функцією множини впорядкованих пар, який містить як елемент пару (x, y), першим елементом якої

буде сам x. Але жодна множина не може вмістити себе таким чином, аби не порушити аксіому регулярності. Виходить, з точки зору теорії множин такі терми є явно сумнівними. Але, в реальності, такі терми не призводять до неузгодженості та служать корисними у контексті лямбда-числення. Крім того, заборона таких термів, як це зроблено в теорії типів, не залишається без наслідків, як то втрата частини виразності нетипового лямбда-числення [21, 25].

1.2.3. Комбінатори

Як було зазначено раніше комбінатори в лямбда-численні – це терми без вільних змінних. Інакше кажучи їх можна визначити як повністю визначені операції, оскільки в них немає вільних змінних. Існує певна кількість корисних комбінаторів лямбда-числення. Табл. 1.1 зазначає деякі з таких комбінаторів. Існує нескінченна кількість комбінаторів, проте наведені в табл. 1.1 комбінатори мають відносно короткі визначення й показали свою значимість в контексті лямбда числення [21].

Таблиця 1.1.

Основні терми комбінатори лямбда-числення.

Символ	**Визначення та застосування**
S	$\lambda x.\lambda y.\lambda z.\,(x\ z\ (y\ z))$ Враховуючи, що $(x\ z\ (y\ z))$ є термом $((x\ z)\ (y\ z))$ аплікації $(x\ z)$ до $(y\ z)$. Тоді *S* – це оператор заміни та застосування: *z* втручається між *x* та *y*: замість застосування *x* до *y* застосовується *(x z)* до *(y z)* [23].
K	$\lambda x.\lambda y.x$, де значення $(K\ M)$ є функцією константи, для будь якого значення аргументу є *M*.
I	$\lambda x.x$ – це функція ідентичності.
B	$\lambda x.\lambda y.\lambda z\ (x\ (y\ z))$ Виклик $(x\ y\ z)$ що є не що інше як $((x\ y)\ z)$, тому цей комбінатор не є тривіальною функцією ідентичності.
C	$\lambda x.\lambda y\ \lambda z.xzy$ – оператор заміни порядку аргументів.
T	$\lambda x.\lambda y.x$ – ідентичний до оператору K, також грає роль значення

	Правди при використанні логіки в лямбда-численні.
F	$\lambda x.\lambda y.y$ – це оператор, що виконує роль Брехні при запровадженні логіки в лямбда-численні.
ω	$\lambda x.(x\ x)$ – це комбінатор самозастосування.
Ω	$(\omega\ \omega)$ – це комбінатор самозастосування до самозатостосування, що редукується в самого себе.
Y	$\lambda y.((\lambda x.(y\ (x\ x)))\ (\lambda x.(y\ (x\ x))))$ – Парадоксальний комбінатор Каррі. Для кожного лямбда-терму X виконується: $Y\ X \rhd ((\lambda x.(X\ (x\ x)))\ (\lambda x.(X\ (x\ x)))) \rhd$ $X\ ((\lambda x.(X\ (x\ x)))\ (\lambda x.(X\ (x\ x))))$. Перший крок в редукції $(Y\ X)$ призводить до терму аплікації $((\lambda x.(X\ (x\ x)))\ (\lambda x.(X\ (x\ x))))$, що повторюється в третьому кроці. Тому терм Y має властивість того що $(X\ Y)$ та $(X\ (Y\ X))$ редукують до одного терму.
Θ	$(\lambda x.\lambda y.y\ (x\ x\ y))\ (\lambda x.\lambda y.y\ (x\ x\ y))$ – це комбінатор фіксованої точки Тьюринга. Для кожного лямбда-терму *X*, *(Θ X)* редукується до *X (Θ X)*.

Нижче наведено табл. 1.2 [21–23] із позначеннями, що використовувалися в цій роботі для дослідження лямбда-числення .

Таблиця 1.2.

Позначення прийняті в дослідження для позначення виразів в лямбда-численні.

Позначення	**Значення**
M N	Застосування функції *M* до аргументу *N*. Зазвичай дужки використовуються для розділення функції від аргументів, як це робиться в *M(N*). Проте, в лямбда-численні дужки є символом групування. Тому в лямбда-численні функція та аргумент пишуться поруч один з одним.
P Q R	Застосування *(PQ)* – терму аплікації до аргументу *R*. Використання правила лівого розкриття дозволяє однозначно визначити порядок застосувань в такому записі терму, що відповідає терму $((P\ Q)\ R)$.
λx. M	Символ λ зв'язує змінну *x* в тілі терму *M*. Такий синтаксис не є офіційним, проте він відображає підхід, що використовувався в

	ранніх роботах логіки, де символ '.' дозволяє виділити формулу ϕ у виразі $\forall x.\phi$, як та що знаходиться під сферою впливу $\forall x$. Згідно з офіційною документацією запис терма абстракції має такий вид $\lambda x[M]$ або із додаванням дужок $(\lambda x[M])$. Було обрано запис терма абстракції через точку, адже це спрощує зображення термів.
M[x := A]	Це лямбда-терм, що був отриманий внаслідок півставки терму *A* на місце всіх вільних входжень *x* в терм *M*. Також можливі інші нотації, наприклад: $M[x/A], M[A/x], M_x^A, M_A^x, [x/A]M$. Вибір між ними є суто формальним рішенням. Було обрано запис $M[x := A]$, адже він представляє присвоєння значення змінній в деяких мовах програмування.
$M \equiv N$	Означає, що лямбда-терм *M* є ідентичний лямбда-терму *N*. Такий запис не є офіційною частиною документації лямбда-числення, проте дозволяє оцінювати відношення між лямбда-термами, що є теоретичною частиною лямбда-числення. Такий запис дозволяє визначати логічну рівність між термами, навіть за умови наявності відмінних змінних, тобто $\lambda x.x \equiv \lambda y.y$.

1.3. Історія лямбда-числення

Лямбда-числення походить з вивчення функція у якості правил. Основоположні складові лямбда-числення вже можуть бути знайдені вже в новаторській роботі Фреге (Frege, 1893) [26]. Фреге зазначив, що при вивченні функцій достатньо зосередитися лише на унарних функціях, тих що приймають рівно один аргумент на вхід. Процедура переходу від функції з однією змінною до функцій з декількома може бути проведена завдяки послідовному застосуванню декількох операцій абстракції, що видають на виході еквівалентну унарну операцію. Дана операція називається карування (***curring***). Або, можливо, з історичної точки зору її було б правильніше назвати фрегуванням (***fregeing)***. В 1920-х математику за ім'ям Мозес Шенфінкель вдалося просунути дану концепцію ще далі, завдяки його дослідженню так званих комбінаторів (***combinators***) [21]. Як це було прийнято на початку вивчення предмету, Шенфінкеля зацікавили трансформації, які можна спостерігати у формальній логіці, і його робота над комбінаторами була спрямована на внесення вкладу в основи формальної логіки.

Подібно до процесу редукції, що можна побачити в препозиційній логіці завдяки роботі Шеффера, Шенфінкель досяг видатних результатів, тим фактом, що всі функції (з точки зору всіх трансформацій) можуть бути представлені використанням всього двох комбінаторів *K* та *S*.

Це було показано в теоремі Шенфінкеля, згідно з якою для будь-якого терму *M*, що побудований, завдяки двом комбінаторам *K* та *S* та змінній *x,* існує терм *F* (побудований тільки з комбінаторів *K* та *S*) такий, що можна вивести $Fx = M$.

Дана теорема доводиться послідовно, нехай існує такий алгоритм, що для даного *M* може побудувати необхідний *F*. Черч назвав даний алгоритм *F* '$\lambda x.M$' (Черч, 1932) [23]. З даної точки зору β-правило є виправданим, якщо '$\lambda x.M$' є функцією *F*, що задовольняє умові $Fx = M$, тоді $\lambda x.Mx$ має спрощуватися до *M*. Це всього на всього конкретний випадок більш загального принципу, що для всіх *N*, $(\lambda x.M)\ N$ має перетворюватися на $M[x:=N]$.

Незважаючи на те, що сьогодні наявна більш чітко визначена система абстракцій та переписування, в свої ранні роки лямбда-числення та комбінаторна логіка були тісно пов'язані з дослідженням основ математики. У руках Каррі, Черча, Кліні та Россера (одних із піонерів цієї галузі) основна увага була зосереджена на визначенні математичних об'єктів та виконанні логічних обґрунтувань у цих нових системах. В подальшому виявилося, що дані спроби в так званому заключному лямбда-численні та комбінаторній логіці були непослідовними та суперечливим. В свою чергу, Каррі ізолював себе та якісно пропрацював суперечливості, що в результаті стало відомим як парадокс Каррі [21].

Лямбда-числення заслуговує окреме місце в історії логіки, як джерело його перших нерозв'язних проблем. Задача: Для даних термів *M* та *N* визначити чи $M = N$. (Теорія рівності щодо лямбда-термів ще не була визначена; визначення було надано пізніше.) Ця проблема також була показана як нерозвязна.

Іншою ранньої проблемою лямбда-числення було питання, чи воно в цілому є не суперечливим (consistent). Під суперечливістю. мається на увазі, що всі терми рівні: можливою є редукція будь-якого лямбда-терму *M* до будь-якого іншого

лямбда-терму *N*. Те що це не так є результатом раннього лямбда числення. Початкові результати, що з'явилися були про те, що деякі терми, як от *K* та *S*, не можуть бути приведені один до одного. Пізніше був отриманий набагато більш значущий результат: теорема Черча-Россера, що дала змогу внести ясність в розуміння β-конверсії і може бути застосована для доведення незвідності між собою цілих класів лямбда-термів [21].

Початково лямбда-числення можна було назвати деяким видом формалізму, до 1960-х, коли останню семантику було затверджено. У той же час зв'язок лямбда-числення з мовами програмування також був з'ясованим та окреслений. До того моменту всі моделі лямбда-числення були побудовані навколо 'синтаксису', в стилі Генкіна, та складалися з еквівалентності класів лямбда-термів (для зручності нотації). Завдяки застосуванням лямбда-числення у семантиці природних мов у роботах Монтегю та інших лінгвістів, відомості про даний предмет розійшлися світом. З тих часів лямбда-числення заслуговує почесне місце у таких областях як математична логіка, комп'ютерні науки , лінгвістика та багато інших [21].

1.4. Редукція

Існують різноманітні визначення скорочення лямбда-термів, але основним є β-редукція. До цього вже було дано визначення β-редукції із використанням символу $\rhd$. Проте варто надати більш специфічне визначення.

Визначення (однокрокова β-редукція). Однокрокова β-редукція (позначається як $\rhd_{\beta,1}$) для лямбда-термів *A* та *B*, що відрізняються одним кроком редукції можна записати як $A \rhd_{\beta,1} B$, якщо існує підтерм *C* терму *A*, змінна *x* та лямбда терми *M* та *N* такі що $C \equiv (\lambda x.M)\ N$. Та терм *B* є термом *A* за виключенням появи терму *C* в *A*, що замінений $M[x := N]$ [21, 23].

Далі наведено декілька прикладів β-редукції.

1) Змінна *x* не може бути β-редукована у будь-що, оскільки це просто змінна, що не є аплікацією.

2) $(\lambda x.x)a \ \triangleright_{\beta,1} \ a$.

3) Якщо *x* та *y* є відмінними змінними, то $(\lambda x.y)a \ \triangleright_{\beta,1} \ y$.

4) Лямбда-терм $(\lambda x.((\lambda y.x\ y)\ a)\ )\ b$ можна β-редукувати в 2 різних лямбда-терми, проте ще один крок редукції все одно веде до одного і того самого терму:

$\left(\lambda x.\left((\lambda y.x\ y)a\right)\right) b \ \triangleright_{\beta,1} \ (\lambda y.b\ y)\ a \ \triangleright_{\beta,1} \ b\ a$

$\left(\lambda x.\left((\lambda y.x\ y)a\right)\right) b \ \triangleright_{\beta,1} \ (\lambda x.x\ a)\ b \ \triangleright_{\beta,1} \ b\ a$

Можливо визначити різні похідні поняття від запису β-редукції одного кроку $\triangleright_{\beta,1}$, що є притаманним для будь-яких бінарних відношень.

Визначення. Так можна визначати послідовність β-редукції від терму *A* до терму *B* як скінченну послідовність $s_1, \dots, s_n$, що починається з *A* і закінчується *B*, за суміжною умовою для (s_k, s_{k+1}) задовольняють умови $s_k \ \triangleright_{\beta,1} \ s_{k+1}$.

Або загально кажучи, будь-яка послідовність *s* – скінченна або нескінченна послідовність β-редукції, що починається з лямбда-терму *A* та задовольняє умови $s_k \ \triangleright_{\beta,1} \ s_{k+1}$ для (s_k, s_{k+1}) [21, 23].

1) Не існує послідовності β-редукцій для змінної x.

2) Для терму $((\lambda x.x)\ a)$ існує єдина послідновність β-редукцій, що є $((\lambda x.x)\ a), a$. Оскільки в кінці послідовності стоїть змінна, то цю послідовність неможливо продовжити згідно з першим прикладом.

3) Комбінатор Ω має цікаву властивість: $\Omega \ \triangleright_{\beta,1} \ \Omega$. Тому кожний терм Ω є частиною нескінченної послідовності β-редукцій терму Ω.

4) Цікава властивість у терму $(K\ a\ \Omega)$, що є початком нескінченної кількості варіантів β-редукцій:

• $(K\ a\ \Omega) \ \triangleright_{\beta,1} \ a$

• $(K\ a\ \Omega) \ \triangleright_{\beta,1} \ (K\ a\ \Omega) \ \triangleright_{\beta,1} \ a$

• $(K\ a\ \Omega) \ \triangleright_{\beta,1} \ (K\ a\ \Omega) \ \triangleright_{\beta,1} \ (K\ a\ \Omega) \ \triangleright_{\beta,1} \ a$

• $(K\ a\ \Omega)\ \triangleright_{\beta,1}\ (K\ a\ \Omega)\ \triangleright_{\beta,1} \ldots$

Якщо a – це змінна, то можна побачити, що всі послідовності скорочень терму $(K\ a\ \Omega)$ починаються з $(K\ a\ \Omega)$, та закінчуються з a.

Визначення (β-редексу). Також можна визначити β-редекс лямбда-терму *M* як поява деякого субтерму *M* в формі $(\lambda x.P)\ Q$. Простіше кажучи β-редекс є кандидатом на β-редукцію в деякому лямбда-термі. Кажуть, що терм перебуває в β-нормальній формі, якщо він не має β-редексів [21, 23].

Чи може терм мати декілька β-нормальних форм? Літерально кажучи “так”, проте по суті “ні”. Тобто якщо *M* та *M′* є β-нормальними формами деякого терму, тоді *M* є α-конвертованим до *M′*. Тому β-нормальна форма є унікально аж до зміни зв'язаних змінних.

Було розглянуто одно-крокову β-редукцію, але можна поєднати декілька кроків β-редукцій в один шляхом транзитивного замикання відношення $\triangleright_{\beta,1}$.

Визначення (β-редукованості). Для лямбда-термів *A* та *B* можна сказати, що A β-редукується до B, що записується як $A\ \triangleright_{\beta,1}\ B$, якщо $A\ \equiv\ B$ або якщо існує скінченна послідовність β-редукцій від *A* до *B* [21, 23].

Визначення (наявності β-нормальної форми). Терм *M* має β-нормальну форму якщо існує терм *N* такий, що *N* є в β-нормальній формі $M\ \triangleright_{\beta,1}\ N$.

Як вже було визначено, редуктивність – це односторонній зв’язок, тому в загальному не вірно якщо $A\ \triangleright_{\beta,1}\ B$, то $B\ \triangleright_{\beta,1}\ A$. Проте, залежно від мети, можливо визначити еквівалентність термів *A* та *B* якщо *A* редукується до *B* чи *B* редукується до *A*. Це означає про можливість розгляду відношення $\triangleright_{\beta,1}$ як рефлексивне, симетричне та транзитивне [21, 23].

Визначення. Для лямбда-термів A та B, кажуть що $A =_\beta B$, якщо $A \equiv B$ чи існує послідовність $s_1, \dots, s_n$, що починається з A та закінчується з B та для послідовних термів (s_k, s_{k+1}) виконується одна з умов $s_k \rhd_{\beta,1} s_{k+1}$ чи $s_{k+1} \rhd_{\beta,1} s_k$.

1.4.1. Стратегії редукції

Терм називається β-нормальною формою, якщо в його тілі відсутні β-редекси, тобто підтерми форми $(\lambda x.M)\ N$. Терм має β-нормальну форму, якщо його можна звести до терму в β-нормальній формі. Має бути інтуїтивно зрозуміло, що якщо терм має β-нормальну форму, то можливо знайти її шляхом вичерпної редукції всіх β-редексів терму, потім вичерпної редукції всіх β-редексів всіх наступних термів і так далі. Сказати, що лямбда-терм має β-нормальну форму, означає сказати, що цей сліпий пошук можливо закінчити [21, 23].

Сліпий пошук β-нормальних форм термів очевидно не є ефективним, через потребу у повному скороченні всіх β-редексів. Необхідна деяка стратегія редукції, бажано обчислювана, для знаходження β-нормальної форми. Проблема полягає в тому, щоб ефективно вирішити, чи існує певні підмножини β-редексів терму, які слід скоротити, а які можуть бути пропущені.

Отже, визначати стратегію β-редукції можна як функцію де вхідними даними є усі лямбда-терми, значення яких на термі M не в β-нормальній формі, є редексним підтермом M, і значення на всіх термах M в β-нормальній формі є просто M [21, 23].

Простіше кажучи, стратегія β-редукції вибирає множину β-редексів які необхідно скоротити. Можна представити стратегію S як відношення $\rhd_S$ на лямбда-термах, з розумінням того, що $M \rhd_S N$ за умови, що N отримано з M за один крок, дотримуючись стратегії S. Якщо розглядати як відношення, стратегії редукції представляють субвіношення $\rhd_{\beta,1}$.

Визначення. Стратегією редукції називають алгоритм, який приймає на вхід терм, що не є нормальною формою, а повертає один з його редексів для редукування. Стратегія редукції – це специфічний методичний підхід, який

використовується в лямбда-численні для вибору та застосування правил редукції до лямбда-виразів. Стратегія визначає, який редекс лямбда-терму має бути редукований на кожному кроці процесу обчислення.

Стратегії β-редукції може мати, або не мати властивість того, що дотримання цієї стратегії зрештою забезпечує досягнення β-нормальної форми, якщо така існує [21, 23]. Вибір стратегії редукції може вплинути не лише на ефективність і результат обчислювального процесу, але й на те, чи завершиться процес або досягне нормальної форми [23].

Тому визначати стратегію β-редукції *S* як нормалізуючу якщо для всіх лямбда-термів *M*, якщо *M* має β-нормальну форму *N*, тоді послідовність $M, S(M), S(S(M)), \ldots$ закінчується на *N*. Деякі стратегії редукції є нормалізуючими, а інші – ні [21, 23].

Стратегії редукції у функціональному програмуванні визначають як і коли обчислюються аргументи функції та як виконується застосування функції. Ці стратегії є ключовими для розуміння операційної семантики мов програмування. Серед найвідоміших стратегій – Call by Name (CBN), Call by Value (CBV), Leftmost Outermost (LO) і Rightmost Innermost (RI) [23, 27–29]. Кожна має своє походження, визначення, алгоритми, переваги, недоліки та випадки використання.

Call by Name(CBN) [23, 27–29].

– **Походження та визначення**: виклик за іменем – це стратегія редукції, де аргументи функції не оцінюються перед викликом функції. Натомість функція застосовується безпосередньо до виразів аргументів без оцінки. Щоразу, коли потрібен аргумент, вираз обчислюється .

– **Алгоритм**: у CBN кожне входження параметра в тіло функції замінюється фактичним виразом аргументу. Якщо аргумент складний, він може бути переоцінений кілька разів, що призведе до потенційної неефективності.

– **Переваги**: може бути більш ефективною, коли аргументи не використовуються в тілі функції, оскільки уникають непотрібних оцінок. Стратегія також підтримує створення керуючих структур і операторів як звичайних функцій.

– **Недоліки**: численні оцінки одного виразу можуть призвести до неефективності, особливо якщо вираз інтенсивно обчислюється. Це також може призвести до неприпинення, навіть якщо існує порядок припинення редукції.

– **Використання**: CBN зазвичай використовується в мовах і сценаріях, де бажана відкладена оцінка, і дуже важливо відкладати обчислення до абсолютної необхідності.

Call by Value (CBV) [23, 27–29].

– **Походження та визначення**: виклик за значенням є найпоширенішою стратегією редукції, коли аргументи функції повністю оцінюються перед викликом функції. Результат цієї оцінки потім передається у функцію.

– **Алгоритм**: у CBV кожен вираз аргументу обчислюється один раз, а отримане значення прив’язується до відповідного параметра в тілі функції.

– **Переваги**: це дозволяє уникнути потенційної неефективності багаторазового обчислення виразу. Даний підхід є передбачуваним і простим, що робить його типовим у багатьох мовах програмування.

– **Недоліки**: може бути неефективним, якщо аргументи складні та не використовуються в тілі функції, оскільки обчислюються вирази, які ніколи не знадобляться.

– **Використання**: CBV широко використовується в імперативних мовах програмування та сценаріях, де необхідне суворе оцінювання або де накладні витрати на потенційні численні оцінки занадто дорогі.

Leftmost Outermost (LO) [23, 27–29].

– **Походження та визначення**: LO – це стратегія, яка часто асоціюється з ледачим оцінюванням у функціональних мовах. Вона завжди вибирає для оцінки крайній лівий, крайній зведений редекс.

– **Алгоритм**: стратегія проходить по дереву виразу, завжди вибираючи крайній лівий крайній редекс для редукції. Це часто відповідає «зовнішньому» виклику функції у вкладеному виразі.

– **Переваги**: LO гарантує знаходження нормальної форми, якщо вона існує, що робить її вигідним для обчислень, де не потрібно оцінювати всі аргументи. Вона добре узгоджується з ледачим обчисленням, уникаючи непотрібних обчислень.

– **Недоліки**: у деяких випадках дана стратегія може виконувати більше операцій редукції, ніж потрібно, порівняно з іншими стратегіями. Ця стратегія також може бути менш інтуїтивно зрозумілою для тих, хто знайомий з строгими парадигмами оцінювання.

– **Використання**: зазвичай використовується в мовах, які підтримують відкладене обчислення, таких як Haskell, що допускає такі конструкції, як нескінченні структури даних.

Rightmost Innermost (RI) [23, 27–29].

– **Походження та визначення**: RI – це стратегія, яка обирає крайній правий внутрішній редекс для скорочення, що є подібним до пошуку в глибину в дереві виразів починаючи справа.

– **Алгоритм**: стратегія обходить дерево виразів, щоб знайти найбільш глибоко вкладений редекс, починаючи з правого боку. Після виявлення цей редекс редукується.

– **Переваги**: у певних випадках, особливо з конкретними арифметичними обчисленнями або коли всі аргументи потрібно врешті-решт оцінити, RI може бути ефективнішим, ніж інші стратегії.

– **Недоліки**: RI не гарантує знаходження нормальної форми, якщо вона існує, і може призвести до непотрібних обчислень, якщо деякі аргументи не використовуються в тілі функції.

– **Використання**: це менш поширена стратегія, але її можна використовувати в певних сценаріях, де відомо, що її шаблон оцінки є корисним.

А також існують стратегії ***Leftmost Innermost (LI)*** та ***Rightmost Outermost (RO)***, що є відповідними модифікаціями LO та RI стратегій відповідно, проте не знайшли такого великого поширення. Згідно з визначенням LI є такою стратегією, що на кожному кроці крайній лівий із внутрішніх редексів редукується, де внутрішній редекс є редексом, який не містить жодних редексів. Та RO є такою стратегією, що на кожному кроці крайній правий із крайніх редексів скорочується, де крайній редекс є редексом, який не міститься в жодному редексі.

Таким чином, CBN і CBV зосереджені на тому аби оцінювати аргументи при застосуванні функцій, причому CBN є ледачою стратегією на відміну від CBV. На відміну від цих стратегій, LO та RI стосуються вибору, яку частину виразу оцінити першою, причому LO надає перевагу крайнім лівим зовнішнім редексам, а RI – крайнім правим внутрішнім. Вибір між цими стратегіями залежить від конкретних вимог і характеристик завдання програмування, а також від природи мови, що використовується. Кожна з них має свої переваги та компроміси, що робить їх придатними для різних сценаріїв у величезному просторі функціонального програмування.

Теорема (Барендрегта про неіснування оптимальної стратегії редукції): У безтиповому лямбда-численні не існує єдиної стратегії редукції, яка була б оптимальною для всіх лямбда-термів. Оптимальна стратегія редукції – це така стратегія, що мінімізує кількість кроків для досягнення нормальної форми для будь-якого лямбда-терму, за умови існування нормальної форми [23].

Пояснення:

• Стратегія редукції в лямбда-численні визначає порядок і спосіб, у який виконуються редукції (зокрема бета-редукції) лямбда-термів.

• Стратегія вважається оптимальною, якщо для кожного лямбда-члена, який має нормальну форму, вона знаходить нормальну форму, використовуючи найменшу можливу кількість кроків β-редукції.

• Теорема Барендрегта стверджує, що такої універсально оптимальної стратегії не існує. Це пов’язано з внутрішньою складністю та різноманітністю лямбда-термів, де різні терми можуть вимагати різних підходів для ефективного скорочення.

• Доказ передбачає демонстрацію того, що для будь-якої запропонованої оптимальної стратегії завжди можна побудувати конкретний лямбда-терм, де альтернативна стратегія досягає нормальної форми за меншу кількість кроків.

1.4.2.Методи підвищення ефективності процесу редукції

Оптимізація стратегій редукції в лямбда-численні – це складна тема, яка перетинає теорію інформатики, реалізацію мови програмування та обчислювальну логіку [30–34]. Мета полягає в тому, щоб підвищити ефективність і передбачуваність обчислень, що виконуються в цій абстрактній структурі. Далі буде представлений опис деяких найсучасніших методів оптимізації стратегій редукції, які були досліджені та розроблені в останні роки.

I. Графова редукція – ефективний метод оптимізації лямбда-числення, що замість дублювання термів, як у бета-редукції, використовує графову структуру для повторного використання загальних підвиразів [32, 33]. Це робить процес швидшим та ефективнішим, і застосовується, наприклад, у мові Haskell [34].

II. Оптимальні стратегії редукції, таких як редукування Леві, зосереджений на мінімізації кроків та уникненні дублювання термів. Хоча ця стратегія теоретично оптимальна, її реалізація складна і потребує значних обчислень [31].

III. Лінива оцінка, або виклик за потребою, відкладає обчислення виразів до моменту, коли їхні значення необхідні, зменшуючи кількість редукцій і уникаючи оцінки термінів, які не використовуються. Цей підхід, подібний до стратегії CBN, широко застосовується в Haskell і суттєво покращує продуктивність функціональних програм, особливо з складними або нескінченними структурами даних [34, 35].

IV. Суперкомпіляція – це метод програмної трансформації, який, аналізуючи поведінку програми під час виконання, усуває непотрібні обчислення та скорочує терміни, перетворюючи програму на більш ефективну версію. Цей метод можна адаптувати для оптимізації виразів лямбда-числення, особливо у функціональному програмуванні [36].

V. Редукція за типом. У типізованих системах лямбда-числення типи можуть керувати процесом редукції. Використовуючи інформацію про тип, можна виконати певну оптимізацію, наприклад, зменшити лише ті вирази, які призводять до правильного типу результату [37]. Це може запобігти безрезультатним скороченням і віддати пріоритет тим, хто з більшою вірогідністю сприятиме остаточному обчисленню.

VI. Паралельна редукція. Стратегії паралельної редукції дозволяють одночасно скорочувати кілька редексів, що прискорює обчислення на багатоядерних процесорах. Методи, як явна заміна, допомагають підтримувати узгодженість замін у паралельних частинах терму [38].

VII. Техніки машинного навчання. Останнім часом зростає інтерес до використання машинного навчання для оптимізації компіляторів та інтерпретаторів [39]. Це дозволяє прогнозувати ефективні шляхи редукції в лямбда-численні, навчаючи моделі на історичних даних редукції. Хоча цей підхід експериментальний, він обіцяє нові можливості на перетині машинного навчання та теоретичних обчислень.

VIII. Квантова редукція. З появою квантових обчислень досліджуються підходи до редукції в лямбда-численні з використанням квантового паралелізму. Це передбачає кодування лямбда-виразів у квантових станах і застосування квантових воріт для скорочень, що може забезпечити експоненційне прискорення порівняно з класичними методами [40].

XI. Графові структури. У теорії обчислень графове подання лямбда-термів дозволяє візуалізувати та обробляти функціональні вирази. Нещодавно дослідження зосередилося на інтеграції імовірнісних часових графів, щоб

моделювати ймовірність і час переходів між термінами. Це дозволяє аналізувати поведінку лямбда-термів з урахуванням стохастичних елементів і часових аспектів, що відкриває можливості для оптимізації обчислювальних стратегій у функціональних мовах.

1.4.3. Лямбда-числення як інструмент верифікації програм та паралельного програмування

У цьому підрозділі розглядаються теоретичні та практичні зв’язки між лямбда-численням, паралельним програмуванням і верифікацією програм. Лямбда-числення служить основою для міркувань про паралельні програми, і представляє формальні інструменти перевірки, які забезпечують правильність паралельного виконання. Ці факти підтверджують те, що лямбда-числення та його розширення забезпечують надійну структуру як для оптимізації паралельних програм, так і для перевірки їх правильності.

Лямбда-числення за своєю суттю добре підходить для паралельного програмування через його акцент на чистих функціях – функціях без побічних ефектів – які піддаються паралельному виконанню. У функціональному програмуванні, яке базується безпосередньо на лямбда-численні, функції є громадянами першого класу, а обчислення виражаються як оцінка математичних функцій. Ця чистота функцій забезпечує паралельне виконання, оскільки кілька застосувань функцій можна оцінювати одночасно, не впливаючи на результати одне одного.

Ключ до такої придатності полягає в чистоті посилань, властивості виразів лямбда-числення, яка гарантує, що той самий вхід завжди дає той самий вихід без зміни стану системи. Ця властивість спрощує процес паралельного виконання, оскільки усуває проблеми щодо умов перегонів, взаємоблокувань або інших пов’язаних із станом проблем паралельності, які є поширеними в імперативних мовах програмування.

На практиці багато функціональних мов програмування, таких як Haskell, Erlang і Scala, використовують принципи лямбда-числення для реалізації

паралелізму. Використовуючи паралельне виконання чистих функцій, ці мови дозволяють програмістам писати паралельні програми, які є не тільки ефективними, але й легшими для розуміння з точки зору правильності.

Незважаючи на властиві переваги лямбда-числення для паралельного програмування, потреба у формальній перевірці залишається. Перевірка гарантує, що паралельні програми правильно ведуть себе в різних сценаріях виконання та дотримуються своїх призначених специфікацій. Це особливо важливо у функціональних мовах, де програми, засновані на лямбда-численні, часто складні та покладаються на функції вищого порядку, рекурсію та відкладене оцінювання, що може ускладнити міркування щодо їх поведінки в паралельному контексті.

Лямбда-числення забезпечує основу для подальших міркувань про поведінку програми як у послідовних, так і в паралельних варіантах. Ключова функція лямбда-числення, дозволяє розробникам формально працювати з еквівалентністю виразів у різних варіантах виконання. Цю властивість можна розширити до паралельних програм, де її можна використати, щоб довести, що паралельні шляхи виконання еквівалентні їхнім послідовним аналогам. Наприклад, можна використовувати правила редукції лямбда-числення, щоб продемонструвати, що паралельне виконання двох функцій еквівалентно їх послідовному створенню, якщо функції незалежні.

Формальні методи перевірки відіграють вирішальну роль у забезпеченні коректності паралельних програм. Ці методи включають побудову формальних моделей програм і перевірку того, що ці моделі дотримуються певних властивостей, таких як безпека, живучість і коректність. У контексті лямбда-числення та функціонального програмування можна застосувати формальні методи перевірки, щоб довести, що паралельні програми поводяться належним чином у всіх можливих сценаріях виконання.

Одним із найбільш широко використовуваних методів формальної перевірки є логіка Хоара, яка надає набір аксіом і правил висновку для міркувань про правильність програм. Хоча традиційно застосовується до імперативних програм,

логіка Хоара може бути адаптована до систем на основі лямбда-числення для перевірки правильності функціональних програм. У випадку паралельних програм логіку Хоара можна розширити для обробки одночасних операцій, дозволяючи розробникам довести, що певні властивості зберігаються для паралельних шляхів виконання.

Також один з потужних підходів для формальної перевірки у функціональних мовах – це логіка розділення, яка розширює логіку Хоара на міркування про програми, які маніпулюють спільними ресурсами. Логіка розділення особливо корисна при перевірці паралельних програм, оскільки вона дозволяє розробникам міркувати про незалежне використання ресурсів паралельними потоками, забезпечуючи безпечне та правильне використання спільних ресурсів.

Лямбда-числення також формує основу для систем залежних типів у функціональних мовах, таких як Haskell та Idris. Ці системи типів дозволяють розробникам кодувати формальні специфікації в самій системі типів, уможливлюючи перевірку таких властивостей, як правильність функції, використання ресурсів і завершення під час компіляції. У паралельних програмах залежні типи можна використовувати для перевірки того, що паралельні обчислення дотримуються певних обмежень, таких як правильне використання ресурсів або відсутність умов змагання.

Доступно кілька формальних інструментів перевірки, які допомагають розробникам у перевірці правильності паралельних програм, особливо тих, що базуються на лямбда-численні та мовах функціонального програмування. Ці інструменти використовують теоретичні основи лямбда-числення, щоб забезпечити автоматичну або напівавтоматичну підтримку для підтвердження правильності програми.

Лямбда-числення, зосереджене на чистих функціях і прозорості посилань, забезпечує надійну теоретичну основу як для паралельного програмування, так і для формальної перевірки паралельних програм. Використовуючи властивості лямбда-числення, мови функціонального програмування можуть виражати

паралельні обчислення у спосіб, який легше міркувати та перевіряти. Формальні інструменти перевірки, такі як Coq, Isabelle і Agda, будуються на цій основі, щоб забезпечити автоматизовану підтримку для підтвердження правильності паралельних програм, гарантуючи, що вони відповідають своїм офіційним специфікаціям і ведуть себе належним чином під час паралельного виконання.

1.4.4. Інструмент моделювання лямбда-термів та його верифікація

На початковому етапі дослідження було визначено задачу створення лямбда-оточення, яке забезпечить ефективну роботу в аспектах проведення експериментів, аналізу даних та розробки програмного коду. Для лямбда-оточення було встановлено наступні вимоги:

– Здатність емулювати лямбда-терми, а саме створювати об'єкти, що представляють змінні, атомарні терми, абстрактні та аплікаційні терми.

– Здатність емулювати процес редукції лямбда-термів згідно з визначеною стратегією редукції.

– Можливість програмного встановлення нових стратегій редукції та проведення редукцій термів з їх використанням.

– Збір статистичних даних про процес редукції, включаючи кількість кроків редукції, час редукції кожного кроку, інформацію про стан терму на кожному кроці редукції, включаючи сам терм, його деревоподібне представлення, кількість вершин, який саме з редексів підлягає редукуванню, глибину редексу та ширину представлення терму у вигляді дерева.

– Генерація датасетів лямбда-термів відповідно до визначених характеристик термів та розміру датасету. Характеристики можуть включати кількість редексів, ширину та глибину терму у деревоподібному представленні, співвідношення кількості аплікацій, абстракцій та змінних у термі, із можливістю задати середні, мінімальні та максимальні значення, а також середньоквадратичне відхилення.

Перед початком розробки було проаналізовано існуючі рішення та підходи. Оскільки основною мовою програмування було обрано Python, вивчалися лише бібліотеки, що доступні для цієї мови. Такі бібліотеки, як *orsinium-labs/python-*

lambda-calculus, *Deric-W/lambda_calculus*, та *ElliotPenson/Nameless* виявилися зручними та функціональними для емуляції лямбда-числення, його основних об’єктів та процесу редукції. Проте, вони мають значні недоліки, такі як відсутність можливості створення нових стратегій редукції, інструментів для збору статистичних даних та генерації датасетів з заданими характеристиками. Описані вимоги є критичними для даного дослідження, тому було прийнято рішення про розробку нового інструменту із забезпеченням необхідного функціоналу та подальшу ефективність роботи.

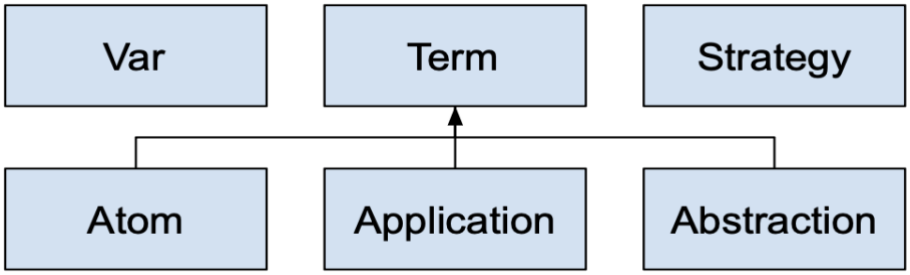


Рис. 1.1. Діаграма основних класів розробленої бібліотеки Lambda Calculus Environment.

Для вивчення лямбда-термів була розроблена бібліотека Lambda Calculus Environment на базі мови програмування Python, яка дозволяє моделювати та аналізувати обчислення за допомогою лямбда-термів. Цей програмний інструмент включає кілька класів, що відтворюють основні компоненти лямбда-числення, а також допоміжні класи для розробки стратегій редукції. Серед ключових класів (Рис. 1.1) можна виділити клас Var, який представляє змінні як унікальні об'єкти з унікальним ідентифікатором, визначеним додатним цілим числом, без структурної складності. Клас Term служить базовим класом для всіх інших класів, які представляють лямбда-терми, включно з класом Atom, що представляє терм однієї змінної, класом Application, що представляє терм аплікації, та класом Abstraction, що представляє терм абстракції. Система також включає допоміжні класи, які дозволяють визначати стратегію редукції, з батьківським класом Strategy.

Додатково, є інструменти для штучної генерації термів та перекладу термів із текстового формату, а також пакет стандартних термів. Клас Term, в свою чергу, надає різноманітні інтерфейси для роботи з лямбда-термами, зокрема можливість редукції терму за обраною стратегією, покрокової редукції вибраного редексу та інші функції для взаємодії з лямбда-термами. Ці інструменти забезпечують необхідний функціонал та дозволяють значно спростити дослідницьку роботу.

Основною обчислювальною операцією в безтиповому лямбда-численні є бета-редукція, яка включає наступні кроки:

– ідентифікація бета-редексу, тобто терму у формі ((@var_name. body_term) term);

– заміна всіх вільних входжень змінної *(x)* у *(M)* на *(N)*, з увагою до збереження стану вільних змінних у *(N)* після заміни.

Для формалізації бета-редукції у дослідженні використовується система координат для термів, яка включає множину *(S)* та застосовує принцип структурної індукції. Цей підхід дозволяє чітко визначити і контролювати процес заміни змінних, забезпечуючи коректність виконання бета-редукції.

Отже, із всього вищеперерахованого можна виділити вимоги до якості функціонування розробленого програмного забезпечення, що включають наступні пункти.

• *Однозначність інтерпретації ідентичних лямбда-термів, навіть за умови відмінності у змінних* – це означає, що програмне забезпечення має розпізнавати два лямбда-терми як однакові, навіть якщо вони використовують різні імена змінних. Наприклад, терми $\lambda x.x$ і $\lambda y.y$ мають бути інтерпретовані як ідентичні, оскільки вони є рівносильними з точки зору лямбда-числення. Це важливо для коректного збереження семантики термів під час обробки.

• *Однозначність виконання операції альфа-конверсії*. Альфа-конверсія – це процес перейменування зв'язаних змінних у лямбда-термах. Однозначність означає, що програмне забезпечення завжди виконує цю операцію правильно, забезпечуючи коректне перейменування змінних без зміни смислу терму.

Наприклад, у термі *λx.x*, альфа-конверсія може перетворити *x* у будь-яке інше ім'я, наприклад *y*, але результуючий терм (*λy.y*) повинен залишатися еквівалентним оригіналу.

• *Однозначність виконання операції бета-редукції*. Бета-редукція – це процес застосування функції до аргументу в лямбда-численні. Однозначність бета-редукції означає, що програмне забезпечення завжди коректно виконує цю операцію, застосовуючи терм до його аргументу і отримуючи єдиний можливий результат. Наприклад, для терму *(λx.x) y* бета-редукція повинна дати результат *y*, і це повинно бути єдине можливе значення.

• *Коректна робота з вільними та зв'язаними змінними*. Змінні у лямбда-термах можуть бути або вільними, або зв'язаними. Зв'язана змінна – це змінна, яка оголошена в межах якоїсь функції, а вільна – це змінна, яка не має такого оголошення в межах терму. Програмне забезпечення повинно коректно розрізняти ці типи змінних і правильно обробляти їх, щоб уникнути помилок, таких як випадкове змішування вільних і зв'язаних змінних під час редукцій.

• *Збереження ідентичності при редукції для лямбда-термів що після певної кількості кроків редукції залишаються такими*. Це стосується термів, які після певної кількості кроків редукції не змінюються і залишаються ідентичними. Програмне забезпечення повинно правильно визначати такі терми і зупиняти редукцію, коли досягнуто цього стану, забезпечуючи, що подальші редукції не змінюють результат.

• *Відповідність стратегій редукції до очікуваного алгоритму та ідентичність траєкторії редукції терму згідно зі стратегією*. Програмне забезпечення повинно відповідати обраній стратегії редукції і забезпечувати, що терми редукуються відповідно до очікуваного алгоритму. Наприклад, якщо обрана стратегія LO, терми повинні редукуватися в певній послідовності, і кінцевий результат має відповідати цій стратегії.

• *Відповідність визначених характеристик штучно-створених лямбда-термів до зазначених параметрів генерації*. Якщо програмне забезпечення генерує

лямбда-терми на основі певних параметрів, важливо, щоб ці терми відповідали цим параметрам. Це може включати певні властивості, такі як кількість змінних, глибина вкладеності або структура терму. Програмне забезпечення повинно забезпечувати, що створені терми відповідають заданим умовам.

Перевірка програми є важливим аспектом розробки програмного забезпечення, гарантуючи, що програма працює належним чином, без помилок, які можуть призвести до неправильної або непередбачуваної поведінки. З теоретичної точки зору перевірка програми сприяє формальній коректності програмних систем, забезпечуючи математичну основу, яка гарантує, що програма відповідає своїм специфікаціям. У контексті розробленого середовища для лямбда-числення перевірка гарантує, що фундаментальні операції, такі як генерація термів, альфа-перетворення та бета-зменшення, виконуються точно відповідно до строгих правил формалізму лямбда-числення. Це важливо для уникнення незначних помилок, особливо у функціональному програмуванні та символьних обчисленнях, де невеликі невідповідності можуть поширюватися та призводити до серйозних проблем. На практиці перевірка програми підвищує надійність і стійкість програмного забезпечення, мінімізуючи помилки та покращуючи продуктивність.

У розвиненому середовищі, особливо тому, що використовується для навчання, дослідження чи розширеної алгоритмічної розробки, перевірка гарантує, що всі функції, від базових маніпуляцій термінами до складних стратегій скорочення, працюють правильно. Це не тільки зміцнює довіру до системи, але й забезпечує стабільну платформу для подальшого розвитку та дослідження, що дозволяє впроваджувати нові функції з упевненістю, що базова система залишається надійною. Таким чином, досягається як теоретична коректність, так і практична надійність, що сприяє розробці високоякісного надійного програмного забезпечення.

Враховуючи зазначені вимоги можна визначити багатоетапний метод верифікації розробленого програмного забезпечення, що включає поетапну перевірку кожного складового блоку розробленого програмного забезпечення.

1. Визначення множини тестових термів, що включає стандартні терми перечисленні в табл. 1.1, та їх поєднання, що дозволило зібрати 30 термів, що перевіряють виконання вимог функціонування. До цих термів увійшли як прості терми на 1–5 змінних так і великі терми, що реалізують певні алгоритми як то пошук найбільшого загального дільника. Однозначність визначається через відповідність текстової репрезентації отримуваних термів.

2. Реалізація основних класів (Term, Var, Abstract, Application) не вимагає ретельного тестування. Проте, наявність функції для текстової репрезентації термів, що зберігається в зазначеній програмній структурі, дозволяє перевірити їх через просте порівняння із очікуваним рядком.

3. Реалізація методу альфа-конверсії, перевіряється з кожним із 30 термів на предмет коректності, результатом очікується незмінна текстова репрезентація із використанням імен-псевдонімів для змінних, проте актуальної заміни вільних змінних.

4. Реалізація методу бета-редукції, для термів що є в нормальній формі та ж сама нормальна форма, а для термів, що мають хоча б один крок редукції має стати наступний терм в траєкторії редукції згідно з обраним редексом.

5. Реалізація методу збереження термів у файлах тестового формату не вимагає додаткових перевірок, адже ґрунтується на використанні методу текстової репрезентації терму.

6. Реалізація методу інтерпретації термів із текстової репрезентації має бути терми, що відповідають текстовій репрезентації із використанням методу представлення терму через імена-псевдоніми. Однозначність перевіряється на всіх зазначених термах.

7. Реалізація методу генерації лямбда-термів вимагає перевірки отриманих термів на вимоги, що ставились під час генерації (що є по суті характеристиками дерева, що представляють лямбда терми) та мінімізації відсотку однакових термів.

8. Реалізація кожної окремої стратегії редукції (як то LO, LI, RI, RO чи розроблених нових стратегій) вимагає перевірки відповідності вибору правильного

редексу під час кожного кроку редукції та відповідність траєкторій редукції згідно із стратегією. Цей етап також вимагає введення специфічних термів, що можуть впадати в нескінченні послідовності редукції через специфіку обраної стратегії.

9. При додаванні нового функціоналу визначено необхідність в перевірці виконання кожного із зазначених пунктів, та можливе перевизначення множини лямбда-термів для тестування.

Рис. 1.2. Основні етапи методу верифікації програмної реалізації Lambda Calculus Environment.

Висновок до розділу 1

В першому розділі дисертації показано, що функціональні мови програмування є ефективним інструментом розв’язання багатьох актуальних задач, зокрема верифікації програмного забезпечення, незважаючи на певні їх недоліки, основним з яких є низька продуктивність виконання коду. Саме цей аспект є фокусом дослідження. На початковому етапі було вирішено перейти від функціонального програмування до лямбда-числення, що дозволяє ефективно

розробляти рішення, не ускладнюючи задачу більше, ніж необхідно. Такий перехід є можливим завдяки тісному взаємозв'язку між лямбда-численням та функціональним програмуванням, де лямбда-числення виступає як фундамент для створення та глибшого розуміння функціональних мов програмування.

Показано, що одним із ключових напрямків удосконалення є підвищення продуктивності стратегій редукції у лямбда-численні. Оскільки універсальної оптимальної стратегії для всіх термів не існує, метою дослідження є підвищення ефективності існуючих стратегій за допомогою методів машинного навчання. Відсутність оптимальної стратегії піднімає питання про покращення ефективності наявних підходів. Для досягнення цієї мети було обрано низку методів машинного навчання, які вже довели свою спроможність у різноманітних застосуваннях.

Серед можливих підходів до підвищення ефективності можна розглянути змішування стратегій. Ця ідея тісно пов'язана з теорією ігор, де в деяких випадках використання однієї з існуючих стратегій у чистому вигляді не гарантує перемогу, а лише запобігає поразці. Так, перемогу можна здобути лише шляхом комбінування існуючих стратегій.

Змішування стратегій полягає у використанні різних стратегій на кожному кроці редукції з певною ймовірністю. Налаштування цих ймовірностей має здійснюватися відповідно до підмножини термів, для яких це сприяє підвищенню ефективності.

Іншим варіантом підвищення ефективності стратегій може стати використання методів штучного інтелекту, які аналізують історію ефективних послідовностей кроків редукції або, що є ближчим до нашого підходу, характеристики терму. Ці методи використовують існуючі дані для визначення, які стратегії є найефективнішими, виходячи з кількості кроків редукції або часу, витраченого на кожен крок.

Подальші кроки дослідження включають побудову та аналіз параметричної рандомізованої стратегії. Довгострокова мета — застосування методів машинного навчання для вибору ефективних стратегій редукції для конкретних термів або

кроків. Актуальні задачі включають розробку алгоритмів, що прогнозують кількість кроків редукції та час виконання для кожного редексу. Це дозволить вибирати найефективнішу стратегію. Для досягнення цього планується провести кластерний аналіз термів для виявлення характеристик, що впливають на оптимальність стратегій.

Також в розділі представлено вирішення одного із пріоритетних завдань розробки лямбда-оточення для проведення експериментів, збору статистичних даних, а також створення датасету, необхідного для експериментальної роботи. До цього лямбда-оточення були поставлені вимоги до функціонування розробленого програмного забезпечення. Згідного вимог до розробленого програмного забезпечення визначено багатоетапну процедуру верифікації. Процедура визначає певний набір лямбда-термів із ключовими умовами які мають бути виконані й враховані при виконанні на них доступних операції. Також процедура визначає перевірку кожного виконання кожного із зазначених пунктів при додаванні нового функціоналу, та перевизначення множини лямбда-термів для тестування.

У розділі розглядається зв'язок між лямбда-численням, паралельним програмуванням і верифікацією програм. Лямбда-числення забезпечує основу для створення й оптимізації паралельних програм завдяки чистим функціям, що дозволяють уникати проблем паралельного виконання, таких як умови перегонів і взаємоблокування. Формальні методи перевірки, включаючи логіку Хоара, логіку розділення і залежні типи, застосовуються для забезпечення коректності паралельних програм. Інструменти на зразок Coq, Isabelle і Agda допомагають автоматизувати перевірку програм, базуючись на принципах лямбда-числення.

РОЗДІЛ 2.
ЗМІШУВАННЯ СТРАТЕГІЙ ТА ПРОГНОЗУВАННЯ ЧАСУ ОДНОГО КРОКУ РЕДУКЦІЇ

2.1. Змішування стратегій

Одним із перших підходів, який було розглянуто у роботі, є змішування стратегій або застосування рандомізованих стратегій як спосіб підвищення ефективності стратегії редукції. Ця ідея походить із теорієї ігор, де змішування стратегій веде до оптимальних результатів [41]. Рандомізовані алгоритми використовуються як засіб випадкової оптимізації, що часом дозволяє зменшити час обробки, складність, або об’єм використаної пам’яті порівняно зі стандартними детерміністичними алгоритмами. Це було підтверджено застосуваннями рандомізованих алгоритмів, що дозволило підвищити продуктивність існуючих рішень, як це показано у дослідженні [42]. Теоретично, додавання випадковості до стандартних стратегій редукції лямбда-термів може зменшити кількість кроків редукції. Основною ідеєю даного підходу є налаштування параметрів випадковості таким чином, щоб оптимізувати алгоритм відповідно до конкретної підмножини термів.

Стандартний метод вимірювання продуктивності стратегій редукції в лямбда-численні базується на підрахунку кількості кроків редукції [23, 43]. Таким чином, продуктивність стратегії визначається як очікуване значення кількості кроків редукції, виміряне для рівномірно згенерованих лямбда-термів однакового розміру. Даний підхід буде використано для оцінки та порівняння продуктивності стратегій редукції. Даний підхід не є ідеальним, оскільки на практиці деякі кроки редукції можуть відрізнятися за часом, як це було показано у роботі [44], та описано у наступному етапі дослідження, тому менша кількість кроків не завжди може означати менший час.

Розглянуті вище стратегії редукції лямбда-термів, включаючи CBN, CBV, LO та RI, є детерміністичними. Ці стратегії у чистому вигляді діють згідно з чітко

визначеними правилами, за якими на кожному кроці редукції вибирається конкретний редекс для редукції терму.

Нижче розглянуто можливі варіанти змішаних та випадкових стратегій [44]:

Uniform Random (UR):

– **Походження та визначення**: UR є стратегією без фіксованого правила вибору редексів, де редекс для редукції вибирається випадково з множини доступних редексів на поточному етапі редукування термів.

– **Алгоритм**: стратегія обходить дерево терму, ідентифікує всі доступні редекси та випадково (або згідно з певним розподілом ймовірностей) вибирає редекс для редукції [44].

– **Переваги**: Може перевершувати детерміністичні стратегії, уникаючи їх недоліків, таких як можливість нескінченних циклічних редукцій.

– **Недоліки**: випадковий вибір може призвести до неоптимальних траєкторій редукції. Усі можливі рішення розглядаються як рівноцінні, що ускладнює оцінку якості рішення; неможливість отримати один і той самий результат при редукції одного терму декілька разів.

– **Використання**: наразі не існує широко застосованих практичних рішень, які використовують цю стратегію.

Mixed Reduction:

– **Походження та визначення**: Mixed Reduction – це стратегія, яка не слідує фіксованому правилу вибору редексів. Замість цього, редекс для редукції обирається за допомогою випадково визначеного правила редукції [44].

– **Алгоритм**: Стратегія переглядає дерево терму, визначає доступні редекси та випадково (або згідно з певним розподілом ймовірностей) застосовує одну із доступних стратегій редукції, згідно з якою відбудеться редукція терму [44].

– **Переваги**: стратегія може перевершувати детерміністичні підходи, уникаючи їх недоліків. Наприклад, застосування стратегії RI у чистому вигляді може призвести до нескінченної циклічної редукції, в той час як стратегія LO для того самого терму може завершити редукцію за обмежену кількість кроків.

– **Недоліки**: випадковий вибір стратегії редукції може призвести до неоптимальних траєкторій редукції лямбда-термів; неможливість отримати один і той самий результат при редукції одного терму декілька разів.

– **Використання**: наразі не існує практичних рішень, які б активно використовували цю стратегію.

2.2. Генерація датасету лямбда-термів

Описаний далі підхід є загальним для всього дослідження, з урахуванням деяких особливостей для різних етапів, таких як кількість термів, та деякі зміни у параметрах генерації. Відповідні нюанси будуть описані окремо для кожного етапу.

Для рівномірного генерування лямбда-термів використовувалася модель Больцмана. Генератор лямбда-термів, заснований на цій моделі, ефективно враховує рекурсивні особливості термів. Рекурентне відношення $S(m,n)$ визначає кількість лямбда-термів з бінарними деревами розміром n і не більше ніж m вільними змінними, що дозволяє встановити функцію генерації лямбда-термів. Ця функція служить для створення бієкцій або функцій ранжування між всіма додатними цілими числами, що не перевищують $S(m,n)$. Підхід базується на роботах Ніенхейса та Уіла [45]. Для процедури генерації лямбда-термів була взята дана стаття [46] за основу.

Розроблена бібліотека Lambda Calculus Environment дозволяє штучно генерувати терми. При генерації лямбда-термів можна встановити мінімальні та максимальні межі для кількості вершин та редексів [43]. Після генерації виконується процедура фільтрації, яка видаляє терми з надто довгими послідовностями кроків редукції, використовуючи обрану стратегію, стандартно LO, або через обмеження на споживання пам'яті, задані наперед.

Використовуючи зазначену модель Больцмана для генерації лямбда-термів була створена відповідна процедура із застосуванням рекурсивного випадкового алгоритму.

Процедура генерації складається з наступних кроків:

1. Встановлення ймовірності генерації підтермів типу Abstraction, Application і Atom.

2. Встановити список змінних, які використовуються для генерації термінів.

3. Встановити максимальну кількість вершин у згенерованому термі.

4. Враховуючи ймовірності генерації підтермів випадковим чином виконати одну з трьох дій.

4.1. Згенерувати терм Atom на основі випадкової величини зі списку змінних.

4.2. Згенерувати терм Abstraction на основі змінної, яка є випадковою змінною зі списку змінних, і тіла, яке є випадково згенерованим термом із параметрів пунктів 1 і 2 і зменшеною кількістю вершин.

4.3. Згенерувати терм Application на основі термів суб'єкта та об'єкта, які випадково генеруються з параметрами пунктів 1 і 2 і зменшеною кількістю вершин.

5. Зменшити кількість вершин на одну та поверніться до пункту 4, доки кількість вершин не досягне 0.

Після генерації набору термів відбувається їх фільтрація за фактичною кількістю вершин, подібними термами та за кількістю кроків редукції; наприклад, за стратегією редукції LO можливо відфільтрувати терми, представлені занадто довгими послідовностями, які з'являються під час процесу редукції [43].

Для даного експерименту згідно з алгоритмом рівномірної генерації лямбда-термів, описаним у [45], згенеровані терми мають значення n у діапазоні від 50 до 60, що визначено обчислювальними обмеженнями тестової системи. З кількох сотень згенерованих термів було обрано 100, які за стратегією LO можуть бути редуковані до нормальної форми за обмежену кількість кроків редукції. Обмеження на кількість кроків редукції було встановлене на рівні 400, що також враховує обчислювальні можливості цільової системи та наявну кількість термів.

2.3. Аналіз продуктивності рандомізованих стратегій

2.3.1 Підбір гіперпараметрів Mixed Reduction стратегії

Згідно з описом стратегії Mixed Reduction [43], використання стратегії вимагає задання вектору ймовірностей p вибору стратегій редукції із обраної

множини стратегій. Цей вектор є гіперпараметром Mixed Reduction стратегії. Отже, необхідно налаштувати вектор ймовірностей p для мінімізації кількості кроків редукції заданого набору лямбда-термів. Для підбору оптимального вектору ймовірностей необхідно вирішити задачу мінімізації суми необхідних кроків редукції для лямбда-термів із множини згенерованих термів.

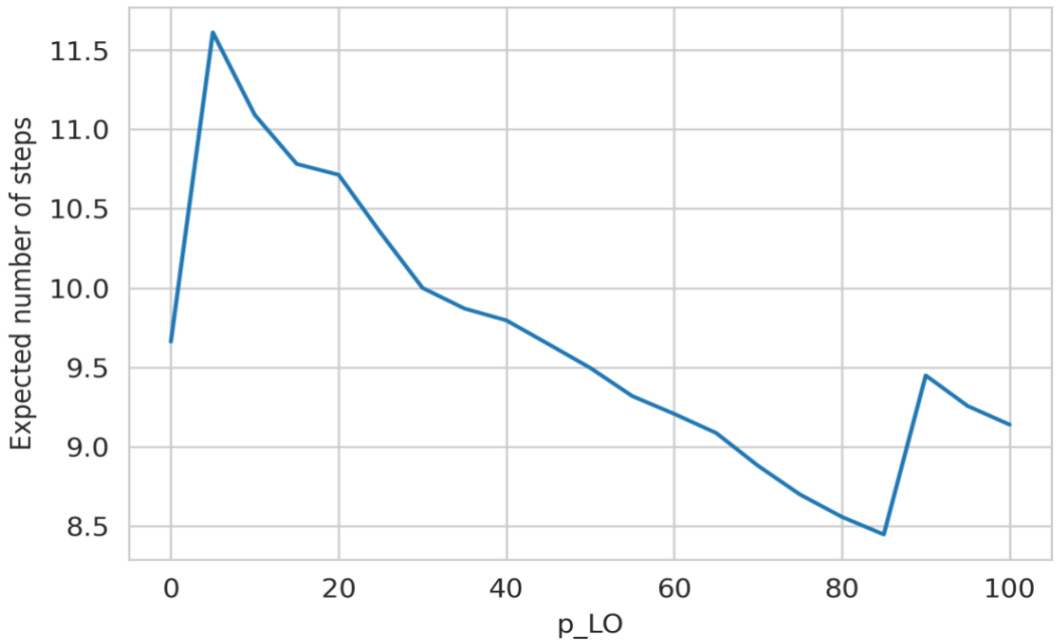


Рис. 2.1. Залежність продуктивності Mixed Reduction зі стратегіями LO, RI від ймовірності використання LO стратегії.

Загальновідомий підхід підбору гіперпараметрів є пошук по сітці (або повний перебір з деяким кроком). Проте такий підхід має недолік, оскільки вимагає тестування всіх наявних на сітці гіперпараметрів, що часто не є обчислювально ефективним. Також пошук по сітці обмежений точністю розбиття сітки гіперпараметрів, що може призводити до неможливості знаходження глобальних мінімумів через занадто великі кроки або використання занадто великої кількості обчислювальних ресурсів [43].

На противагу повному перебору пошуку по сітці генетичний алгоритм дозволяє знайти найкращу комбінацію за менший час і меншу витрату

обчислювальних ресурсів. Генетичні алгоритми дозволяють пошук у всьому просторі гіперпараметрів, не обмежуючись лише пошуком по сітці [47].

Розглянуто стратегію Mixed Reduction із двома стратегіями LO та RI, вектор p складається з ймовірностей для стратегій LO – p_{LO} та RI – p_{RI}, при чому $p_{LO} + p_{RI} = 1$ тоді налаштовуючи тільки гіперпараметр p_{LO} стає можливо налаштувати гіперпараметр p_{RI}. На Рис. 2.1 [43] показані результати такого налаштування із використанням перебору по сітці. Як показано на Рис. 2.1 отримане значення $p_{LO} = 0.85$ та відповідне $p_{RI} = 0.15$ дозволяють мінімізувати кількість кроків редукції для стратегії Mixed Reduction.

Таким чином, було знайдено гіперпараметри для стратегії Mixed Reduction, із використанням двох стратегій LO та RI. Показано, що така стратегія є ефективнішою за RI для нормалізації рівномірно згенерованих лямбда-термів, і на відміну від RI з більшою вірогідністю редукує всі терми до нормальної форми.

Далі було розглянуто стратегію Mixed Reduction, що є сумішшю LO, RI, LI , RO та UR стратегій. Для підбору гіперпараметрів вектору p ймовірностей стратегій в даному випадку використовувався генетичний алгоритм. Функція пристосованості для вектору p була критерієм продуктивності обраної змішаної стратегії. Генетичний відбір був реалізований через елітарний відбір за функцією пристосованості. Оператор кросинговеру був реалізований через бінарний кросовер, та оператор мутації реалізований через поліноміально обмежену функцію [43]. Результатом застосування цього генетичного алгоритму став вектор ймовірностей для стратегій де $p_{LO} = 0.72, p_{RI} = 0.02, p_{LI} = 0.02, p_{RO} = 0.17$ та $p_{UR} = 0.07$.

Розроблений генетичний алгоритм для підбору гіперпараметрів можна розширити на будь-які стратегії редукції в Mixed Reduction. Результатом є програмний компонент Pure Calculus Environment, що використовує Mixed Reduction та налаштовує гіперпараметри лямбда-термів еволюційним процесом. Використання LO в Mixed Reduction з ненульовою ймовірністю призводить до нормалізації терму, якщо така форма існує.

2.3.2. Результати експериментів визначення продуктивності змішаних стратегій

Були проведені експерименти на зазначеному наборі 100 лямбда-термів для виявлення продуктивності стандартних детерміністичних стратегій редукції LO, RI, а також розроблених стратегій UR та Mixed Reduction. На основі множини згенерованих 100 лямбда-термів було побудовано гістограми розподілів необхідної кількості кроків редукції для отримання нормальної форми термів за допомогою стратегій редукцій LO, RI, UR та Mixed Reduction (із стратегіями LO, LI, RI, RO, UR), результати наведені на Рис. 2.2, Рис. 2.3, Рис. 2.4 та Рис. 2.5 [43]. Також було перевірено гіпотези щодо законів розподілу значень кількості кроків відносно обраної стратегії для рівномірно згенерованих лямбда-термів. Були розглянуті припущення про такі закони розподілу: нормальний, Коші, χ^2, експоненційний, узагальнений нормальний (експоненціальна ступінь), гамма, логарифмічний нормальний та Рейлі. Отримані графіки зазначених розподілів також наведені на Рис. 2.2, Рис. 2.3, Рис. 2.4 та Рис. 2.5.

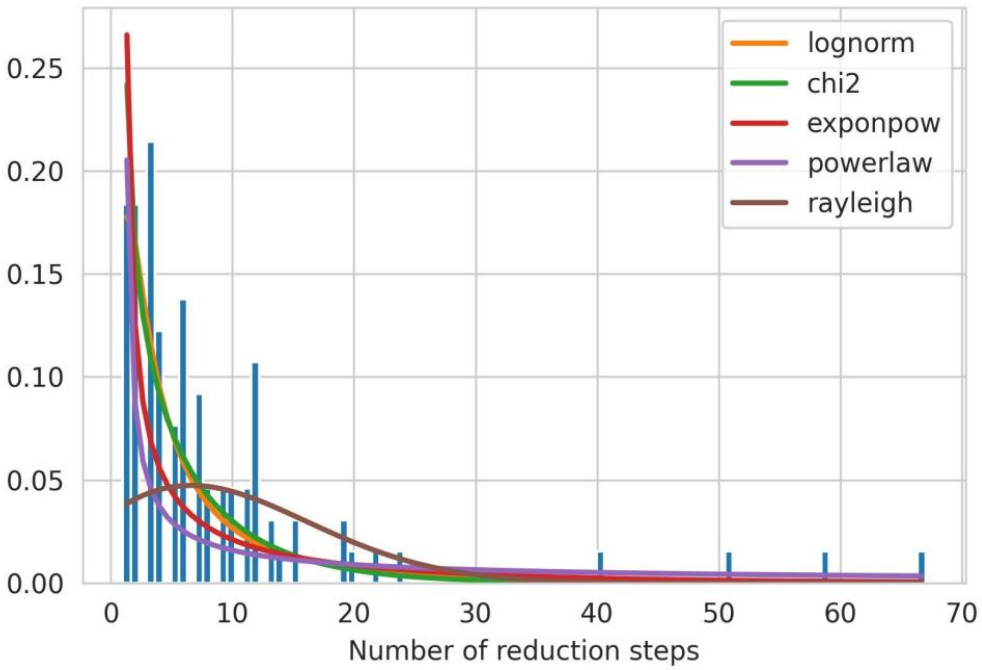


Рис. 2.2. Гістограма та графіки отриманих розподілів кількості кроків редукції за стратегією LO для рівномірно згенерованих лямбда-термів.

При апроксимації даних за допомогою логарифмічного нормального розподілу для рівномірно згенерованих лямбда-термів редукованих за допомогою стратегії LO (Рис. 2.2): очікуване значення (середнє) кількості кроків скорочення становить 1.65; стандартне відхилення становить 1.06; очікуване значення кількості кроків за цим розподілом становить 9.14.

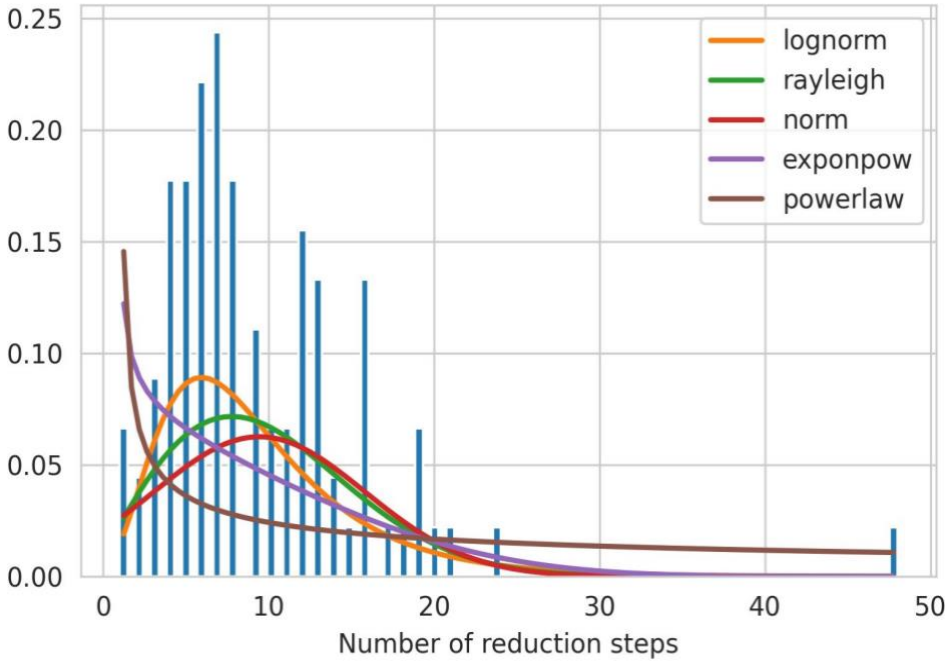


Рис. 2.3. Гістограма та графіки отриманих розподілів кількості кроків редукції за стратегією RI для рівномірно згенерованих лямбда-термів.

Ті ж показники для рівномірно згенерованих лямбда-термів редукованих за допомогою стратегії RI (Рис. 2.3): очікуване значення (середнє) кількості кроків скорочення становить 2.04; стандартне відхилення становить 0.68; очікуване значення кількості кроків за цим розподілом становить 9.66.

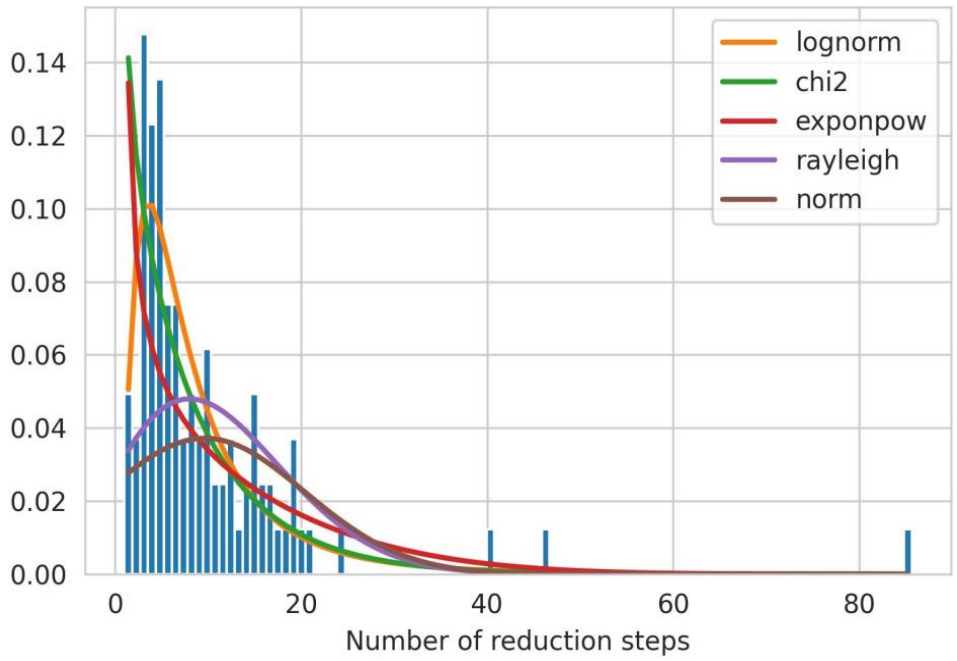


Рис. 2.4. Гістограма та графіки отриманих розподілів кількості кроків редукції за стратегією UR для рівномірно згенерованих лямбда-термів.

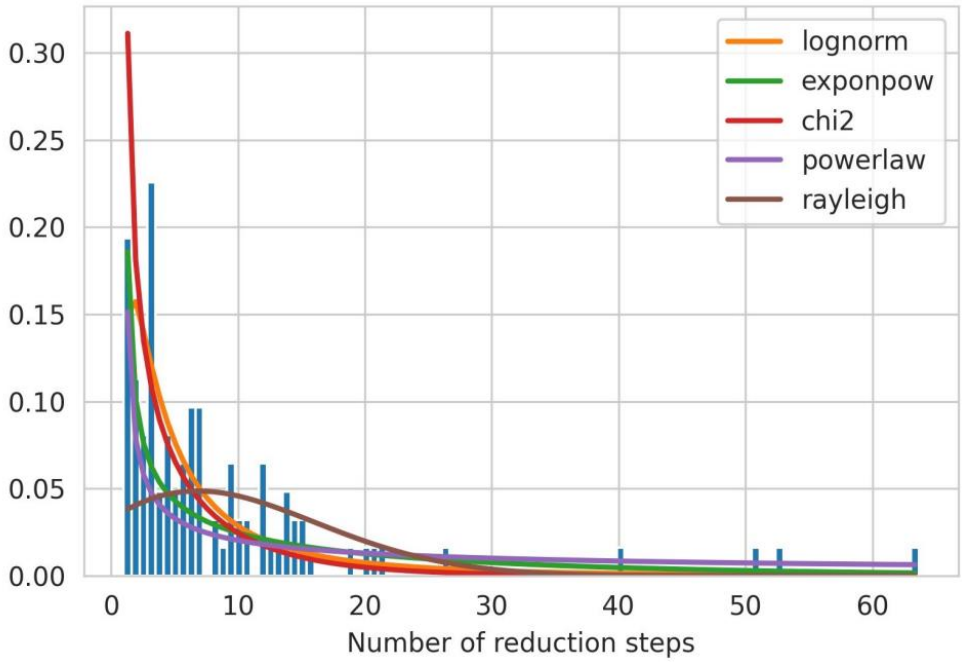


Рис. 2.5. Гістограма та графіки отриманих розподілів кількості кроків редукції за стратегією Mixed Reduction (із стратегіями LO, LI, RI, RO, UR) для рівномірно згенерованих лямбда-термів.

Показники для log-нормального розподілу для даних рівномірно згенерованих лямбда-термів редукованих за допомогою стратегії UR (Рис 2.4): очікуване значення (середнє) кількості кроків скорочення становить 1.92; стандартне відхилення становить 0.8; очікуване значення кількості кроків за цим розподілом становить 9.42.

Результати для логарифмічного нормального розподілу для даних рівномірно згенерованих лямбда-термів редукованих за допомогою стратегії Mixed Reduction (із стратегіями LO, LI, RI, RO, UR) (Рис. 2.5): очікуване значення (середнє) кількості кроків скорочення становить 1.67; стандартне відхилення становить 0.96; очікуване значення кількості кроків за цим розподілом становить 8.39.

З отриманих результатів можна побачити, що ефективність стратегії UR співмірна з ефективністю LO та RI стратегій. Таким чином, стратегію UR можна використовувати для редукції лямбда-термів так само, як і детерміністичні стратегії. Стратегія Mixed Reduction виявилася найкращою серед розглянутих стратегій з точки зору очікуваного середнього значення за логарифмічним нормальним розподілом. Крім того, розглянута стратегія Mixed Reduction (із стратегіями LO, LI, RI, RO, UR) веде до нормальної форми терму із великою ймовірністю (іншими словами є нормалізуючою). Також всі 100 згенерованих лямбда-термів були нормалізовані зазначеною Mixed Reduction стратегією.

2.4. Прогнозування часу одного кроку редукції

2.4.1. Постановка проблеми

В лямбда-численні існує складна проблема оцінки обчислювальної складності редукції термів [44, 48]. На відміну від загальноприйнятого спрощення, що всі кроки редукції є однаковими, варто зосередитися на їх фактичній відмінності: не всі кроки є однаково складними – деякі складніші за інші. Розуміння цієї диференціації є критичним, оскільки воно має безпосередній вплив на ефективність обчислень.

Ще самі перші математичні роботи присвячені лямбда численню зазначають непрактичність ідеї сприйняття кроків редукції як ідентичних процесів [44]. Проте

так і не було запропоновано достатньо простого й універсального рішення цієї проблеми. Багато досліджень зосереджувалися на стратегіях редукції лямбда-термів, але не приділяли уваги тому скільки обчислювальних ресурсів витрачається на проведення редукцій, що в реальності залежить від топології терму та обраного редексу [44, 48]. Проте варто зазначити, що існують роботи де були спроби їх авторів обчислення обчислювальної складності редукції використовуючи аналіз деяких стратегій редукції та їх вимоги в пам’яті [49, 50], або вимірювати складність за допомогою моделей оцінки вартості редукції [51]. Також автори роботи [52] визначали деякі класи термів по обчислювальним вимогам редукції, а також визначали типи стратегій редукції відносно їх очікуваних обчислювальних вимог. А також була розглянута в роботі [53] слабка модель інваріантної вартості редукції термів лямбда числення, що була заснована на теоретичних припущеннях.

Ефективність редукції впливає на продуктивність програм, зменшуючи ресурси та прискорюючи обчислення, що важливо для систем реального часу та складних завдань. Стратегії редукції, такі як CBN, CBV, LO і RI, мають різні переваги та недоліки, що впливають на продуктивність. Вибір стратегії ускладняється через різницю в обчислювальних витратах. Немає загальноприйнятої метрики для точного вимірювання складності редукції термів; існуючі методи або занадто спрощені, або складні. Необхідна метрика, що точно відображає обчислювальну вартість редукції та застосовується до реальних задач. Розробка метрики на основі реальних умов лямбда-числення допоможе поєднати теорію з практикою.

2.4.2. Формулювання гіпотези і засобів дослідження

Для аналізу редукції термів у безтиповому лямбда-численні потрібно сформулювати гіпотези та методи дослідження. Передбачається, що параметри лямбда-термів, такі як кількість вершин, редексів, висота і ширина дерева, а також глибина редексу, можуть визначати час одного кроку редукції. Це дозволить оцінити обчислювальну складність редукції. Виходячи з цього, можна висунути 2 припущення [54].

1. **Припущення перше (прогнозування складності)** виражається в тому, що параметри терму в поєднанні з даними про редекс, обраний для редукції, перед процесом редукції можуть дати змогу передбачити обчислювальну складність наступного кроку редукції. Ця модель прогнозування має на меті передбачити час, необхідний для редукції, на основі характеристик терму перед редукцією.

2. **Друге припущення (визначення складності)** розширює перше шляхом включення даних термів як до, так і після процесу редукції разом із даними про сам редекс. Цей підхід призначений для визначення обчислювальної складності після редукції, пропонуючи зрозуміти фактичні обчислювальні зусилля, витрачені на процес редукції.

Збір даних. Дослідження обчислювальної складності лямбда-термів передбачає збір деякого набору даних лямбда-термів з різноманітною структурою та складністю. Цей набір даних містить інформацію про параметри терму як до так і після редукції, характер задіяних редексів і фактичний час, витрачений на кожен крок редукції.

Опис розширеної методології. Методологія дослідження обчислювальної складності редексів лямбда-термів ґрунтується на принципах машинного навчання і регресійного аналізу з метою розробки моделі прогнозування, яка точно оцінює час, витрачений на окремі кроки скорочення в термах лямбда-числення. Дана оцінка є сурогатною мірою обчислювальної складності. У наведеному нижче розділі розкриваються методи регресії, які розглядалися спочатку, вводяться додаткові складні моделі та обговорюються теоретичні основи та практичні міркування щодо застосування цих методів [54].

I. Обговорення методів регресії:

1. **Лінійна регресія** – це статистичний метод для моделювання зв’язку між залежною змінною та однією або кількома незалежними змінними. Мета – знайти лінійне рівняння, яке найкраще прогнозує залежну змінну. У простій регресії з однією незалежною змінною використовується лінія $y = mx + b$, а в множинній –

гіперплощина для кількох змінних. Лінійна регресія широко застосовується для прогнозування та аналізу даних [55].

Практичні міркування. Для вирішення задачі лямбда-числення, лінійна регресія може бути доволі ефективним засобом для пошуку взаємозв'язків між простати термами, та часом редукції редексу. З іншого боку для більш складних термів ефективність даного підходу може зменшитися.

2. **Регресійні дерева рішень** – це метод навчання, який прогнозує постійні результати шляхом розподілу даних на підмножини у формі дерева. Вузли дерева представляють точки прийняття рішень, які розділяють дані за певними умовами, а листки – це прогнози. Поділ на вузлах здійснюється на основі критеріїв, таких як середньоквадратична помилка (MSE). Регресійні дерева легко інтерпретувати і використовувати, обробляють нелінійні зв'язки без попередньої обробки, але можуть перенавчатися, що вимагає технік, як обрізка чи обмеження глибини [56].

Практичні міркування: Регресійні дерева рішень є більш ефективними ніж лінійна регресія з точки зору роботи з нелінійними залежностями, але з іншого боку вони більш схильні до перенавчання.

3. **Support Vector Regression (SVR)** – це метод регресії, заснований на принципах SVM. Він прогнозує безперервні значення, намагаючись вмістити якнайбільше даних у межах певного порогу помилки (епсилон) і мінімізувати складність моделі. SVR зосереджується на опорних векторах, які знаходяться близько до гіперплощини, і ігнорує дані в межах епсилон, що робить його стійким до викидів. Метод підтримує різні ядра (лінійне, поліноміальне, RBF), що дозволяє моделювати як лінійні, так і нелінійні зв'язки між змінними. [57].

Практичне міркування: SVR може краще працювати для термів зі складними залежностями, де зв'язок між часом та термом є нелінійним та складним.

4. **Метод *k*-найближчих сусідів (kNN)** – це простий, універсальний алгоритм для класифікації та регресії, що визначає результат для точки даних на основі її *k* найближчих сусідів. Значення *k* обирається користувачем, а сусіди знаходяться за відстанню (наприклад, Евкліда або Мангеттена). Для регресії метод присвоює

середнє значення сусідів. Хоча kNN легко реалізувати, метод є обчислювально дорогим з ростом розміру даних і чутливий до масштабування та вибору *k* [58].

Практичні міркування: Метод доволі ефективним, але вимагає ретельного підбору параметра *k*, оскільки від нього може залежати якість навчання.

5. **Регресія штучних нейронних мереж (ШНМ)** – метод машинного навчання, натхненний біологічними нейронними мережами. ШНМ має вхідний, кілька прихованих і вихідний рівні, де нейрони виконують обчислення та навчаються моделювати складні зв'язки. Нелінійності досягаються через функції активації (Sigmoid, ReLU та інші). Навчання ШНМ полягає в налаштуванні ваг для мінімізації помилок прогнозування за допомогою зворотного розповсюдження і градієнтного спуску. ШНМ ефективні для класифікації, регресії і розпізнавання образів, особливо в глибокому навчанні для великих даних [59].

Практичні міркування: ШНМ потребують більше даних і обчислювальних ресурсів для навчання, але мають великий потенціал для роботи зі складними нелінійними залежностями. Як і інші моделі, ШНМ можуть перенавчатися і потребують тонкого налаштування гіперпараметрів.

II. Міркування щодо оцінки та вибору моделі:

1. **Перехресна перевірка** – це статистичний метод для оцінки моделей машинного навчання. Метод включає поділ даних на підмножини, навчання моделі на одній підмножині та перевірку на іншій, щоб перевірити її надійність на нових даних. Це допомагає запобігти перенавчанню і забезпечує точніший показник ефективності [60]**.**

2. **Налаштування гіперпараметрів** – це пошук оптимальних значень для параметрів, які контролюють навчання моделі. Використовують методи, такі як пошук у сітці, випадковий пошук або Байєсовська оптимізація, для створення і оцінки моделей з різними наборами гіперпараметрів для підвищення продуктивності [61].

3. **Зменшення розмірності множини змінних.** Зменшення розмірності змінних: Аналіз основних компонентів (PCA) і t-розподілене стохастичне

вбудовування сусідів (t-SNE) спрощують складні дані, зберігаючи важливу інформацію. PCA перетворює змінні в нові компоненти з найбільшою дисперсією, а t-SNE зберігає локальну структуру даних для візуалізації у 2D або 3D просторах [62, 63].

Методологія дослідження використовує кілька моделей різної складності для забезпечення комплексності та адаптивності аналізу. Тестування цих моделей дозволить краще зрозуміти лямбда-терми і набір даних. Розробка точної моделі для прогнозування часу редукції терму може суттєво покращити продуктивність функціональних програм.

2.4.3. Винесення припущень щодо очікуваних результатів

Можливість визначати час редукції для окремих кроків редукції допоможе відрізняти редекси за складністю та обирати найпростіший, що може зменшити обчислювальні витрати на редукцію. Це також підкреслить залежність між топологією терму і його складністю, покращуючи розуміння лямбда-числення. Розробка прогностичних моделей не лише оптимізує редукцію лямбда-термів і виконання програм, але й може бути використана для покращення компіляторів та інтерпретаторів. Такий підхід, що включає збір даних і оцінку моделей, допоможе краще зрозуміти динаміку редукції лямбда-термів.

2.4.4. Збір та аналіз даних

На даному етапі дослідження для експериментів було згенеровано 220 термів, які були потім редуковані за допомогою стратегій LO та RI, тож при розгляді кожного кроку редукції цих 220 термів було отримано 3931 запис із термом та часом у наносекундах, витраченим на даний крок редукції. Також потрібно зазначити, що в даному випадку час, витрачений на редукцію, складається з часу, витраченого на пошук редексу за обраною стратегією та час, витрачений на виконання редукції. В експериментах використовувалась операційна система на основі Unix для збору даних про час, оскільки вона дає більш чітке представлення часу, витраченого на процеси обчислення, ніж інші операційні системи. Такий

підхід має забезпечити максимально точну оцінку споживання обчислювальних ресурсів.

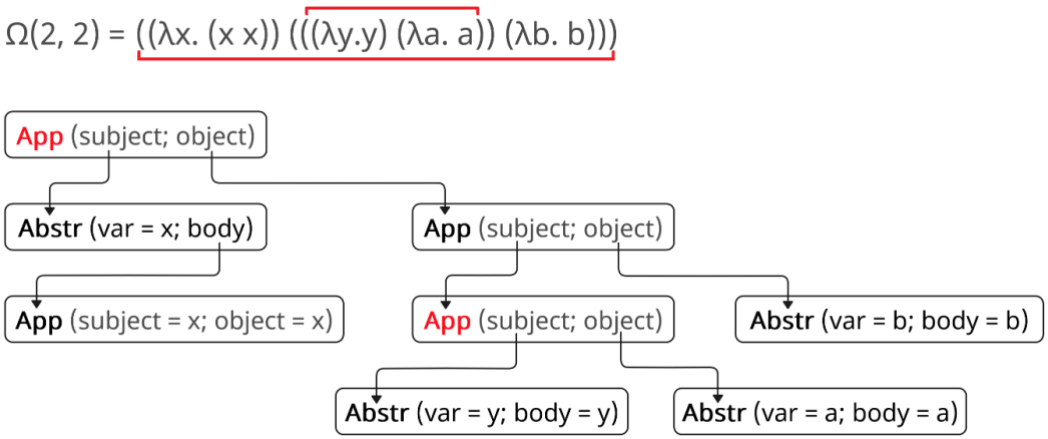


Рис. 2.6. Омега(2, 2) терм із позначеними редексами та його представлення дерева.

На Рис. 2.6 [54] показано, що терм можна представити у вигляді дерева, тому ознаки, які збираються про терм, є також параметрами, які описують це дерево:

1) **Висота** – найдовша послідовність вершин від кореня до листа дерева терму, що зазвичай є атомарним термом (для терму зображеного на Рис. 2.6 значення висоти дорівнює 4).

2) **Ширина** – кількість листів дерева терму, що зазвичай представляє собою кількість використаних атомарних термів у термі. (для прикладу дорівнює 4).

3) **Redexes** – кількість редексів у тілі терму (на Рис. 2.6 в стандартному представленні можна побачити 2 редекси, позначені червоним кольором, та у представленні терму у вигляді дерева деревом, вони позначені як аплікації, також червоним кольором).

4) **Вершини** – підрахунок вершин у представленні терму у вигляді дерева, з точки зору лямбда-числення, це являє собою всі аплікації, абстракції та атомарні терми (для терму на Рис. 2.6 кількість вершин дорівнює 12).

5) **Глибина редексу** – довжина послідовності від кореня дерева до обраного редексу (що представлений аплікацією) (для першого редексу терму показаного на Рис. 2.6, який є коренем дерева, вона дорівнює 1, а для другого редексу – 3).

6) **Час кроку** – час, витрачений на пошук відповідного редексу за допомогою обраної стратегії плюс час, витрачений на редукцію обраного редексу в наносекундах. Час витрачений на крок редукції було підраховано в розробленому середовищі Lambda Calculus при використанні Unix операційної системи.

В якості додаткового етапу, було зібрано параметри, що описують терм після процесу редукції, вони є ідентичними описаним вище, за винятком глибини редексу, який не визначається після етапу редукції. Також було отримано показники різниці між значеннями параметрів до та після процедури редукції.

Аналіз отриманих даних.

З точки зору регресійного аналізу, точність прогнозу залежить від розуміння взаємозв'язку між цільовою змінною та вхідними даними. На доцільність подальшого дослідження впливає наявність або відсутність такого взаємозв'язку.

Кореляція вимірює силу та напрямок лінійного зв’язку між двома змінними, кількісно визначаючи, як зміни в одній змінній пов’язані зі змінами в іншій. У той час як коефіцієнт кореляції Пірсона спеціально фіксує лінійні залежності, інші показники, такі як рангова кореляція Спірмена та тау Кендалла, можуть оцінити нелінійні зв’язки, зосереджуючись на ранжуванні значень, а не на їх фактичних величинах. В даному дослідженні була використана кореляція Пірсона, отже основний фокус було покладено на лінійні взаємозв'язки.

На Рис. 2.7 [54] представлені результати проведення кореляційного аналізу де показано: найбільше значення кореляції з часом необхідним на редукцію мають дані змінних, що характеризують представлення терму у вигляді дерева (висота, ширини, кількість вершин) та дані про кількість редексів – 0.56–0.77 (Рис. 2.7.a).

Також були розглянуті характеристики термів після редукції, вони мають меншу кореляцію з часом редукції – 0.48–0.68 (Рис. 2.7.b), а найменшу кореляцію має різниця змінних до та після редукції – -0.033–0.13 (Рис. 2.7.c).

Такі параметри, як висота та ширина, кількість редексів та вершин, їх показники після редукції та різниця показників до та після редукції мають високі значення кореляції, близькі до 1, це вказує на те, що ці параметри пов’язані

топологією дерева. Також слід підкреслити низьку кореляцію між часом кроку редукції та обраною стратегією та іншими параметрами терму. Виходячи з результатів кореляційного аналізу можна зробити висновок про те, що використання представлених даних для вирішення задачі регресії має потенціал розвитку.

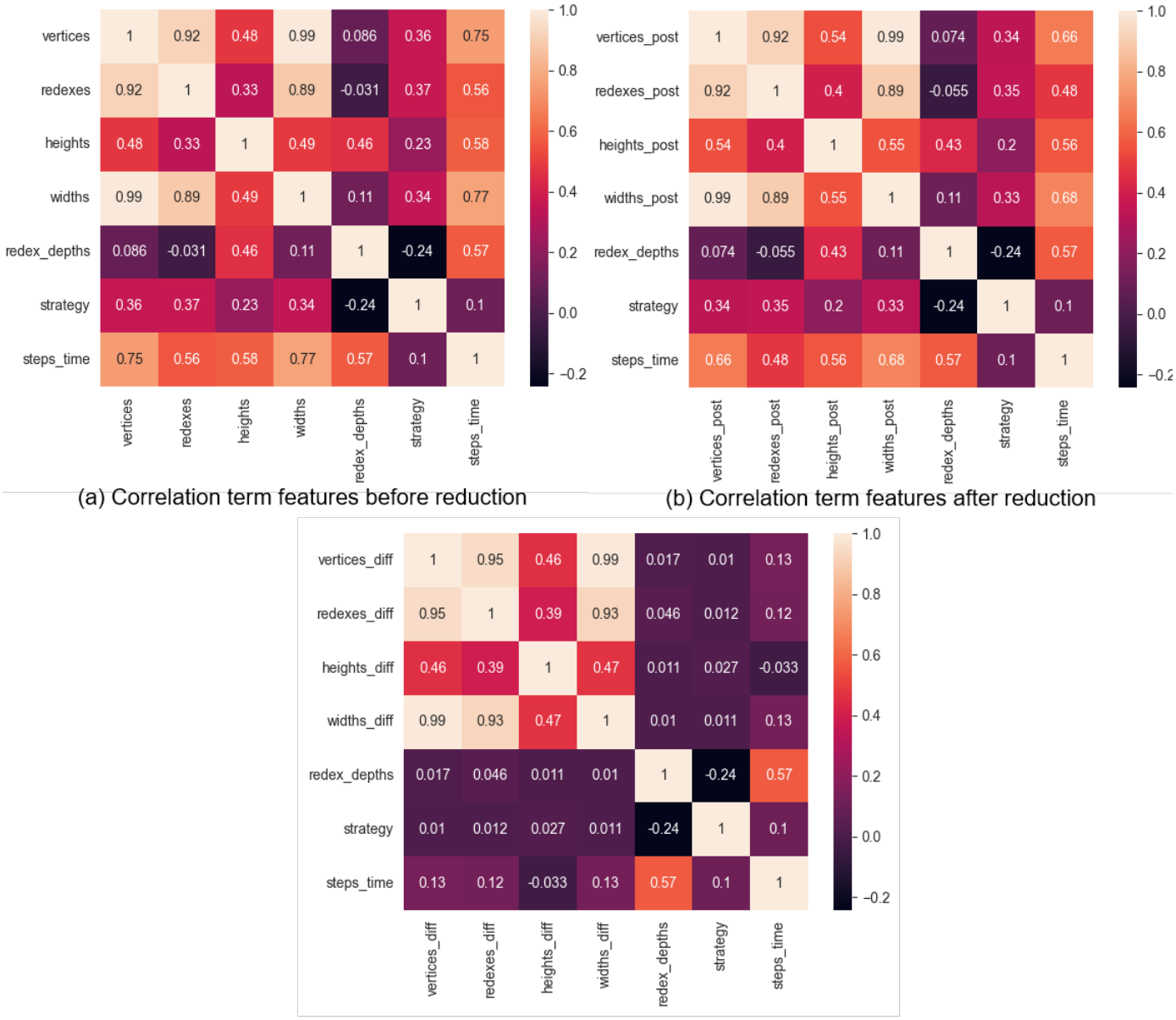


Рис. 2.7. Кореляційна матриця для параметрів терму (a) до виконання кроку редукції; (b) після виконання кроку редукції; (c) різниці параметрів терму до та після виконання кроку редукції.

2.4.5. Аналіз результатів

Аналіз припущень в контексті зібраних даних.

Попередньо було висунуто два припущення. Виходячи з яких, тепер для тренування та тестування моделей машинного навчання можливо визначити три набори даних обраних для вирішення проблеми регресії:

1. **Для прогнозування складності** кроку редукції необхідно використовувати дані терму до процесу редукції та дані про редекс.

Вхідні дані: ширина, висота, кількість вершин, кількість редексів та глибина обраного редексу.

2. **Для визначення складності** кроку редукції необхідно використовувати дані терму до і після процесу редукції та дані про кількість редексів і про обраний для редукції редекс.

Вхідні дані: ширина, висота, кількість вершин, кількість редексів, ширина після редукції, висота після редукції, кількість вершин після редукції, кількість редексів після редукції і глибина обраного редексу.

3. **Для визначення складності** кроку редукції, як і в попередньому випадку можливо використовувати дані терму до і після процесу редукції та значення різниці цих даних та дані про кількість редексів і дані про обраний редекс.

Вхідні дані: ширина, висота, кількість вершин, кількість редексів, ширина після редукції, висота після редукції, кількість вершин після редукції, кількість редексів після редукції, різниця ширини, різниця висоти, різниця кількості вершин, різниця кількості редексів і глибина обраного редексу.

Усі три варіанти датасету будуть використані для прогнозування часу витраченого на наступний крок редукції. В першому випадку задачею стоїть **прогнозування** складності, оскільки як вхідні дані використовуються дані, що доступні до процесу редукції, а в двох інших випадках – **визначення**, оскільки використовуються дані до та після процесу редукції

Цільова змінна: час кроку редукції.

Попередня обробка даних.

Для багатьох методів машинного навчання є важливим розподіл вхідних та вихідних даних, одним із найкращих варіантів є, наприклад, нормальний розподіл – (0; 1), (-1; 1). Розподіл такого типу гарантує якнайшвидше сходження алгоритмів.

Тож в роботі було проаналізовано розподіл ознак термів, що є вхідними даними, та розподіл вихідних даних – час одного кроку редукції.

Результати аналізу показують деякі проблеми в розподілі даних (Рис. 2.8), де першою проблемою є немасштабовані або занадто широкі розподіли, та другою є ненормальний розподіл даних деяких характеристик.

Проблеми як ненормальність та немасштабованість для розподілів вхідних та вихідних даних можуть погано впливати на роботу алгоритмів та процес навчання моделей. Серед характеристик терму з ненормальним розподілом можна виділити ширину дерева (Рис. 2.8.a, Рис. 2.8.e), кількість вершин (Рис. 2.8.c, Рис. 2.8.g), кількість редексів (Рис. 2.8.d, Рис. 2.8.h) як до, так і після процесу редукції. На Рис. 2.8 представлені у немасштабованому вигляді та детально описані всі змінні. В якості способу попередньої обробки з ціллю нормалізації розподілу цільової змінної (часу кроку редукції) було вирішено застосувати до неї функцію логарифму. Для вхідних змінних було застосовано перетворення Йео-Джонсона [64], оскільки воно може приймати нульові значення, та обробляє дані методом дуже схожим до логарифмічного, це дозволяє нормалізувати розподіл даних і автоматично масштабувати їх.

На Рис. 2.9 показано нові розподіли даних, що було отримано після застосування степеневого перетворення Йео-Джонсона для вхідних даних і логарифмування цільової змінної.

Нові розподіли даних для ширини дерева (Рис. 2.9.a, Рис. 2.9.e), кількості вершин (Рис. 2.9.c, Рис. 2.9.g) і кількості редексів (Рис. 2.9.d, Рис. 2.9.h) до і після кроку редукції, а також час кроку редукції, тобто цільової змінної (Рис. 2.9.n) стали наближеними до нормального розподілу, це можна легко побачити візуально. Такі ознаки як висота (Рис. 2.9.b, Рис. 2.9.f), а також показники різниці зазначених змінних (Рис. 2.9.i, Рис. 2.9.j, Рис. 2.9.k, Рис. 2.9.l) не змінилися. Дане перетворення

дозволяє забезпечити компактний розподіл значень для характеристик терму та цільової змінної. Представлені способи обробки даних дозволяють покращити якість моделі та прискорити конвергенцію алгоритму.

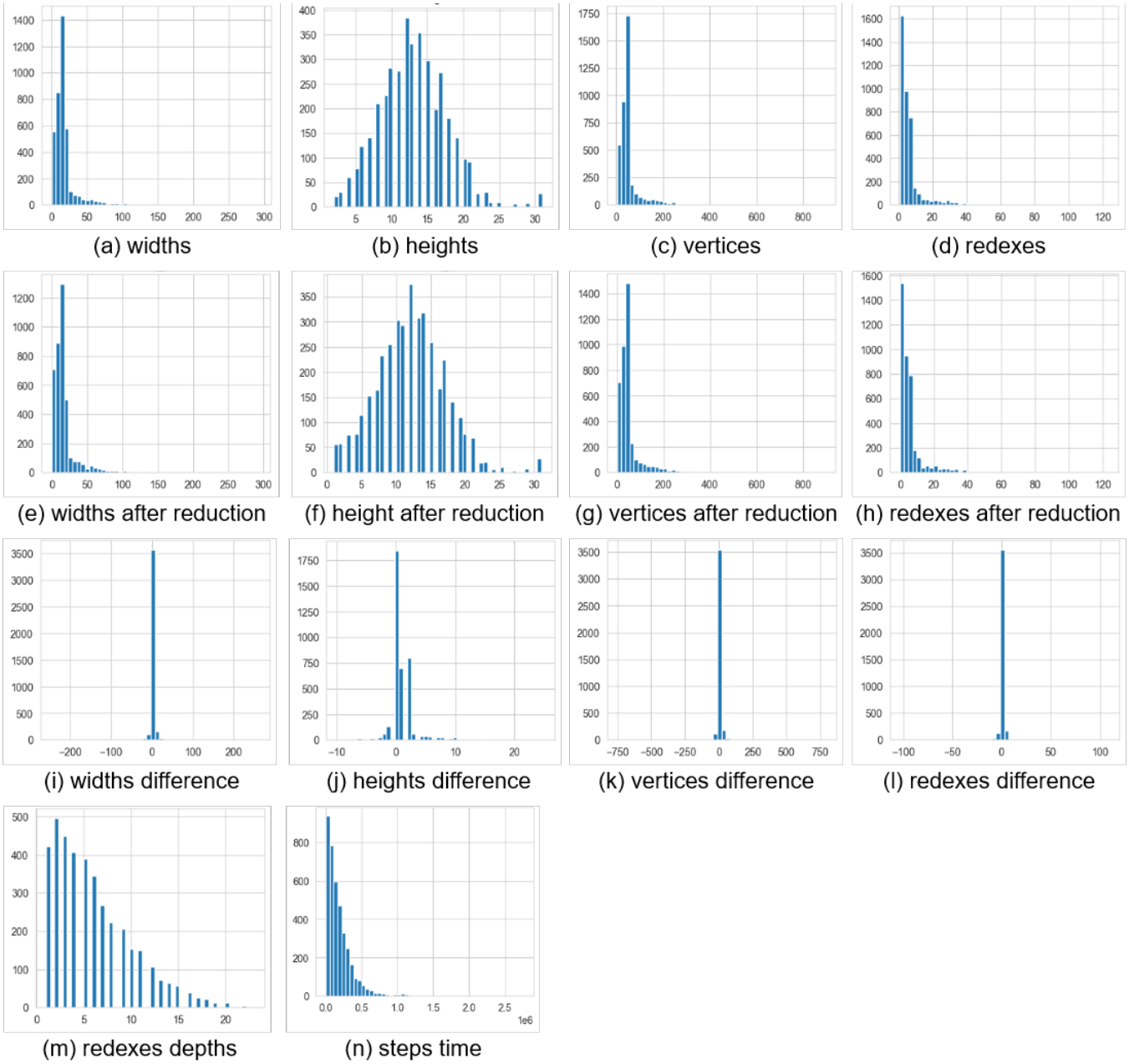


Рис. 2.8. Розподіл даних для всіх зібраних характеристик термів: (a) ширини, (b) висоти, (c) вершини та (d) редекси до редукції; (e) ширини, (f) висоти, (g) вершини (h) редекси після редукції; різниці (i) ширини, (j) висоти, (k) вершин та (l) редексів до та після редукції; (m) глибина редексів як ознака редексів;
і цільова змінна (n) кроків час.

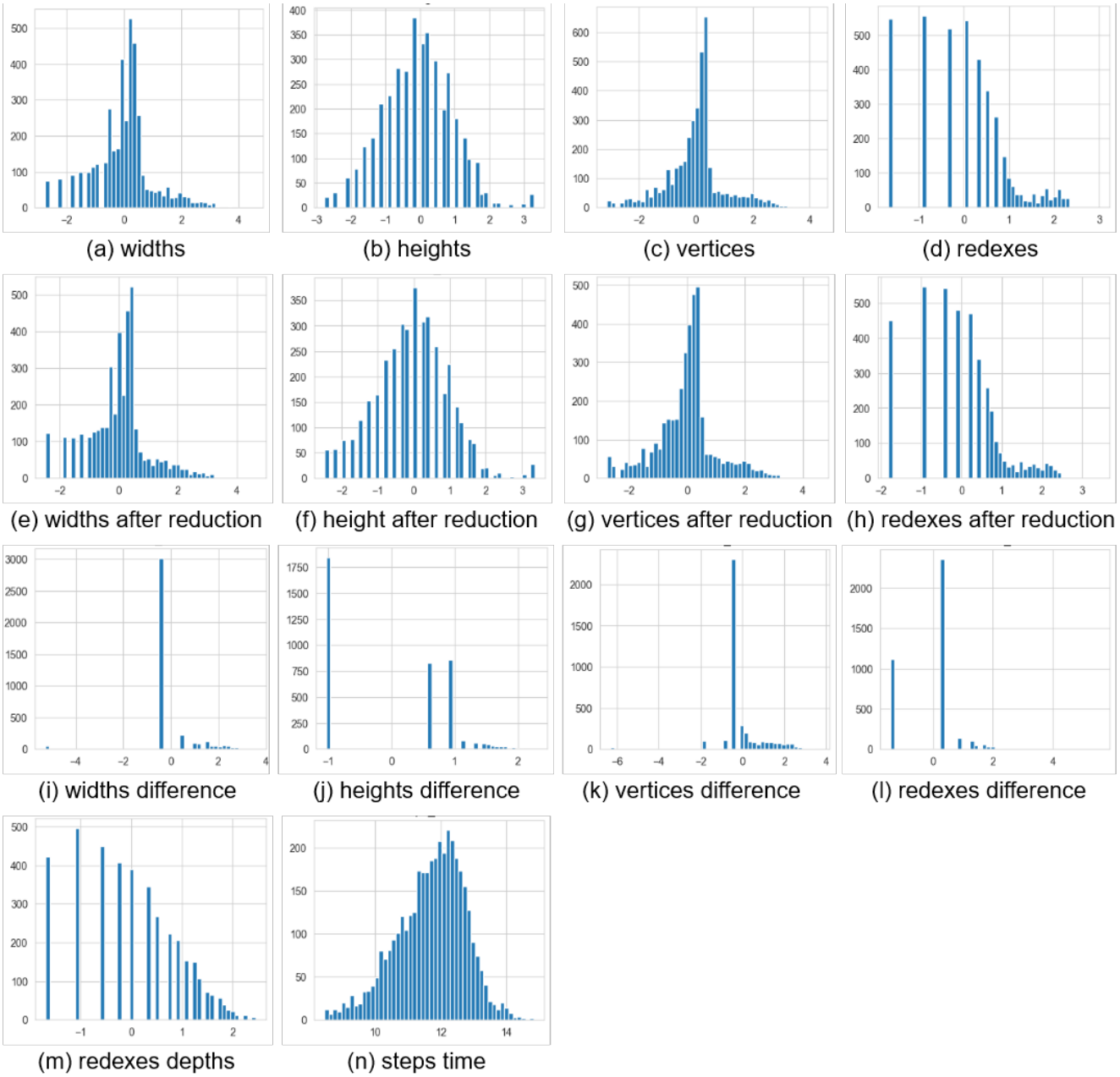


Рис. 2.9. Розподіл даних після застосування нормалізації Йео-Джонсона для всіх зібраних характеристик термів: (a) ширини, (b) висоти, (c) вершини та (d) редекси до редукції; (e) ширини, (f) висоти, (g) вершини (h) редекси після редукції; різниці (i) ширини, (j) висоти, (k) вершин та (l) редексів до та після редукції; (m) глибина редексів як ознака редексів; і цільова змінна (n) кроків час.

Звичайною практикою машинного навчання є розділення даних на підмножини для тренування, валідації та тестування, перед початком роботи з моделями.

Дані підмножини використовуються для тренування моделі, налаштування гіперпараметрів та тестування. Тестовий набір, що не використовувався раніше, забезпечує об'єктивність прогнозування. Дані термів розділені на тренувальну та тестову підмножини у пропорції 70%/30%. Валідація та налаштування гіперпараметрів виконані на тренувальному наборі через перехресну валідацію. Після розподілу датасету отримано 2782 та 1149 зразків відповідно. Розподіл даних у наборах ідентичний, тому точність прогнозу залежить від моделі, а не від розподілу даних.

Стандартні підходи.

На першому наборі даних (характеристик терму до редукції) було перевірено декілька стандартних алгоритмів машинного навчання для вирішення задачі регресії. В табл. 2.1 представлені результати експериментів, аналіз було проведено для таких методів: 1. Лінійна регресія, 2. Регресійні дерева рішень, 3. Регресія опорних векторів, 4. Регресія K-найближчих сусідів і 5. Регресія ШНМ. Найнижчі значення похибки (RMSE = 0.0497–0.2105) вирішення задачі регресії показали методи 2 і 4 на тренувальних даних, але на тестових даних значення похибки було відносно високим (RMSE = 0.30–0.4), це свідчить про низьку здатність до узагальнення цих алгоритмів в даному випадку. Методи 1, 3 і 5 на тренувальних даних показали вище значення помилки (RMSE = 0.27–0.29), але для тестових даних похибка була майже ідентичною (RMSE = 0.28), можна сказати, що в даному випадку алгоритми мають кращі можливості до узагальнення. Виходячи з отриманих результатів для подальших експериментів було обрано алгоритми 1 і 5.

Результати в табл. 2.1 показують, що алгоритми лінійної регресії та ШНМ мають високу здатність до узагальнення даних та невисоке значення показника RMSE 0.28–0.29 на тестовому наборі даних для першого датасету. Результати вказують на те, що ШНМ є ефективним інструментом з точки зору здатності апроксимації залежності часу кроку редукції від топології терму, а лінійної моделі цілком достатньо для вирішення проблеми прогнозування часу одного кроку редукції.

Таблиця 2.1.

Значення середньо квадратичних помилок, для різних алгоритмів, на трьох сформованих наборах даних.

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

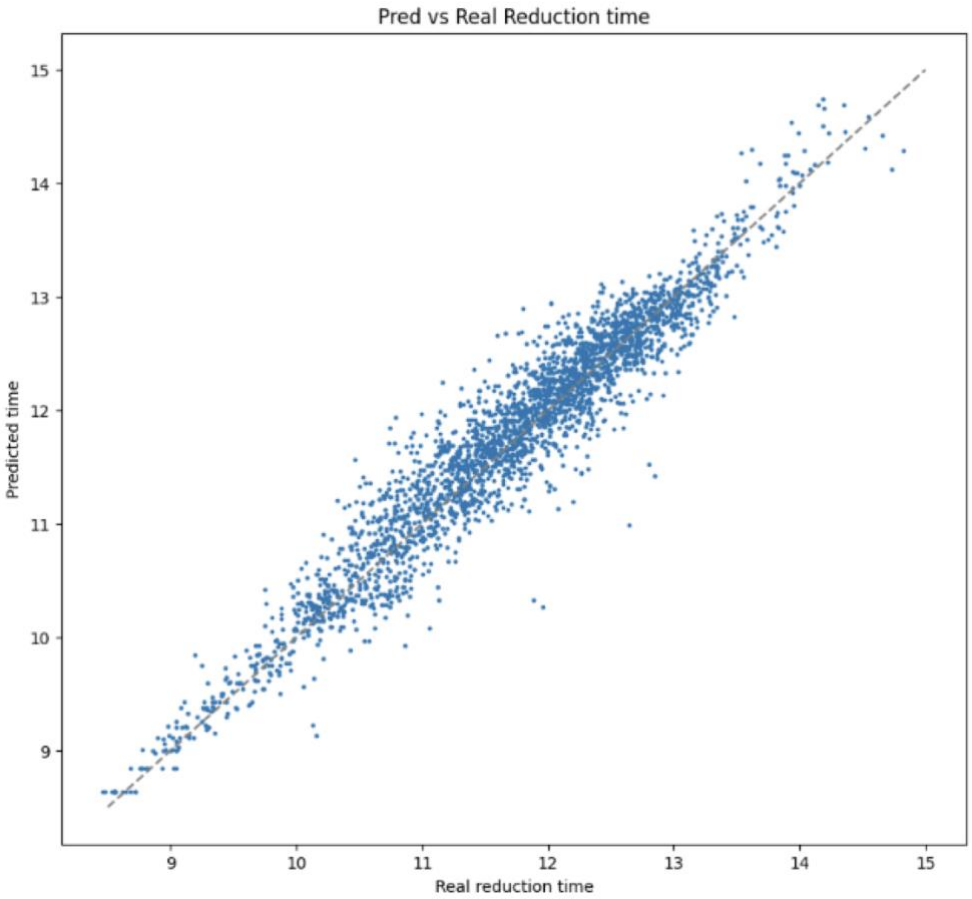


Рис. 2.10. Розподіл відношення логарифмів часу кроку редукції до оцінок, отриманих моделлю ШНМ на 1-му наборі даних, досягнутої на навчальному наборі даних.

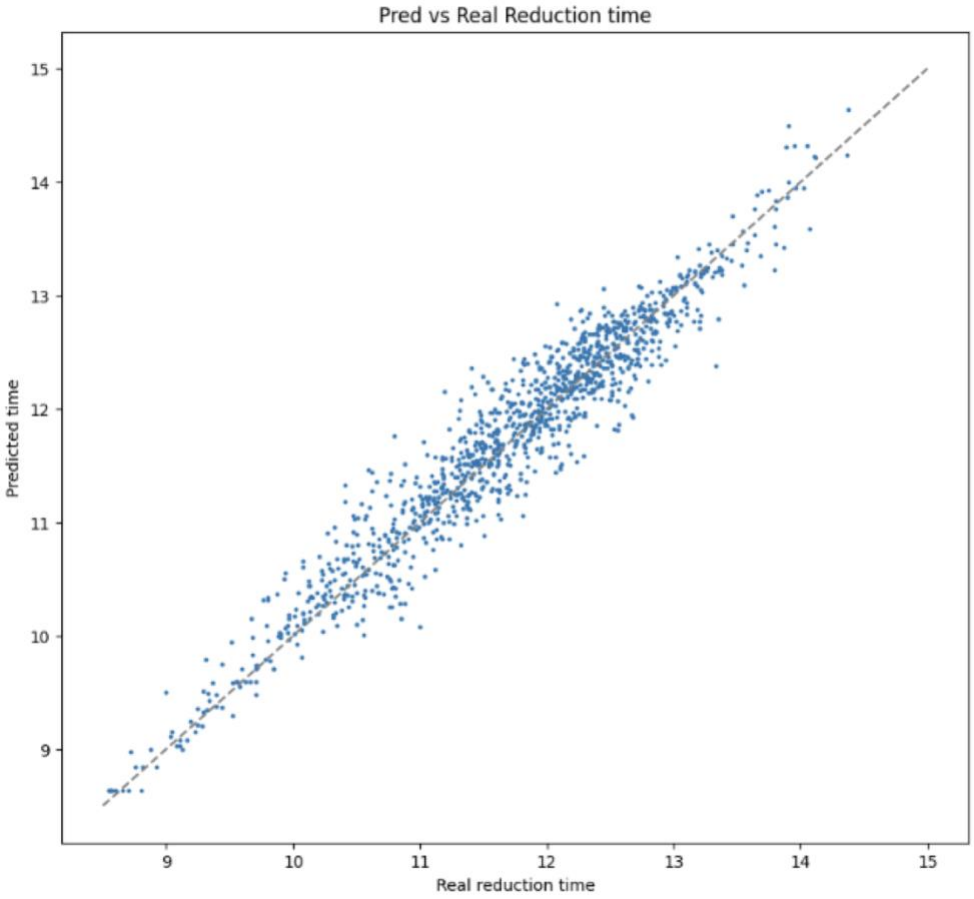


Рис. 2.11. Розподіл відношення логарифмів часу кроку редукції до оцінок, отриманих моделлю ШНМ на 1-му наборі даних, досягнутої на тестовому наборі даних.

Виходячи з попереднього досвіду було вирішено провести експерименти на 2-му та 3-му наборах даних для визначення складності одного кроку редукції, були отримані значення похибки в діапазоні 0.27–0.29. У висновку для алгоритму ШМН збільшення кількості вхідних характеристик позитивно впливає на точність прогнозу. На Рис. 2.10 та Рис 2.11 представлено порівняння прогнозів та очікуваних значень для алгоритму ШНМ на першому наборі даних. Графік будується за принципом розстановки точок, що представляють собою лямбда-терм з датасету, в просторі з осями, що відповідають реальним значенням цільової змінної для цього лямбда терму та значенню прогнозу алгоритму для цього лямбда-терму. Тож діагональ, позначена сірим кольором на малюнку, відповідає ідеальному співпадінню цільової змінної з прогнозом, то чим ближче точки знаходяться до

діагоналі – тим краще. Точки вище діагоналі представляють собою завищені прогнози, точки під діагоналлю – занижені прогнози. Для другого та третього наборів даних графіки для лінійної регресії та ШНМ є доволі схожими, що свідчить про те, що складніші алгоритми надають кращою апроксимації даних, це може свідчити про те, що природа залежності між характеристиками терму та часом одного кроку редукції є лінійною.

Таблиця 2.2.

Значення показників MSE, MAE і MAPE для моделей лінійної регресії та ШНМ для трьох наборів даних під час тренування та тестування.

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
2. ANN Regr. (MSE)	0.0858	0.0766	0.0788	0.0814	0.0766	0.0786
3. Lin. Regr. (MAE)	0.2276	0.2196	0.2284	0.2326	0.2244	0.2370
4. ANN Regr. (MAE)	0.22	0.2116	0.2158	0.2206	0.2141	0.2146
5. Lin. Regr. (MAPE)	1.958	1.894	1.9641	2.013	1.93	2.047
6. ANN Regr. (MAPE)	1.898	1.827	1.872	1.925	1.846	1.856

У табл. 2.2 представлені показники MSE, MAE і MAPE, для оцінки алгоритмів лінійної регресії та ШНМ на 1-му, 2-му та 3-му наборах даних. Аналіз результатів показав, що інші показники точності, не дає значних переваг, у питанні порівняння моделей за якістю, але деякі метрики, як MAE, що дає середнє значення помилки, може допомогти більш наглядно зрозуміти якість, та MAPE що представляє середнє значення помилки у відсотках, може допомогти зрозуміти точність з відносної сторони (Рис. 2.10 та Рис 2.11).

Результати експериментів показують, що моделі лінійної регресії, як і моделі ШНМ здатні ефективно прогнозувати час одного кроку редукції терму на основі

характеристик його топології. Виходячи з цього можна зробити висновок, щодо першого припущення щодо прогнозування часу редукції є релевантним.

З іншою сторони, друге припущення, щодо визначення складності та використання розширених наборів характеристик термів після редукції та різницевих значень, не показало значних покращень точності.

Тож можна зробити висновок, що ці параметри можуть допомогти з оцінкою найкращої стратегії редукції, але вони не забезпечують суттєвої точності, що є слабкою ознакою того, що параметри термів пов’язані зі складністю редукції термів.

Висновок до розділу 2

У розділі аналізується підхід до підвищення продуктивності стратегій редукції через змішування стратегій, ідея якого наближена до теорії ігор. Розглянуто два основних підходи: застосування змішування стратегій у чистому вигляді та використання рандомізованої стратегії для випадкового вибору редексів для редукції з тіла терму.

На основі аналізу в даному розділі можна зробити висновки, що змішування стратегій може ефективно замінити стратегії в чистому вигляді. Це підтверджується подібною продуктивністю з точки зору середньої кількості кроків, необхідних для редукції лямбда-термів, при цьому змішування стратегій частіше призводить до досягнення нормальної форми термів, якщо вона існує.

Цей розділ також аналізує підхід до прогнозування часу, необхідного для одного кроку редукції лямбда-терму. Заснований на ідеї, що, прогнозуючи час редукції для кількох редексів, можна вибрати стратегію, яка надаватиме перевагу найпростішому редексу згідно з прогнозом.

У цьому дослідженні було введено та застосовано графове представлення терму у вигляді дерева. В якості характеристик терму використано ширину, висоту, кількість вершин, а також кількість і глибину редексів, що підлягають редукції. На базі цих характеристик було створено три набори даних: перший містить показники

до редукції, другий – показники до і після редукції, а третій додатково включає різницю між показниками до і після редукції.

На даному етапі дослідження було розглянуто два підходи для прогнозування часу кроку редукції з використання показників тільки до редукції та визначення складності кроку редукції з використання показників після та різниць показників. Ідея першого є доволі прикладною, оскільки дозволяє обирати найпростіший редекс, а ідея другого є більш теоретичною, оскільки на практиці невідомі показники до та різницеві значення до моменту редукції, також ідея підходу є в вивченні залежностей та їх характеру для використання в подальшому процесі вивчення.

Цей етап дослідження відкриває шлях для глибшого застосування методів машинного навчання у вивченні лямбда-числення, особливо в контексті процесу редукції. Як наступний крок, слід розглянути можливість прогнозування загальної кількості кроків редукції лямбда-терму до нормальної форми. Стратегія, заснована на прогнозуванні часу одного кроку редукції та виборі найпростішого редексу, є жадібною, і може не забезпечувати зменшення загального часу редукції через складність структури лямбда-термів. Також важливо провести кластерний аналіз множини згенерованих термів, щоб виявити існуючі залежності та можливі підгрупи термів зі схожими характеристиками.

РОЗДІЛ 3.
ОЦІНКА КІЛЬКОСТІ КРОКІВ РЕДУКЦІЇ ЗА ПЕВНОЮ СТРАТЕГІЄЮ МЕТОДАМИ МАШИННОГО НАВЧАННЯ

3.1. Постановка проблеми оцінки кількості кроків редукції лямбда-термів за заданою стратегією

Методи оптимізації стратегій редукції раніше включали або нові стратегії, як UR і Mixed Reduction, або розробку стратегій, що враховують обчислювальні витрати через оцінку часу кроку редукції терму. Ці підходи не враховували можливість вибору стратегії для конкретних термів, і точність оцінки може варіюватися. Наприклад, можна застосовувати машинне навчання для автоматизації вибору стратегії редукції або для точнішого визначення причин вибору стратегії, враховуючи обчислювальні витрати або кількість кроків редукції.

В роботі було використано два типи представлення термів: спрощене, де всі змінні замінюються на “x”, та повне, що зберігає всі оригінальні імена змінних. Спрощене представлення застосовувалося для зменшення обчислювального навантаження, оскільки one-hot-encoding може значно збільшуватися з числом змінних. Припускалося, що спрощеного представлення буде достатньо для вирішення задачі. В подальших експериментах використано повне представлення лямбда-термів та нову архітектуру ШНМ для обробки текстової інформації, що дозволило ефективно використовувати його як вхід.

У цьому розділі використано три архітектури для роботи з текстом: LSTM, CNN і Transformer. Мета – прогнозування кількості кроків редукції для лямбда-терму, використовуючи текстове представлення, яке кодується з використанням one-hot-encoding. Задача може бути розглянута як регресія, класифікація (31 клас) або задача з корзинами. Терми відфільтровані за максимальною кількістю кроків (30). Використано попередньо натреновану модель з доданим повнозв’язним шаром як класифікатором. Замість Microsoft CodeBERT було використано вбудовування, і проаналізовано повне представлення терму з усіма змінними.

Основна увага – прогнозування кількості кроків редукції лямбда-терму при використанні обраної стратегії.

Можливість робити прогнози кількості кроків редукції лямбда-терму для різних стратегій редукції з достатньою точністю дозволяє обрати стратегію, яка відповідно до прогнозу, є більш ефективною для даного терму. В даній роботі була використана архітектура моделі Transformer. Даний підхід випливає з попередніх досліджень що показали свою ефективність у вирішенні математичних задач [65], виконання коду та оптимізації компіляції коду [66, 67], в даних дослідженнях було використано великі мовні моделі на базі архітектури Transformer. Також перспективні результати були досягнені при вирішенні проблеми прогнозування типу терму, описана проблема вимагає глибокого розуміння терму, що є ключовим. Деякі з робот були сфокусовані на дослідженні впливу стратегій редукції на зменшення кількості редексів [48, 49, 68] У дослідженні [50] автор розглядає обчислювальні аспекти стратегій редукції. Однак жоден із них не намагається виділити певні ознаки термів, які сигналізують про перевагу одних стратегій над іншими. Також у даному дослідженні [50] було проаналізовано обчислювальні аспекти стратегій редукування. Проте жодна з розглянутих робіт не фокусується на розумінні продуктивності стратегій відносно конкретного терму.

3.1.1. Гіпотези дослідження

Основною ідеєю прогнозування кількості кроків редукції терму за заданою стратегією – є можливість у подальшому використовувати отриманий прогноз для автоматичного вибору більш оптимальної стратегії, та розробити компілятор на основі даного підходу. В якості методу дослідження цього процесу було обрано ШНМ, які мають де-факто властивість збору статистики [69]. Оскільки стандартний лямбда-терм, як і функціональна програма являє собою послідовність ключових слів, операторів і змінних, тому в дано випадку було обрано актуальні методи [70, 71, 72] для роботи з текстовими послідовностями: CNN, LSTM і моделі на базі архітектури Transformer.

В даному дослідженні в якості навчальних і тестових даних було використано синтетично згенерований датасет, оскільки синтетичні дані, з точки зору даного дослідження мають ряд переваг:

1. Вони є легшими з точки зору збору, порівняно з природними;

2. При генерації можуть бути налаштовані на те щоб одразу належати якійсь підмножині (наприклад належати одному розподілу співвідношень абстракцій та аплікацій);

3. Не потребують спеціального налаштування для виділення ознак.

Даний підхід дозволяє проводити тренування моделей з урахуванням широкого набору типів термів. Однак з іншої сторони вони можуть не охоплювати всіх реально існуючих комбінацій, через дослідження термів, що не розглядаються в реальності достатньо часто.

Далі описані проблеми, що мають бути вирішені завдяки використанню передових моделей штучного інтелекту та синтетично згенерованого набору лямбда термів.

1. **Генерація набору даних лямбда-термів необхідного розміру** для забезпечення стабільного навчання моделей, що можуть бути пізніше протестовані на реальних даних та матимуть співвідносний рівень помилок.

2. **Налаштування гіперпараметрів** та **навчання моделей** машинного навчання для отримання найнижчого рівня помилок в задачі прогнозування кількості кроків редукції для обраної стратегії.

3.1.2 Спрощене представлення лямбда-термів

У дослідженні використовується спрощене представлення лямбда-терму, де всі змінні замінені на «*x*» для зменшення розміру вектору при one-hot-encoding. Використані моделі: CNN, LSTM і BERT (Transformer). Ці моделі були натреновані з нуля на згенерованих даних лямбда-термів.

Основною проблемою роботи з лямбда-термами є їх нескінченна кількість змінних. Можливе рішення – обмежити кількість змінних до 64, 128 або 512, оскільки це може значно зменшити розмір вхідних даних, параметрів моделі та

обчислювальних ресурсів [73]. Також, однакові терми з різними іменами змінних можуть помилково вважатися різними термами. Тому прийнято рішення використовувати спрощене представлення, де всі змінні замінюються на «*x*». Це дозволяє зберегти структуру терму та його внутрішні зв'язки, зменшуючи об'єм обчислювальних ресурсів. Припускається, що спрощене представлення буде достатнім для моделей штучного інтелекту для прогнозування кількості кроків редукції стратегій LO та RI.

Підхід до вирішення задачі

Одним із питань даної роботи є вибір правильного представлення цільової змінної з точки зору моделі. В ході роботи було сформульовано та проаналізовано три можливі підходи [73]:

– *Розв'язання у вигляді задачі регресії*. Основним плюсом даного підходу є простота та, в деякому плані природність. Кількість кроків витрачених на редукцію лямбда терму дійсно може бути від 0 до нескінченності, але треба пам'ятати, що кількість кроків може бути лише цілим значенням більше нуля. Тут виникає можлива несумісність із виходами моделі, оскільки модель може прогнозувати реальні значення від мінус нескінченності до плюс нескінченності. Тож наступним постає питання метрики, та питання необхідності округлення чисел з плаваючою точкою до цілих і правила такого округлення.

– *Розв'язання у вигляді задачі класифікації*. Даний підхід враховує особливості того, що кількість кроків має бути цілим числом, в то же час при використанні даного підходу необхідно обмежити кількість кроків для термів, що будуть використані для аналізу. Також постає питання вибору правильної функції втрат для нейронної мережі, та в той же час підбір або розробка правильних метрик, що можуть переводити представлення класів у числові та аналізувати їх на рівні кроків, а не класів.

– *Розв'язання у вигляді бінової задачі*. Даний підхід є дуже близьким до розв'язання проблеми у вигляді класифікації. Якщо бути точним, то проблема класифікації є підтипом проблеми з бінами, де розмір біну становить 1. Тож в

проблемі з бінами можлива кількість кроків розбивається на декілька таких як 0-4, 5-9, 10-14 і так далі. Дане рішення в теорії має дозволити спростити задачу для моделі, у випадку якщо задача класифікації у чистому вигляді є занадто складною. Тут також постає питання подальшої обробки виходів моделі та підбір правильної метрики для можливості наглядно оцінити якість прогнозування.

Взявши до уваги розподіл кількості кроків редукції термів, що в більшості випадків знаходиться в межах від 0 до 30 кроків (Рис. 3.7) було обрано варіант роботи з цією проблемою, як проблемою класифікації. Також варто зазначити, що класична метрика точності (accuracy) не підходить в даному випадку, оскільки для цієї метрики помилка у класах 29 і 30 є рівнозначною помилці між 0 і 30.

За основу було взято декілька класичних метрик для проблеми регресії [74] таких як абсолютна середня помилка (MAE), що вимірює абсолютну різницю між прогнозованою кількістю кроків редукції та реальним значенням, дана метрика дозволяє доволі наглядно зрозуміти точність моделі. Також в роботі було використано корінь середньоквадратичної похибки (RMSE), що є коренем квадратним різниці квадрату прогнозованої кількості кроків та реальної кількості кроків. Було проаналізовано залежність між реальною кількістю кроків та показником MAE як показано на Рис. 3.12–3.14, це було зроблено виходячи з гіпотези про те, що терми з більшою кількістю кроків матимуть більше значення помилки.

3.2. Моделі машинного навчання для прогнозування кількості кроків редукції терму

3.2.1. Аналіз моделей для прогнозування кількості кроків редукції лямбда-термів за обраною стратегією

1. **Hidden Markov Models (HMM)** – статистичні моделі для процесів з прихованими станами і марковськими переходами між ними. HMM використовують ймовірності переходів і розподіли виходу для моделювання спостережуваних послідовностей, де токени відображають стани [75, 76]. *Переваги*: стійкість до змін у часі, ефективне моделювання послідовних даних, просте

навчання. *Недоліки*: припущення про незалежність станів, обмежена контекстна пам’ять, складність масштабування.

2. Метод опорних векторів (SVM) – алгоритм для класифікації та регресії, який шукає гіперплощину для максимального розділення класів, максимізуючи відстань між опорними векторами кожного класу. Нелінійна класифікація можливе завдяки функціям ядра, які працюють у багатовимірному просторі без явного обчислення координат [77]. *Переваги*: ефективність у великих просторах, універсальність для різних типів даних і завдань, ефективне використання пам’яті [78]. *Недоліки*: складність вибору ядра, погана масштабованість, низька точність при перекритті класів [78].

3. Системи на основі правил або експертні системи використовують задані людиною правила для прийняття рішень на основі знань або даних, імітуючи прийняття рішень експерта через прямий або зворотний ланцюг [79, 80]. *Переваги*: пояснюваність, узгодженість результатів, простота модифікації [80]. *Недоліки*: обмежена гнучкість, складність обслуговування та масштабування [80].

4. ШНМ – основна модель машинного навчання, натхненна біологічними нейронними мережами. Вона складається з шарів взаємопов’язаних вузлів (нейронів), які виконують прості обчислення. Вузли розташовані в вхідному, в одному або кількох прихованих і у вихідному шарах, з'єднані зваженими зв’язками, які налаштовуються під час навчання. *Переваги*: гнучкість, адаптивність, здатність до навчання і узагальнення. *Недоліки*: високі обчислювальні вимоги, ризик перенавчання, непрозорість процесів [70–72].

3.2.2. Моделі машинного навчання для обробки тексту

Дослідження розглядає текстове представлення лямбда-термів. На сьогоднішній день існує безліч методів і моделей машинного навчання для роботи з текстовою інформацією. Дані методи можна розділити на дві групи: прості та складні. До простих можна віднести метод опорних векторів, повнов’язні штучні нейронні мережі з малою кількістю шарів, наївні байєсівські мережі та інші, до складних методів можна віднести архітектури глибинного машинного навчання,

зокрема CNN, рекурентні нейронні мережі (RNN) і моделі на базі архітектури Transformer. [81–83]. На Рис. 3.1 [83] продемонстрований приклад використання моделі CNN для роботи із задачами класифікації тексту. Для розв'язання простих текстових задач, наприклад, класифікації, моделі CNN пропонують надійні рішення. CNN може виконувати роль індикатора слів, вказуючи на наявність необхідного слова чи фрази в тексті.

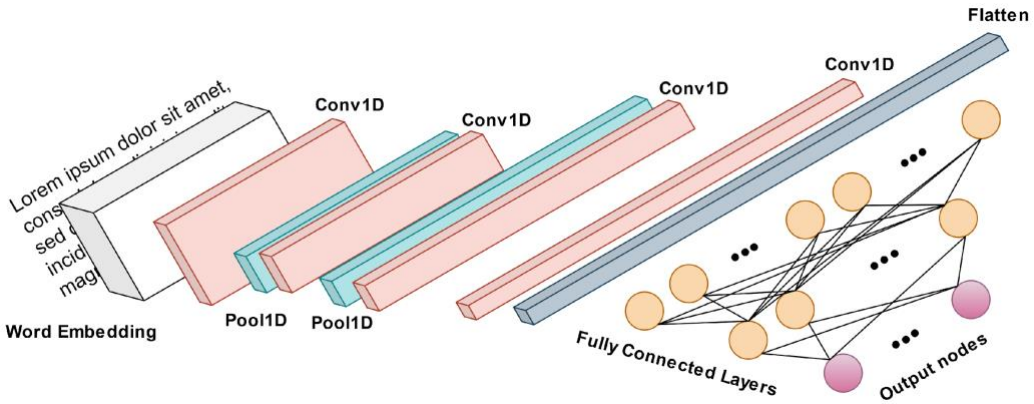


Рис. 3.1. Архітектура моделі CNN для вирішення задачі класифікації тексту [83].

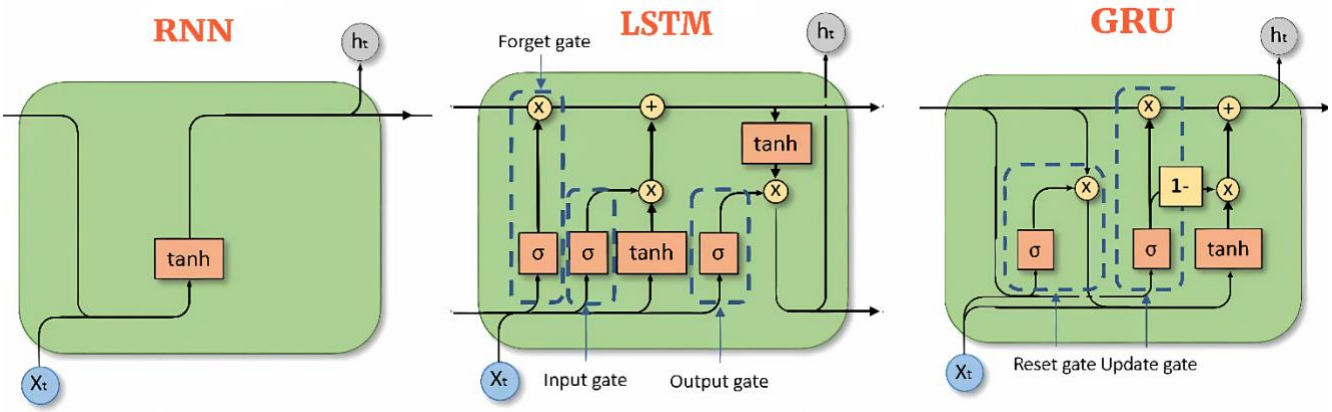


Рис. 3.2. Схема блоків RNN, LSTM та GRU [84].

На Рис. 3.2 продемонстровано іншу архітектуру обробки тексту [84]. Тут представлені вузли RNN, LSTM та стробованої рекурентної одиниці (GRU). Основний принцип роботи RNN, LSTM і GRU дуже схожий. Кожен з вузлів працює над деякою частиною тексту, зазвичай на рівні символів або слів. Також кожен з вузлів для збереження цілісного представлення тексту може використовувати свою

пам'ять або вихід. Дане представлення зберігається у вигляді вектору і називається вбудовуванням.

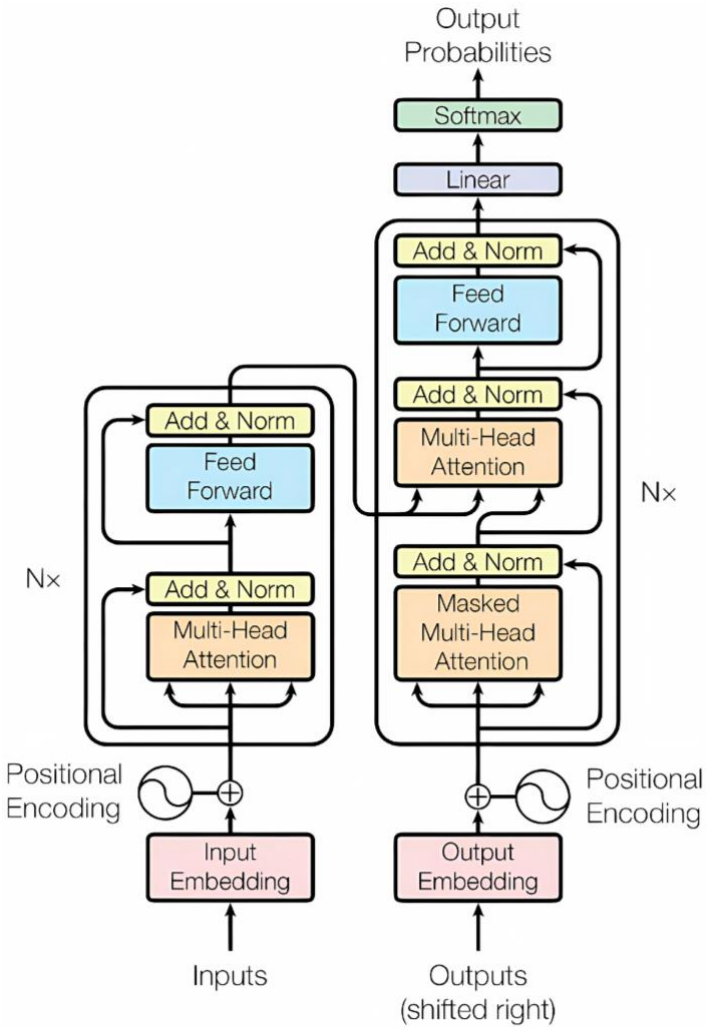


Рис. 3.3. Архітектура моделі Transformer [81].

Описаний підхід спрощує роботу з текстовими даними, дозволяючи вирішувати задачі класифікації, генерації та перекладу тексту. Хоча архітектури на базі Transformer значно покращують результати порівняно з CNN, вони можуть мати проблеми із занадто довгими послідовностями і зникненням градієнта, як у RNN [84].

Архітектура Transformer [81, 82], представлена на Рис. 3.3 [81], обробляє дані паралельно, подібно до CNN, з фіксованим розміром вхідного вектору. Основний

інструмент – механізм самоуваги, який фіксує довгострокові залежності і забезпечує високу продуктивність у роботі з текстом [84].

1. Згорткові нейронні мережі (CNN) спеціалізуються на обробці структурованих даних, таких як зображення. Вони використовують операцію згортки, ковзаючи фільтр по даним для створення карти ознак, яка відображає наявність особливостей або шаблонів. Процес повторюється на кількох рівнях, створюючи ієрархію складніших функцій [70]. *Переваги*: ефективність для однотипних даних (зображень), оптимальне використання ваг через фільтри, стійкість до позицій і варіацій характеристик. *Недоліки*: потреба в однотипних даних, високі обчислювальні вимоги, непрозорість процесів..

2. Мережі довготривалої короткочасної пам'яті (LSTM) – це тип RNN, що вирішує проблему довгострокових залежностей, яка є у стандартних RNN. LSTM використовують комірку пам’яті (cell) для зберігання інформації та ворота (gate) для контролю її потоку, що робить їх ефективними для послідовностей, таких як мова та текст [71]. *Переваги*: пам'ять про попередні дані, ефективність у обробці послідовностей, зменшення проблеми зникаючого градієнту. *Недоліки*: висока обчислювальна складність, неможливість паралелізації обчислень.

3. Трансформери – це архітектура глибнного навчання, представлена в статті Васвані та ін. у 2017 році [81]. Вони використовують самоувагу для зважування значень слів у послідовності, незалежно від їхньої відстані, що покращує розуміння контексту і дозволяє паралельну обробку послідовностей, на відміну від RNN і LSTM [72]. *Переваги*: ефективне розпаралелювання, покращене розуміння довгострокових залежностей, масштабованість. *Недоліки*: високі обчислювальні вимоги, схильність до перенавчання, складність архітектури.

3.2.3. Обрані моделі для проведення експериментів

Для експериментів було обрано кілька архітектур ШНМ для перевірки гіпотези про прогнозування кількості кроків на основі текстового представлення терму. Використано три різні архітектури ШНМ, перевірено їх налаштування та конфігурації гіперпараметрів.

Моделі глибинного навчання складаються з трьох частин: вхідна частина для отримання та попередньої обробки даних; шари виділення ознак, що перетворюють дані на семантичне представлення у прихованому просторі; повнозв'язні шари для перетворення ознак на форму виходу (клас, число тощо) [73].

Всі моделі використовують описаний підхід і відрізняються частиною виділення ознак (feature-extractor), яка залежить від архітектури (CNN, LSTM, Transformer). Зміна цієї частини впливає на принцип побудови та складність вектору ознак, або вбудовування.

Налаштування повнозв'язної частини визначає тип виходу і задачі. Налаштування також важливе для аналізу вектору ознак. На Рис. 3.4 показано налаштовану згорткову модель.

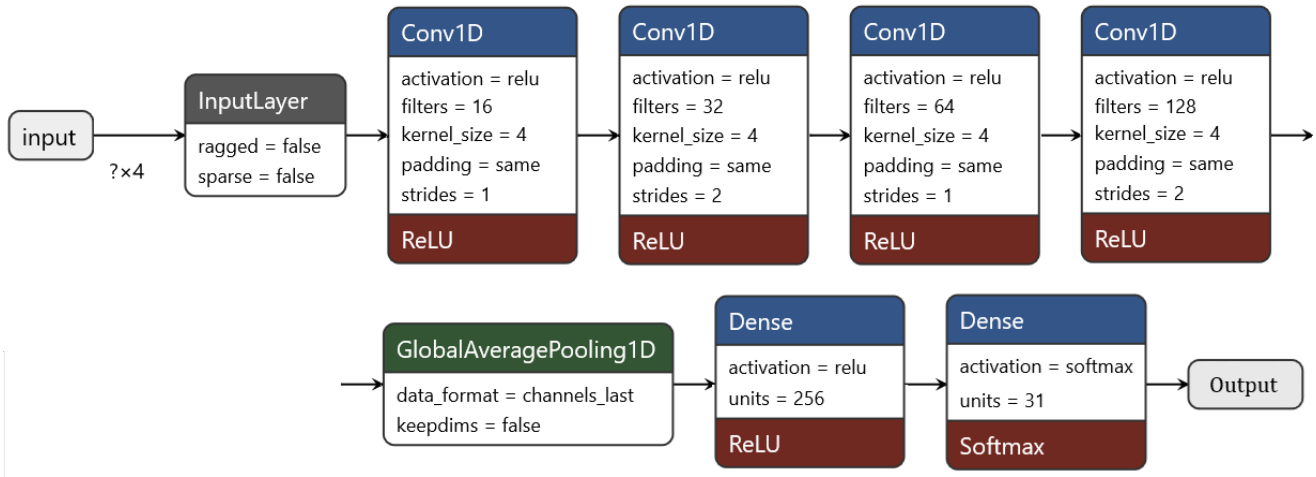


Рис. 3.4. Архітектура згорткової моделі ШНМ, що використовується для оцінки кількості кроків редукції терму за обраною стратегією.

Для роботи з послідовностями розглянуто два підходи:

– Перший: використання фіксованої довжини термів із обрізанням або доповненням нулями. Фіксована довжина термів дозволяє навчання партіями, проте є втрата інформації при обрізанні і можливе зашумлення при додаванні нулів.

– Другий: використання змінної довжини терму без зміни. Довжина виходу коригується за допомогою шару GlobalAveragePooling1D для вирівнювання розмірності [73], що продемонстровано на Рис. 3.4.

Архітектура розробленої моделі CNN складається з чотирьох шарів Conv1D, в деяких використано розмір кроку (stride), що дорівнює 1, в деяких – 2, даний підхід дозволяє шар Conv1D з шаром зменшення розмірності Pool1D.

Основну ідею архітектури можна описати як вилучення ознак з лямбда-термів, представлених одновимірними векторами довжини N, і їх перерозподіл у 128-розрядному векторі. Подібно до CNN, ідея полягає у зменшенні ширини вектору на кожному рівні мережі при підвищенні семантичного контексту ознак. Наступний рівень використовує повнозв'язний шар розміром 256 для обробки отриманих ознак. Фінальний рівень містить 31 нейрон з активацією Softmax для класифікації кроків редукції від 0 до 30.

Модель ШНМ на основі LSTM, що використана в експериментах, зображена на Рис. 3.5. Як і модель на основі CNN, модель LSTM має два варіанти: з фіксованою довжиною входу та без.

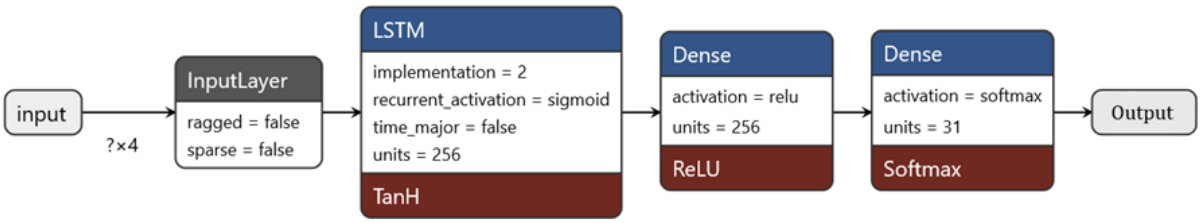


Рис. 3.5. Архітектура моделі ШНМ на основі блоку LSTM, що використовується для оцінки кількості кроків редукції терму за обраною стратегією.

Основною моделлю обрано варіант без фіксації довжини, що дозволяє працювати з послідовностями різної довжини і зменшує кількість ваг, зокрема розмір моделі. Експерименти показали, що блок LSTM з 256 нейронами достатній для вилучення ознак і збереження стану. Аналогічно до CNN, використано таку ж повнозв'язну частину. Також була використана модель Bidirectional Encoder Representations from Transformers (BERT) [72], що є варіацією архітектури

Transformer і функціонує як автокодувальник з фіксованою максимальною довжиною входу. Обмеження довжини вхідної послідовності встановлено на 512 токенів, що враховує архітектуру класичної моделі BERT [72].

BERT обробляє вхідні дані за допомогою трьох типів векторів для кожного токену.

– Токени вбудовувань: представляють вхідні токени як вектори через матрицю вбудовування, забезпечуючи щільне представлення на основі вивчених вбудовувань.

– Вбудовування сегментів: використовуються для розрізнення двох виразів у парі, допомагаючи BERT розуміти межі і зв’язки між виразами.

– Позиційні вбудовування: додаються для надання інформації про порядок токенів у послідовності, оскільки механізм самоуваги в Transformers не фіксує порядок послідовності.

Вбудовування підсумовуються, щоб утворити остаточні вхідні вбудовування для архітектури BERT, що дозволяє їй ефективно обробляти контекст і зв’язки в тексті. Архітектура BERT має два прихованих шари по 84 нейрони, проміжний шар з 64 нейронів і 6 головок самоуваги, де кожна головка фокусується на різних особливостях даних. Вихідний результат – матриця 512×84, що зменшується до 10×84 завдяки AveragePooling1D з кроком 50. Далі матриця перетворюється в одновимірний вектор розміром 840 за допомогою шару Flatten і проходить через шар Dropout з коефіцієнтом 0.1. Фінальний повнозв’язний шар – Dense з 31 нейроном для класифікації кількості кроків (Рис. 3.6).

В табл. 3.1. наведено остаточну складність моделей та їх кількість параметрів для навчання. Найбільша кількість вагових коефіцієнтів представлена у моделі LSTM. Кажучи про моделі CNN, її ваги рівномірно розподілені між екстрактором ознак та повнозв’язними частинами. Даний факт можна пояснити специфікою екстрактора ознак (feature-extractor) на основі згортки та необхідністю аналізу згорткових ознак. Через доволі складну архітектуру та високу ефективність

екстрактору ознак (feature-extractor) моделі на базі Transformer, в даному випадку було використано найменший за розміром екстрактор ознак.

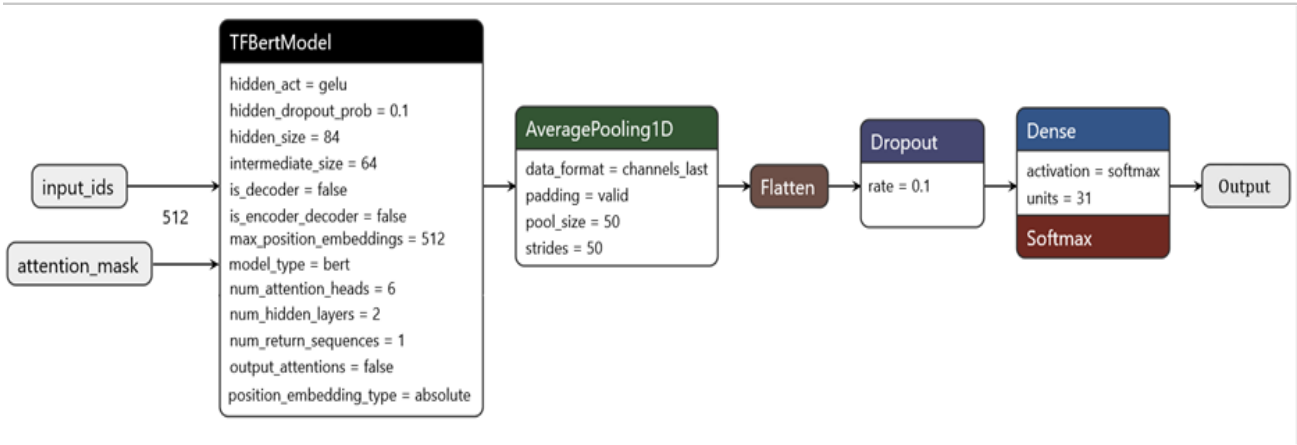


Рис. 3.6. Архітектура моделі ШНМ на основі моделі BERT, архітектури Transformer, що використовується для оцінки кількості кроків редукції терму за обраною стратегією.

Таблиця 3.1.

Порівняння ваг розроблених моделей ШНМ.

	Convolution based	**LSTM based**	**Transformer based**
Feature extractor weights	43504	267264	130832
Fully connected weights	267264	73759	26071
All trainable weights	130832	341023	156903

3.3. Проведення експериментів із оцінки кількості кроків редукції лямбда-термів за заданою стратегією

Підготовка даних для навчання та перевірки отриманих моделей.

Для збільшення розміру датасету термів, всі терми спочатку редуковані за стратегією LO для забезпечення нормальної форми. Потім терми спрощені, і підраховується кількість кроків редукції за стратегіями LO і RI. Подібні терми фільтруються, і для кожної стратегії редукції формуються набори даних, з яких

видаляються терми з більше ніж 30 кроками редукції. Додатково введено більше фільтрів для ускладнення набору даних і створення різноманітніших термів.

Генерація та збір навчальних наборів (табл. 3.2) були розраховані відповідно до закону Chinchilla, який вимагає, щоб кількість токенів у навчальному наборі була щонайменше в 20 разів більша за кількість ваг моделі Transformer [85]. Модель має 156 тис. ваг, тому навчальний датасет має бути не менше 3 млн токенів. Згенерований датасет містить 3.8 млн токенів, що достатньо для навчання моделей. Рис. 3.7 показує розподіл довжин термів для двох наборів даних, створених для стратегій LO та RI. Набори сформовані з тих самих термів, але терми, які не редуковані за RI, були виключені. Набір за стратегією LO має більше термів з довжиною понад 200 (див. Рис. 3.7).

Таблиця 3.2.

Набори даних для тестування та тренування моделей ШНМ в задачі передбачення кількості кроків редукції для стратегій LO та RI

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

У дослідженні проаналізовано розподіли кількості кроків редукції для стратегій LO та RI (Рис. 3.8). Виявлено, що хоча графіки схожі, терми для стратегії LO зазвичай мають трохи більше кроків (Рис. 3.8.а). Для експериментів створено два набори даних: один для LO, інший для RI. Більшість термів у наборах однакові, але терми, що перевищують встановлену кількість кроків редукції, вилучаються з відповідних наборів.

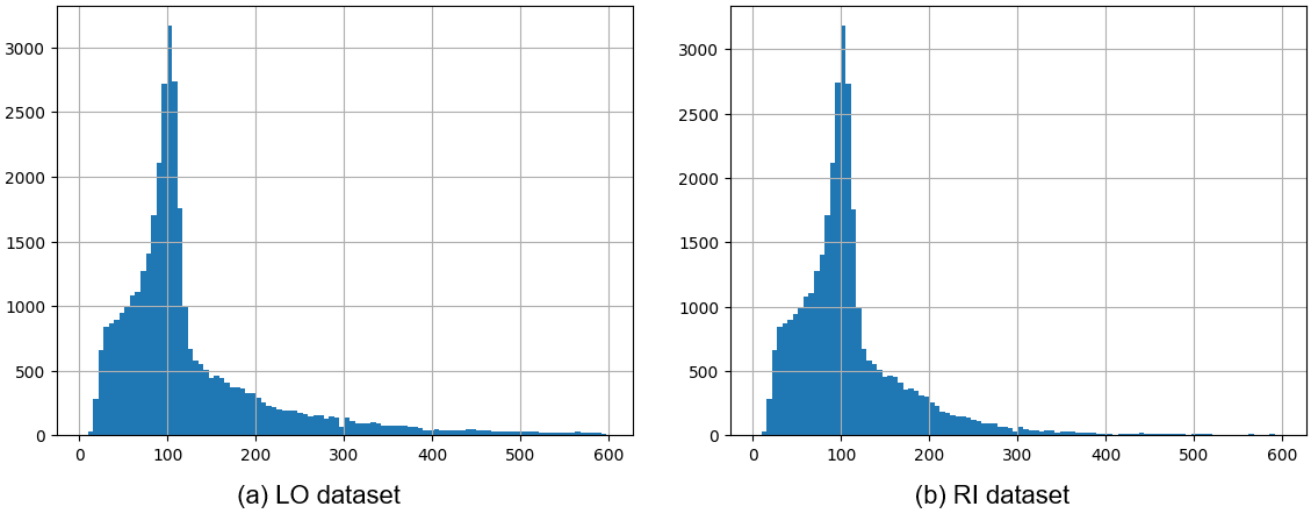


Рис. 3.7. Розподіл довжини послідовності термів для (a) набору даних LO, (b) набору даних RI.

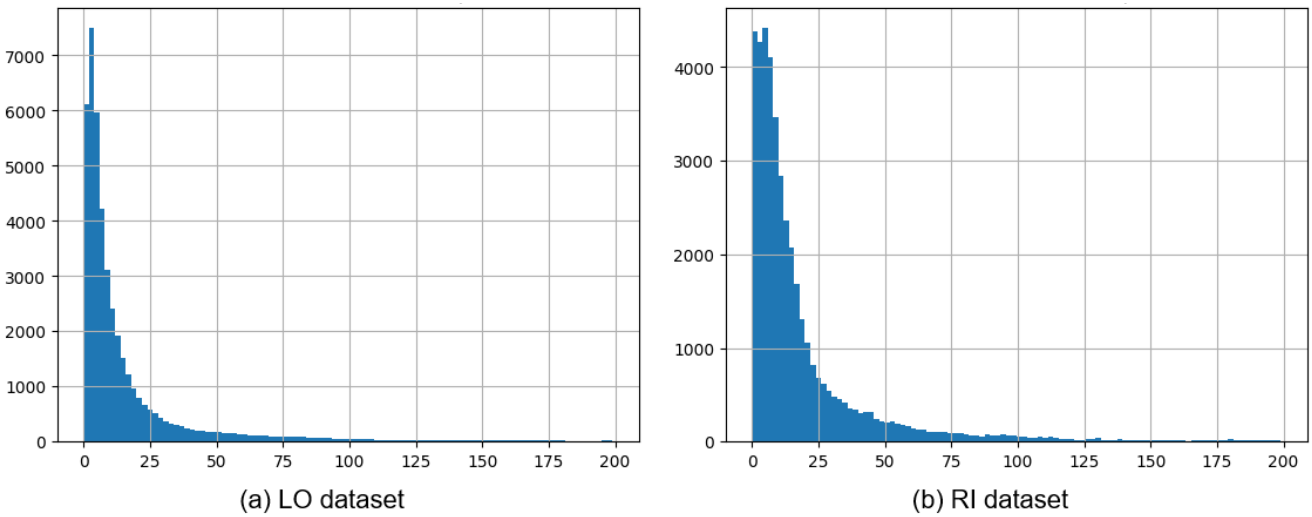


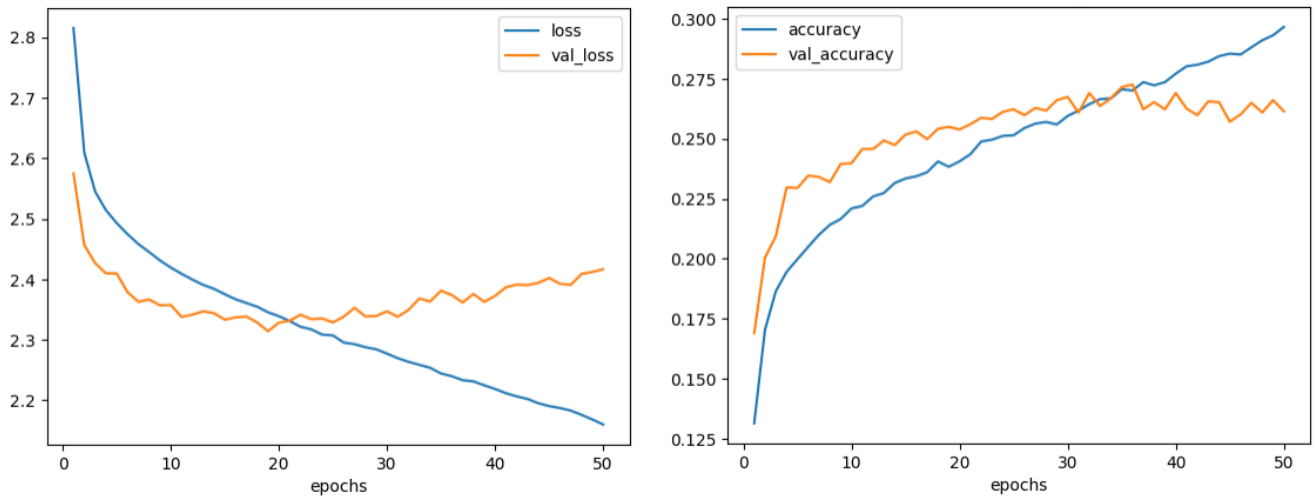
Рис. 3.8. Розподіл кількості кроків редукції для (a) набору даних LO та (b) набору даних RI.

3.3.1. Навчання моделей глибинного навчання

Після порівняння декількох можливих підходів було обрано вирішувати задачу як проблему класифікації, використовуючи категоріальну перехресну ентропію як функцію витрат. Під час тренування моделі застосовано підхід з навчанням з контрольними точками, що дозволяє зберігати ваги мережі при

покращенні метрик на валідаційному наборі, забезпечуючи найкращу версію моделі незалежно від подальшого перенавчання або погіршення метрик.

В експериментах використовувався оптимізатор Adam, популярний алгоритм для навчання моделей глибокого навчання. Adam поєднує переваги AdaGrad і RMSProp, розраховуючи адаптивну швидкість навчання для кожного параметра, що допомагає обробляти розріджені градієнти та нестаціонарні цілі. Це робить його ефективним для великих наборів даних і моделей на базі Transformer [72, 85, 86].



(a) Model loss (b) Model accuracy

Рис. 3.9. Навчальні криві моделі на основі архітектури Transformer на наборі даних LO: (a) крива втрат, (b) крива точності.

В ході даного етапу дослідження було навчено 6 моделей: для кожного із побудованих датасетів (для LO та RI стратегії) було проведено тренування кожного з трьох типів архітектур (CNN, LSTM, Transformer). На Рис. 3.9 показані типові криві процесу навчання моделей для тренувального та тестового датасету. Рис. 3.9.a представляє значення функції втрат, а Рис. 3.9.b представляє значення точності. Криві демонструють процес навчання архітектури моделі на базі Transformer для прогнозування кількості кроків редукції на наборі побудованому для стратегії LO.

На графіках точності та функції втрат (Рис. 3.9) видно момент, коли криві для тренувального та валідаційного набору починають розходитися, що сигналізує про

початок перенавчання моделі. Це означає, що модель більше запам'ятовує правильні відповіді, ніж вивчає залежності між архітектурою термів і кількістю кроків. Для уникнення перенавчання було використано метод EarlyStopping: ваги моделі зберігаються на моменті початку перенавчання, щоб забезпечити найкращу точність. Цей підхід застосовано для всіх шести моделей.

3.3.2. Результати оцінки кількості кроків редукції лямбда-термів за заданою стратегією з використанням моделей глибинного навчання

Як було описано раніше, в даному дослідженні ефективність моделі була досліджена не лише з використанням показника точності, що є класичною для проблеми класифікації. Треба пам'ятати, що в даному випадку класами є кількість кроків редукції, тому додатково також були використані метрики MAE та RMSE, що дозволяють краще зрозуміти ефективність моделі з точки зору розв'язання проблеми регресії. Відповідні результати для моделей архітектур CNN, LSTM та Transformer навчених на даних стратегії LO наведено в табл. 3.3.

Таблиця 3.3.

Результат навчання обраних моделей на наборі даних LO.

	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

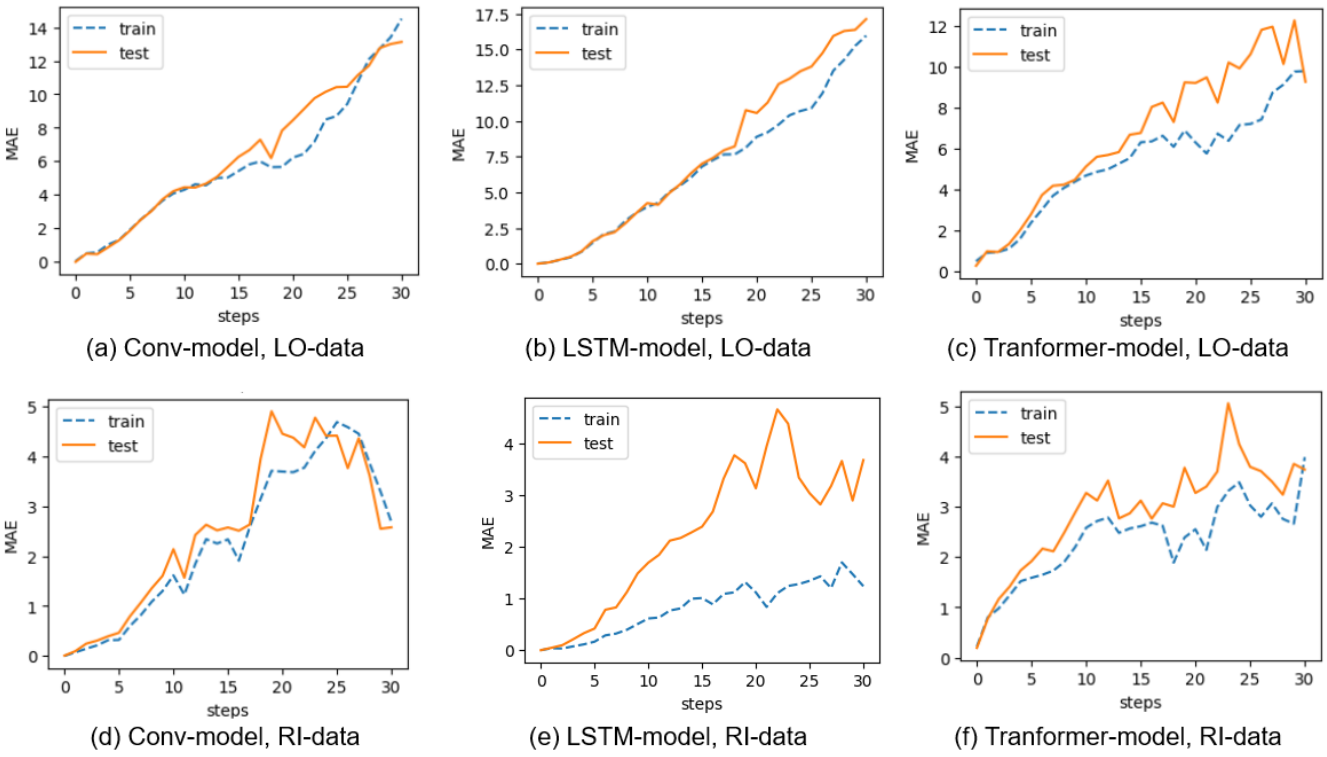
Модель на основі LSTM демонструє високу точність, але показники MAE та RMSE свідчать, що якість прогнозу не перевищує інші моделі. LSTM робила менше помилок у класифікації, але з більшим розкидом (наприклад, правильна відповідь – 2, прогноз – 20). Високий розкид значень MAE та RMSE для всіх моделей вказує на значні помилки, такі як прогноз 1 замість 30. Продуктивність базової архітектури

Transformer обмежена довжиною вхідної послідовності, що впливає на повноту представлення деяких зразків.

Таблиця 3.4.

Результат навчання обраних моделей на наборі даних RI.

	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) Conv-model, LO-data (b) LSTM-model, LO-data (c) Tranformer-model, LO-data

(d) Conv-model, RI-data (e) LSTM-model, RI-data (f) Tranformer-model, RI-data

Рис. 3.10. Значення MAE залежно від складності терму для: (a) згорткової моделі та LO-даних, (b) LSTM-моделі та LO-даних, (c) Трансформер моделі та LO-даних, (d) згорткової моделі та RI-даних , (e) LSTM-модель і RI-даних, (f) Трансформер модель і RI-даних.

Результати в табл. 3.4 показують, що моделі, натреновані на наборі RI, мають вищу точність порівняно з набором LO. Це пояснюється тим, що терми в наборі LO містять більше токенів (Рис. 3.7). Найнижчий рівень помилок продемонструвала модель LSTM, навчена на наборі RI, яка також показала вищу точність і нижчі значення MAE та RMSE.

Більш детальне порівняння ефективності моделей представлено на Рис. 3.10, де наведено графіки залежності похибки MAE від фактичної кількості кроків редукції терму (складність терму). Дані розбито на 31 групу за кількістю кроків, і для кожної групи обчислено середнє значення помилки. Графіки показують, що менша похибка зазвичай відповідає термам з меншою кількістю кроків, а найбільша похибка — у моделей, навчених на наборі стратегії LO. Для всіх графіків, крім Рис. 3.10.e, показники похибки для тренувальних і тестових наборів є близькими, що свідчить про ефективність обраної стратегії боротьби з перенавчанням.

3.3.3. Аналіз результатів оцінки кількості кроків редукції лямбда-термів за заданою стратегією

Використання моделей для прогнозування кількості кроків редукції термів є перспективним підходом для автоматизації вибору оптимальної стратегії редукції. Результати свідчать, що цей підхід може ефективно визначати стратегію редукції для кожного лямбда-терму, особливо в поєднанні з попередніми дослідженнями прогнозування часу редукції [54]. Це може сприяти оптимізації функціональних мов програмування. Виявлено, що спрощене представлення термів достатнє для прогнозування кількості кроків редукції. Однак, складність терму збільшує похибку прогнозу. Важливо зауважити, що робота проводилася на спрощеному представленні термів без урахування імен змінних, що слід врахувати в подальших дослідженнях.

3.4. Використання вбудовувань для прогнозування кількості кроків редукції за обраною стратегією

Наступним етапом дослідження є робота з термами у класичному їх представленні та використання вбудовувань. Основна ідея є ідентичною –

прогнозування кількості кроків редукції терму. Дане дослідження порівнює попередній підхід з використанням моделі CodeBERT та новим підходом з використанням вбудовування OpenAI Embeddings.

Вбудовування – це числове представлення даних, яке зазвичай використовується в контексті обробки природної мови для представлення слів, фраз або навіть цілих документів [87]. Вбудовування представляють собою попередньо натреновані вектори характеристик, що можуть бути використані для вирішення специфічних завдань. Ці щільні вектори фіксують семантичні властивості так, що подібні значення є близькими у просторі вбудовування. Вбудовування отримують за допомогою таких алгоритмів, як Word2Vec [87], GloVe [88], або через рівні в моделях глибокого навчання, таких як BERT [72] або GPT [89]. Вони дозволяють моделям обробляти текст, перетворюючи його у форми, які алгоритми машинного навчання можуть розуміти та виконувати над ними операції, такі як класифікація, кластеризація або пошук подібності. Ця техніка має вирішальне значення для роботи з величезними обсягами текстових даних і вилучення значущих шаблонів із неструктурованих даних.

Одним із що вирішують вбудовування є перетворення текстового представлення у вектори [72, 87–89].

Подальший аналіз даних вбудовувань може дозволити краще зрозуміти внутрішні зв’язки даних, в нашому випадку лямбда-термів, та виділити окремі характеристики, що впливають на кількість кроків редукції, а отже і на продуктивність виконання функціональних програм.

3.4.1. Мета і підхід

Метою даного етапу дослідження є покращення ефективності вилучення ознак лямбда-термів, що в подальшому дозволить покращити загальне розуміння процесу редукції. Як і в попередніх етапах дане розуміння в подальшому дозволить оптимізувати функціональні мови програмування. Дану задачу можна виконати при застосування передових методів машинного навчання, таких як вбудовування, що дозволяють ефективно перетворювати текстове представлення термів на

вектори ознак. Наступним кроком може бути аналіз цих характеристик за допомогою ШНМ для виявлення залежностей між ознаками та загальною кількістю кроків редукції.

Тож задачами даного етапу дослідження можна назвати:

1. Вибір моделей штучного інтелекту для перетворення лямбда-термів у вбудовування.

2. Налаштування гіперпараметрів моделей ШНМ для роботи з вбудовуваннями та вирішення задачу прогнозування кількості кроків редукції для обраних стратегій.

3. Порівняння отриманих результатів з попередньою роботою.

Дослідження працює зі штучно створеним набором лямбда-термів, використовуючи повне представлення з іменами змінних, що покращує аналіз внутрішніх зв'язків і прогнозування кількості кроків редукції. Однак збереження інформації про змінні потребує більше обчислювальних ресурсів. Використовувалися спеціалізовані та загальні LLM для перетворення текстового представлення лямбда-термів у вектори вбудовування, які далі застосовуються для прогнозування кількості кроків редукції, що є важливим для вибору продуктивної стратегії редукції.

Центральна гіпотеза дослідження: вбудовування текстового представлення лямбда-термів достатньо для прогнозування кількості кроків редукції та вибору оптимальної стратегії. Було використано моделі OpenAI, доступні через API, і проведено порівняння з моделлю CodeBERT. Якщо CodeBERT показує нижчі результати, це свідчить про недостатність невеликих моделей для відображення внутрішніх залежностей терму. Обидві моделі приймають терми у текстовому вигляді, але CodeBERT потребує додаткової обробки виходів, представлених у вигляді матриці, на відміну від OpenAI.

Порівняння моделей штучного інтелекту для генерації вбудовувань:

1. **CodeBERT**, розроблена корпорацією Майкрософт, є варіантом архітектури BERT, призначеної для мов програмування. Модель навчалася на

поєднанні природної мови та коду з кількох мов програмування, що дозволяє їй розуміти та генерувати код. CodeBERT відмінно справляється з такими завданнями, як пошук коду, створення документації коду та завершення коду [72].

2. **OpenAI text-embedding-ada-002**: ця модель від OpenAI є частиною серії моделей, оптимізованих для ефективного вбудовування тексту. Модель «ada» менша та швидша, що робить її придатною для додатків, де час відгуку та ефективність ресурсів є вирішальними. Вона розроблена для широкого спектру завдань із вбудовування тексту загального призначення [90].

3. **OpenAI text-embedding-3-small**: це більш компактна версія моделей вбудовування третього покоління OpenAI. Вона встановлює баланс між обчислювальною ефективністю та продуктивністю в задачах вбудовування. Це корисно для додатків, яким необхідно ефективно обробляти текстові дані, але при цьому потрібен значний рівень розуміння та контекстуалізації [90].

4. **OpenAI text-embedding-3-large**: як більший аналог, ця модель пропонує покращену продуктивність завдяки своєму розміру та глибині нейронної мережі. Вона розроблена для застосування у задачах з високими вимогами, де якість вбудовування значно впливає на результат, наприклад, у складних завданнях розуміння природної мови [90].

Таким чином, у той час як CodeBERT спеціалізується на розумінні та створенні текстів, пов’язаних з програмуванням, моделі вбудовування OpenAI орієнтовані на загальні завдання вбудовування тексту з опціями, адаптованими до різних операційних потреб – від легких (ada-002) до більш потужних (text-embedding-3).

3.4.2. ШНМ для прогнозування кроків скорочення

Дослідження передбачає, що вбудовування моделей OpenAI достатні для вибору ефективної стратегії редукції терму. Замість прямого прогнозування стратегії спочатку прогнозується кількість кроків для різних стратегій, а потім обирається найефективніша. Експерименти показали, що цей підхід є найбільш ефективним і перспективним.

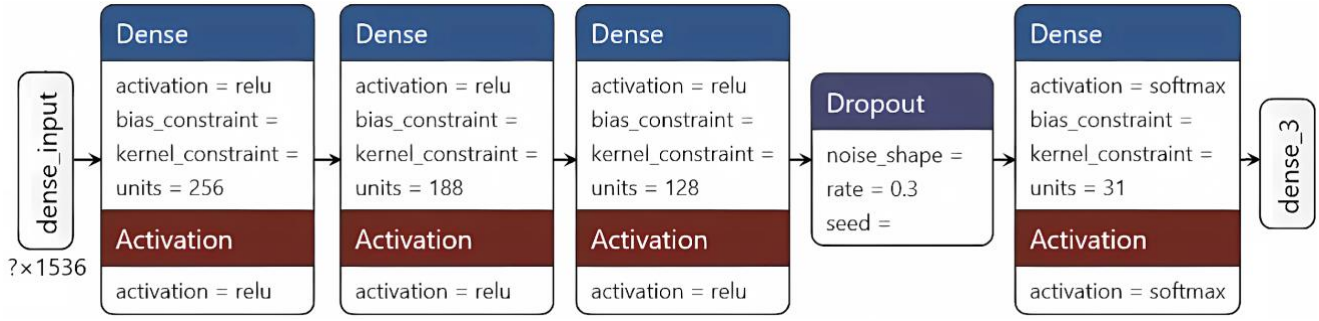


Рис. 3.11. Архітектура повністю зв'язаної моделі ШНМ, яка використовується для оцінки кроків нормалізації кількості з обраною стратегією скорочення.

Для вирішення цієї задачі використано модель ШНМ, що приймає на вхід вектори вбудовувань OpenAI і прогнозує кількість кроків редукції терму за обраною стратегією. Архітектуру ШНМ показано на Рис. 3.11. Прогнозування кількості кроків редукції вирішувалося як задача класифікації на 31 клас, де кожен клас відповідає кількості кроків від 0 до 30. Дані були розділені у співвідношенні 80/20 для тренування та тестування.

На Рис. 3.12 показана зміна точності й функції втрат під час навчання моделі ШНМ на вбудовуваннях OpenAI text-embedding-3-small. Табл. 3.5 містить результати тренування ШНМ для прогнозування кількості кроків редукції стратегії LO та порівняння з попередніми результатами на спрощених термах. Для тренувальних наборів використання вбудовувань може покращити результати до трьох разів, хоча для тестових наборів результати близькі, але все ж кращі, ніж з Microsoft CodeBERT. Зниження прогнозу кількості кроків для термів з великою кількістю кроків редукції підтверджується показниками RMSE.

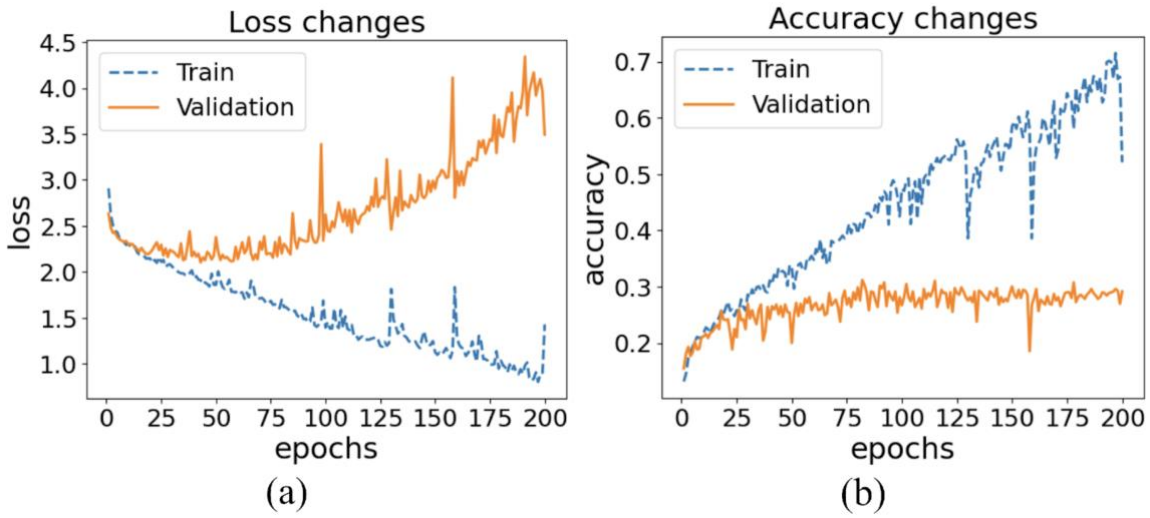


Рис. 3.12. Навчання повнозв'язної моделі ШНМ на вбудовуваннях OpenAI text-embedding-3-small для задачі прогнозування кількості кроків редукції для стратегії LO: (a) функція втрат; (b) прогресування точності

Таблиця 3.5

Результат навчання моделі ШНМ для прогнозування кількості кроків скорочення LO за допомогою вбудовування або спрощеного представлення термів

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 conv model [11]	2.91	2.74	5.28	5.16

Табл. 3.6 містить результати тренування моделей ШНМ для прогнозування кількості кроків редукції стратегії RO і моделі LSTM, як найефективнішої, за результатами попереднього розділу [73]. Виявлено, що використання вбудовувань не забезпечує значних покращень у прогнозуванні кроків редукції для стратегії RI порівняно з попередніми результатами. Хоча показник MAE для вбудовувань text-

embedding-3 кращий на навчальному наборі, на тестовому він гірший. RMSE для моделей з OpenAI показує незначне загальне покращення продуктивності.

Таблиця 3.6

Результат навчання моделі ШНМ для прогнозування кількості кроків редукції RI за допомогою вбудовування та спрощеного представлення термів

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

На Рис. 3.13 продемонстровано детальний аналіз показника похибки MAE. На 10 графіках для типів моделей та стратегій редукції представлені залежність показника похибки MAE від фактичної кількості кроків редукції терму (Рис. 3.13). На перших восьми графіках (Рис. 3.13.a–3.13.h) представлені результати для моделей вбудовування OpenAI, на двох останніх графіках (Рис. 3.13.i та Рис. 3.13.j) представлені результати для найкращою моделі попереднього етапу дослідження [73] з використанням спрощених термів для стратегії LO та RI.

З результатів тестування можна зробити висновок про те, що в більшості випадків використання вбудовувань та LLM з повним представлення лямбда-термів дають кращі результати, ніж моделі навчені на спрощеному представлені.

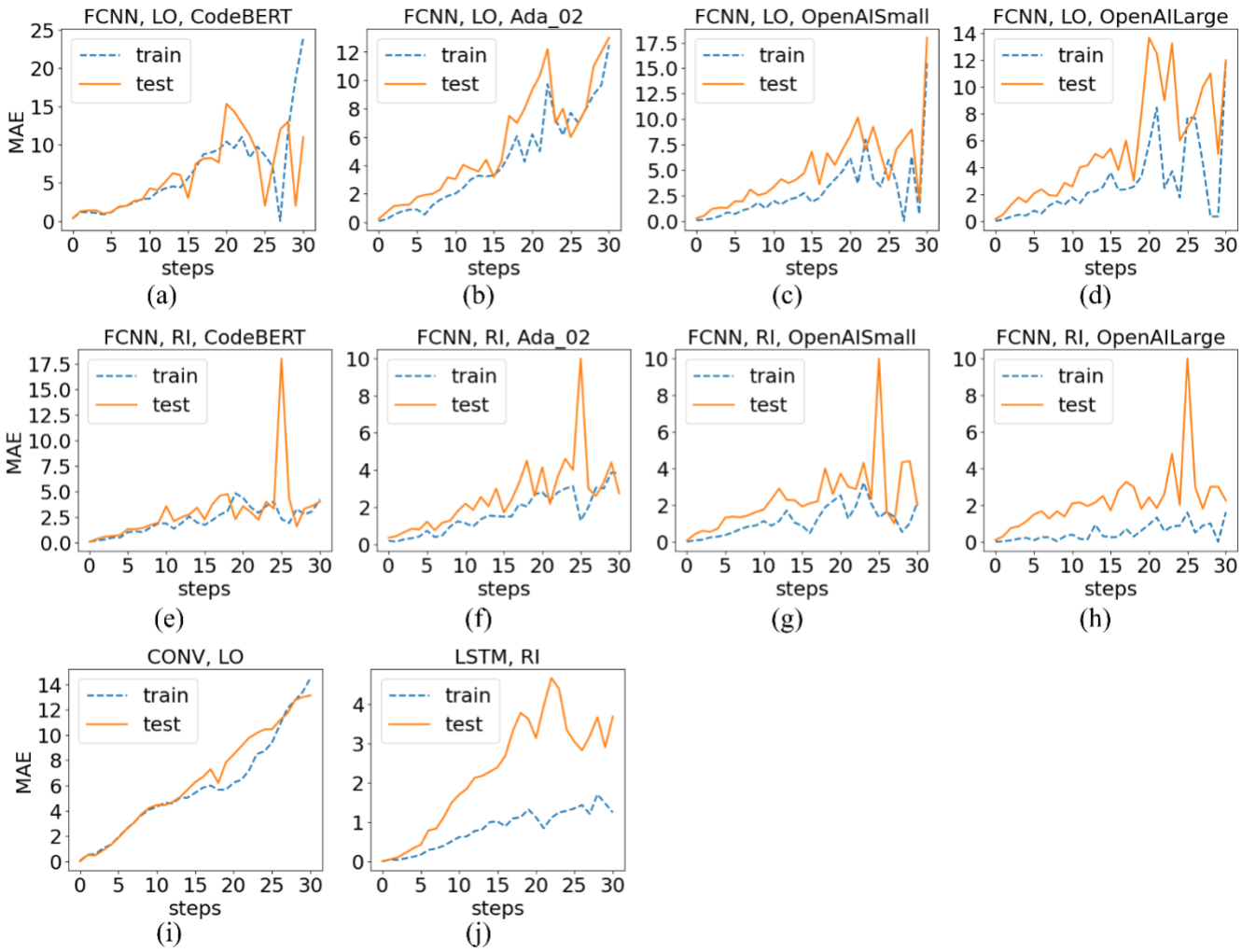


Рис. 3.13. Значення MAE залежно від кількості кроків редукції терму для: (a) CodeBERT на LO-термах; (b) text-embedding-ada-002 на LO-термах; (c) text-embedding-3-small на LO-термах; (d) text-embedding-3-large на LO-термах; (e) CodeBERT на RI-термах; (f) text-embedding-ada-002 на RI-термах; (g) text-embedding-3-small на RI-термах; (h) text-embedding-3-large на RI-темах; (i) найкращі спрощені LO-терми; (j) найкращі спрощені RI-терми.

Варто також зазначити, що представлені моделі не були спеціально треновані для аналізу лямбда-термів. Проте загальний рівень показників похибки MAE або RMSE, порівняно спеціально натренованими моделями розділу 3.1–3.3 [73]. Даний факт можна пояснити урахуванням інформації змінних, а також більшою кількістю ваг попередньо нетренованих моделей для представлень вбудовувань.

3.4.3. Аналіз результатів експериментів

На даному етапі дослідження було продемонстровано передові методи машинного навчання, що можуть бути використані для виявлення прихованих характеристик лямбда-термів. Знайдені характеристики можуть бути використані для оцінки та прогнозування більш продуктивної стратегії для конкретного лямбда-терму.

Потенційно дані характеристики можуть бути виділені іншими більш ефективними способами, що в подальшому можуть бути використані напряму у інтерпретаторах та компіляторах функціональних мов програмувань.

На відміну від попереднього етапу дослідження [73], в даному випадку були використані попередньо натреновані моделі ШІ, що не навчалися на представлених даних, та використання повного текстового представлення лямбда-термів.

З отриманих результатів можна побачити нижчі рівні показників похибки MAE та RMSE для задачі прогнозування кількості кроків редукції термів для стратегії RI (див. табл.3.6), ніж для стратегії LO (див. табл. 3.5), це може вказувати на більшу залежність між кількістю кроків для стратегії RI та характеристиками, що були виділені моделями вбудовувань, або те що, характеристики, що впливають на продуктивність стратегії RI є більш легко помітними.

Також цікавим є факт того, що моделі вбудовувань загального призначення OpenAI перевершують модель Microsoft CodeBERT, що була попередньо спеціально натренована на мовах програмування.

Даний факт можна пояснити тим, що моделі OpenAI мають більшу кількість вагових коефіцієнтів та були натреновані на більшому об’ємі даних, через це можуть більш ефективно виділяти внутрішні залежності.

Одним із обмежень даного дослідження при роботі з моделлю Microsoft CodeBERT є той факт, що дана модель була попередньо на тренована на таких мовах програмування як (Go, Java, Python та іншими), що не пов’язані напряму з лямбда-численням, цей нюанс може призвести до не зовсім правильної інтерпретації моделлю лямбда-термів.

Проте дана ідея також може бути застосована і до моделей OpenAI Embeddings, що були натреновані для вирішення загальних задач, а не прямої роботи з лямбда-численням.

Іншим обмеженням даного дослідження можна назвати використання штучно генерованих термів, що можуть на передавати напряму всіх залежностей та особливостей функціонального коду, що написаний людиною, а також відносно невеликий розмір датасету.

В даній роботі було проведено порівняльний аналіз чотирьох моделей для генерації вбудовувань текстових представлень лямбда-термів. Як і для попередніх експериментів було використано штучно згенерований набір даних розміром близько 4 тисяч лямбда термів. Робота із вбудовуваннями була проведена для аналізу можливості використання моделей LLM в якості екстракторів характеристик лямбда-термів з текстового представлення, що впливають на кількість кроків редукції для обраної стратегії, що в подальшому дозволить оптимізувати процес виконання функціональних мов програмування.

Впродовж роботи було побудовано вісім датасетів для тренування і тестування, для кожної з чотирьох моделей вбудовувань та двох стратегій редукції LO та RI. В якості цільової змінної була взята кількість кроків редукції терму для заданої стратегії. Архітектура моделей та їх тренування відбувалося для вирішення задачі класифікації на 31 клас – від 0 до 30 кроків редукції. В ході експериментів було виявлено, що даний підхід є найбільш багатообіцяючим.

Для аналізу моделей були зібрані показники похибки MAE та RMSE, а також досліджена залежність між MAE та фактичною кількістю кроків редукції терму. Отримані показники було порівняно з попередніми результатами для того самого набору даних, але при використання спрощеного представлення термів, та тренування моделей на даних з нуля. Отримані результати вказують на покращення метрик у вирішенні проблеми прогнозування кількості кроків редукції терму. Більшого прогресу було досягнуто саме у відношенні стратегії редукції LO. Для прогнозування кількості кроків стратегії RI значних покращень не було досягнуто

у порівнянні з найкращими результатами попереднього етапу дослідження [73]. Аналізуючи все вищесказане можна зробити висновок що LLM та моделі вбудовувань загального призначення можуть бути використані для вилучення характеристик лямбда-термів та подальшого вибору більш ефективної стратегії на основі цих характеристик.

Також можна сказати що загальний підхід та ідея використання LLM у процедурі вилучення ознак може бути використана для покращення роботи інтерпретаторів та компіляторів, що працюють з функціональними мовами програмування.

В якості наступних кроків дослідження має сенс проаналізувати LLM та моделі вбудовування великих розмірів та кількістю ваг, що були натреновані напряму на термах лямбда-числення або на дуже схожих мовах програмування чи областях. Також один із можливих покращень може бути використання більшого датасету для ширшого аналізу, та використання реальних функціональних програм, побудованих людиною замість штучно генерованого набору термів. Одним із наступних кроків також має сенс провести глибокий аналіз вилучених характеристик термів, наприклад методами неінформованого навчання, для глибшого розуміння внутрішніх структур та залежностей лямбда-термів.

Висновок до розділу 3

На даному етапі дослідження було проаналізовано декілька підходів для вирішення проблеми прогнозування кількості кроків редукції терму за обраною стратегією. В першому, другому та третьому розділі були проаналізовані сучасні базові архітектури ШНМ для роботи з текстовими послідовностями – CNN, LSTM та BERT. Дані моделі були навчені з нуля на згенерованому наборі даних лямбда-термів для вирішення задачі прогнозування кількості кроків редукції для заданої стратегії. На вхід подавалися лямбда-терми у спрощеному представленні. Потім терми були представлені у стандартному текстовому форматі та подані до різних моделей ШНМ.

У четвертому підрозділі були протестовані такі моделі як Microsoft CodeBERT та моделі вбудовувань від OpenAI. В обох випадках моделі приймали на вхід текстове представлення термів. Але в розділі 3.4 було використане звичайне представлення термів.

В усіх архітектурах моделей було використано повнозв'язну ШНМ у якості фінального шару. В обох випадках задача прогнозування кількості кроків редукції лямбда терму вирішувалася, як задача класифікації на 31 клас, де кожен клас представляє собою кількість кроків редукції від 0 до 30. Особливістю підходу з використанням моделей CNN, LSTM та BERT є те, що в даному випадку робота проводилася зі спрощеним представленням термів, де всі імена змінних були замінені однією змінною «*x*». Це було зроблено на перших етапах дослідження для зменшення обчислювального навантаження, через зменшення розміру вхідного вектору.

Далі в підрозділі 3.4 було порівняно результати з використанням невеликої моделі з відкритим кодом Microsoft CodeBERT та трьох моделей вбудовувань OpenAI. За результатами можна сказати що, на тренувальних даних використання вбудовувань дозволяє покращити результат до трьох разів, з іншої сторони ж на тестовому наборі даних таких значних покращень не відбувається. Можна також зробити висновок, що вбудовування дозволяють відобразити більшу частину інформації або те, що архітектура моделей ШНМ побудованих з використанням вбудовувань більш схильна до перенавчання.

Експерименти також показують що використання спрощеного представлення термів є достатньо для прогнозування кількості кроків редукції терму, хоча повне стандартне представлення дозволяє досягти кращих результатів та передати інформацію більш точно.

В якості наступних кроків має сенс провести аналіз вбудовувань для виявлення внутрішніх характеристик та взаємозв'язків в лямбда-термах, без прямого фокусу на прогнозуванні кількості кроків або більш продуктивної стратегії. Ідея даного підходу має на меті краще зрозуміти особливості датасету та

включає в собі використання інформативних методів машинного навчання та проведення кластерного аналізу для виявлення існуючих підмножин термів зі схожими характеристиками, що дозволить в подальшому працювати з кожною із підмножин окремо.

РОЗДІЛ 4.
НЕІНФОРМОВАНЕ НАВЧАННЯ

4.1. Розробка підходу до вилучення даних лямбда-термів

4.1.1. Опис проблеми вилучення даних лямбда-термів

У даному розділі описано альтернативні підходи для аналізу внутрішніх залежностей лямбда-термів та можливих варіантів застосування архітектур LLM для вирішення проблем лямбда-числення.

У розділі зібрано результати експериментів, проведених під час усього дослідження, що пов’язані з неінформованим навчанням, та деякими новими підходами для вирішення проблем лямбда-числення, зокрема виконання кроку редукції лямбда терму за обраною стратегією з використанням LLM. Також у підрозділі 4.4. представлено можливий метод імплементації попередньо отриманих знать для розробки інструменту оптимізації продуктивності функціональних мов програмування.

Як і в підрозділі 3.4 на даному етапі дослідження в якості основної моделі було використано модель Microsoft CodeBERT, що була попередньо натренована на мовах програмування для вирішення задач, пов’язаних з програмним кодом. В ході використання моделі Microsoft CodeBERT у якості екстрактора ознак лямбда термів, були отримані матриці вбудовувань, що є проміжними входами моделі, що в подальшому були перетворені на векторні представлення розміром 768.

Для проведення аналізу на основі отриманих векторі вбудовувань в даній роботі були використані такі алгоритми як Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [91] та Hierarchical Agglomerative Clustering [92]. В ході роботи було проведено аналіз найбільш інформативних змінних, що впливали на кластеризацію.

Особливістю описаного підходу можна назвати використання інформативних методів машинного навчання в якості інструменту для більш глибокого розуміння лямбда-числення в цілому. В результаті проведеного кластерного аналізу було

виявлено велику кількість однакових інформативних змінних, даний факт можна пояснити схожою формою отриманих кластерів.

Було проведено аналіз утворених кластерів та порівняння їх з реальними підмножинами термів розділеними за пріоритетами стратегій, отримані групи кластерів не мають чіткого перетину. Даний факт може бути пояснений недостатньо чітким зв’язком між отриманими з вбудовувань характеристиками термів та пріоритетними стратегіями редукції, а також тим фактом, що модель Microsoft CodeBERT не була напряму натренована на термах лямбда-числення.

З представленого аналізу можна зробити висновок про те, що модель Microsoft CodeBERT не забезпечує необхідної точності визначення пріоритетності стратегій редукції, що можна пояснити тим фактом, що початково модель не була навчена для аналізу лямбда-термів.

Розроблений підхід може бути використаний для вилучення важливих характеристик лямбда термів, що можуть бути у подальшому використані для визначення залежностей та внутрішніх зв’язків, що допоможуть удосконалити процес редукції лямбда-термів та в подальшому функціональних програм. Правила та інструменти, розроблені в ході дослідження можуть бути використані для підвищення ефективності компіляторів та інтерпретаторів функціональних мов програмування.

Ідея полягає у використанні моделей штучного інтелекту для вилучення залежностей між характеристиками терму та пріоритетними стратегіями редукції, для підвищення продуктивності процесу редукції та самих компіляторів та інтерпретаторів.

Як і у попередньому розділі на даному етапі дослідження використовується той самий попередньо згенерований датасет лямбда-термів. В даному випадку в підрозділі 4.1. проводиться робота зі спрощеним представленням лямбда термів, а в підрозділі 4.2. представлений аналіз звичайного представлення. Даний набір даних, як і було описано раніше є доволі надійним та достатнім для проведення експериментів покращення якості редукції лямбда термів [43, 54].

Даний етап дослідження ставить перед собою метою розробку підходу вилучення характеристик лямбда-термів, що мають вплив на процес редукції та ефективність деяких стратегій по відношенню до лямбда-терму. Даний етап дослідження фокусується на використанні методів неінформованого машинного навчання для пошуку та аналізу внутрішніх структур на залежностей у просторі лямбда-термів. Даний підхід дасть можливість забезпечити ефективне та якісне представлення лямбда-термів у вигляді вектору ознак, що можуть бути використані для вирішення широкого спектру задач лямбда-числення.

Також аналіз отриманих векторних представлення на роздільність, наявність кластерів та окремих підмножин зі схожими характеристиками дозволить працювати з кожною із отриманих груп окремо або отримати вагомі знання про структуру зв'язків лямбда-термів, що дозволить в подальшому більш ефективно вирішувати питання розробляти підходи оптимізації стратегій редукції лямбда-термів, а отже і функціональних мов програмування.

Для досягнення мети поточного етапу дослідження були поставлені такі задачі:

– аналіз можливості застосування обраного методу машинного навчання для виділення вагомих характеристик лямбда-термів;

– проведення кластерного аналізу для виявлення внутрішніх взаємозв’язків, підмножин та можливості розділення даних;

– перевірка залежностей між виділеними характеристиками та існуючими кластерами або групами, наприклад з точки зору пріоритетності однієї із стратегій;

– перевірка залежностей між лямбда-термами в середині отриманих кластерів та стратегіями редукції, що є більш або менш ефективними для певних кластерів.

4.1.2 Матеріали та методи кластерного аналізу

Об’єктом представленого етапу роботи є процес вилучення характеристик та внутрішніх зв’язків лямбда термів, що можуть прямо або опосередковано впливати на процес редукції та більшу ефективність певних стратегії. Як приклад можливих характеристик можна виділити комбінації специфічних типів редексів або

наявність тих або інших підтермів, що мають суттєвий вплив процес редукції, зокрема його траєкторію, кількість кроків тощо.

Головна гіпотеза полягає в тому, що застосування методів машинного навчання, їх правильний підбір та налаштування дозволить ефективно перетворювати текстове представлення лямбда термів у змістовне числове. А отримане числове представлення може бути використано у подальшому для вирішення вибору стратегії редукції.

Даний етап дослідження спрямований на отримання більш глибшого загального розуміння процесу редукції та внутрішніх структур лямбда термів. Оскільки, як і обговорювалося раніше, лямбда-числення має спільну базу з функціональними мовами програмування, це в подальшому дозволить отримати краще розуміння внутрішніх структур і функціональних мов.

Як вже було описано в попередніх розділах, сьогодні найдосконаліші моделі навчання побудовані на базі архітектури Transformer, тож саме вони використовувалися як інструмент дослідження. Основною ідеєю, як і в третьому розділі, в цьому розділі розглядається можливість використання виходів середніх шарів моделей LLM у якості вектору ознак лямбда-термів, що також називаються вбудовуваннями.

На даному етапі дослідження було проведено порівняльний аналіз існуючих моделей LLM, розроблених для роботи з мовами програмування. На даний момент сервіс HuggingFace є одним із найкращих джерел для пошуку та роботи з найбільш популярними моделями, тож в роботі був використаний саме він. Порівняльний аналіз моделей представлено в табл. 4.1. Одними з найдосконаліших LLM для роботи з програмним кодом на сьогодні є моделі Code Llama [93], які представлені у трьох можливих розмірах 7B, 13B і 34B параметрів.

Представлені моделі були натреновані для вирішення проблем пов’язаних із кодом на найпопулярніших мовах програмування, таких як Python, Java, JavaScript та іншими. Проте деякі представлені моделі вимагають великих обчислювальних потужностей, що не дає змогу використовувати їх для експериментів. В табл. 4.1

представлено порівняльний аналіз LLM, що були розглянуті. В якості основної моделі було обрано Microsoft CodeBERT, оскільки вона має відносно високу якість розуміння коду та достатньо невеликий об’єм ваг, щоб бути застосованою для експериментів.

Таблиця 4.1.

Порівняння LLM для мов програмування.

№	Модель	Опис	Задачі	Розмір
1	CodeTrans model	Модель заснована на T5-small та має власну модель словника SentencePiece. Використовувалося попереднє навчання для 7 неконтрольованих наборів даних у сфері розробки програмного забезпечення. Потім модель було налаштовано на завдання синтезу програмного коду DSL, натхненного Lisp [94]	Програмний синтез; генерація документації; зведення коду; генерація коментарів	242 MB
2	Replit Code	Причинно-наслідкова мовна модель навчена на підмножині набору даних Stack Dedup v1.2. Навчальна суміш включає 20 різних мов програмування, перерахованих тут у порядку зменшення кількості токенів: Markdown, Java, JavaScript, Python, TypeScript, PHP та інші [95]	Доповнення коду	10.4 GB
3	Code Llama	Code Llama – це набір попередньо підготовлених і точно налаштованих генеративних текстових моделей із масштабом від 7 до 34 мільярдів параметрів [93]	Загальний синтез і розуміння коду	~13 GB – ~70 GB
4	CodeT5	CodeT5 – це уніфікована модель Transformer з попередньою підготовкою. Модель було попередньо навчено на CodeSearchNet (Go, Java, JavaScript, PHP, Python і Ruby). Крім того, автори зібрали два набори даних C/C# з BigQuery1, щоб переконатися, що всі подальші завдання	Узагальнення коду, генерація, переклад, уточнення та виявлення дефектів	892 MB

		перекривають мови програмування з даними попереднього навчання [96]		
5	CodeBERT	CodeBERT – це бімодальна попередньо навчена модель для мов програмування та природної мови, що використовує нейронну архітектуру на основі Transformer і гібридну цільову функцію, яка включає завдання попереднього навчання виявлення замінених токенів. Автори розглянули набори даних, що містять Go, Java, JavaScript, PHP, Python, Ruby та інші зразки коду для навчання [97]	Генерація кодової документації	499 MB

4.1.3 Засоби неконтрольованого розділення даних та оцінки такого розділення

Дослідження показало, що неінформоване навчання, зокрема кластерний аналіз, є ефективним для аналізу даних, зокрема розподілів усереднених вбудовувань. Кластеризація характеристик виявила можливість виділення окремих кластерів за пріоритетністю стратегій редукції, що відкриває перспективи для автоматичних підходів у майбутньому. Однак на цьому етапі не ставилась задача розрізнення термів за пріоритетністю стратегій; основна мета полягала в загальному аналізі внутрішніх залежностей та структур.

Методи використані для проведення кластерного аналізу наведені далі.

– ***k-means*** – популярний алгоритм кластеризації в неконтрольованому машинному навчанні, що поділяє дані на k роздільних кластерів, мінімізуючи дисперсію в кожному з них. Процес включає вибір k початкових центроїдів, ітераційне призначення точок до найближчих кластерів і перерахунок центроїдів до стабілізації їх позицій. Метод ефективний для великих наборів даних, але потребує попереднього визначення кількості кластерів і може бути чутливим до початкового вибору центроїдів [98].

– ***DBSCAN*** – алгоритм кластеризації, який ідентифікує кластери на основі щільності точок даних, обробляючи різні форми і розміри кластерів. Він визначає

точки як основні, граничні або шумові, використовуючи параметри minPts (мінімальна кількість точок для щільної області) та епсилон (радіус околиці). Основні точки формують кластер, граничні точки приєднуються до найближчого кластера, а шумові точки не входять до жодного з них. DBSCAN ефективний для даних з шумом і неоднаковими формами кластерів, і не вимагає попереднього визначення кількості кластерів [91].

– ***Gaussian Mixture Model (GMM)*** – імовірнісна модель, яка припускає, що дані генеруються з суміші кінцевої кількості розподілів Гауса з невідомими параметрами. GMM ідентифікує кластери в даних, де кожен розподіл Гауса представляє окремий кластер. Основні аспекти GMM [99] показані далі.

1. Компонент Гауса: представляє кластер з середнім значенням (центр), коваріацією (форма і орієнтація) і вагою (значимість кластеру).

2. Очікування-максимізація: оцінка параметрів GMM через ітеративне призначення точок даних до компонентів Гауса на основі ймовірностей, з наступним оновленням параметрів для кращого відповідності даним.

GMM моделює складні форми кластерів завдяки коваріаційним матрицям і м’якій кластеризації, надаючи імовірнісну міру приналежності для кожної точки, що робить його гнучкішим, ніж k-means [99].

– ***Hierarchical Agglomerative Clustering (HAC)*** [92] – це метод кластеризації, який створює ієрархію кластерів шляхом поступового злиття пар кластерів. HAC починається з окремих кластерів, які поступово об'єднуються до досягнення єдиного кластера або бажаної кількості кластерів. Ключові особливості HAC:

1. Об’єднання кластерів базується на критерії зв’язку, такому як мінімальна, максимальна, середня відстань або відстань між центроїдами. Ці критерії впливають на форму і розмір кластерів.

2. Дендрограма візуалізує процес злиття кластерів і допомагає визначити кількість кластерів шляхом розрізання на відповідному рівні.

HAC корисний для даних з ієрархічною структурою і забезпечує детальну ієрархію кластерів, але є обчислювально витратнішим, ніж k-means, особливо для великих наборів даних [92].

Після проведеного візуального аналізу побудованих кластерів (Рис. 4.1) в якості основних алгоритмів кластеризації було обрано DBSCAN і HAC для проведення подальшого, більш глибокого аналізу.

Оскільки описані вище методи вимагають тонкого налаштування гіперпараметрів, для валідації обраних гіперпараметрів було підібрано набір метрик для оцінки якості кластеризації:

1. ***Silhouette Score*** [100] – це показник якості кластерів, який вимірює, наскільки об'єкт схожий на власний кластер (згуртованість) порівняно з іншими кластерами (відокремлення). Оцінка варіюється від -1 до 1:

• +1: Об'єкт добре підходить до свого кластера і погано до сусідніх.

• 0: Об'єкт на межі між двома кластерами.

• -1: Об'єкт може бути в неправильному кластері.

Оцінка розраховується для кожного зразка та може бути усереднена для загальної ефективності кластеризації. Це допомагає визначити оптимальну кількість кластерів і оцінює компроміс між згуртованістю і відокремленням кластерів.

2. ***Davies-Bouldin Index (DBI)*** [101] – це метрика для оцінки кластеризації, яка вимірює розділення та компактність кластерів. DBI оцінює середню «подібність» між кожним кластером і найбільш схожим, комбінуючи дисперсію всередині кластерів та їх розділення. Ключові аспекти DBI:

• Розрахунок: Для кожного кластера DBI обчислює подібність до інших кластерів, використовуючи співвідношення дисперсії всередині кластера до міжкластерного розділення, і знаходить максимальні значення подібності.

• Інтерпретація: Низький DBI вказує на кращу кластеризацію з меншою внутрішньокластерною дисперсією і більшим міжкластерним розділенням. Вищий DBI сигналізує про погану кластеризацію.

DBI не потребує інформації про реальні класи і оцінює лише атрибути кластерів, проте може віддавати перевагу сферичним скупченням.

3. *Calinski-Harabasz Index (CHI)* [102], або критерій співвідношення дисперсії, оцінює якість кластеризації, вимірюючи відношення міжкластерної дисперсії до внутрішньокластерної. Ключові аспекти CHI:

• Розрахунок: CHI визначається як відношення міжкластерної дисперсії (сума квадратів різниць між центроїдами кластерів і загальним центроїдом, масштабована на кількість кластерів мінус один) до внутрішньокластерної дисперсії (сума квадратів відстаней між точками даних і центроїдами, масштабована на загальну кількість точок даних мінус кількість кластерів).

• Інтерпретація: Вищий CHI вказує на кращу кластеризацію, з добре розділеними і компактними кластерами. CHI корисний для вибору оптимальної кількості кластерів, порівнюючи значення для різних кількостей кластерів.

CHI широко використовується через свою простоту та ефективність у визначенні якості кластеризації, але може віддавати перевагу сферичним кластерам і бути чутливим до кількості кластерів і розміру набору даних.

4. *Within Cluster Sum of Squares (WCSS)* [103] є показником для оцінки компактності кластерів у кластерному аналізі, вимірюючи загальну дисперсію всередині кожного кластеру. Ключові аспекти WCSS:

• Розрахунок: WCSS обчислюється як сума квадратів відстаней між кожною точкою та її центроїдом в кожному кластері, потім сума цих значень для всіх кластерів.

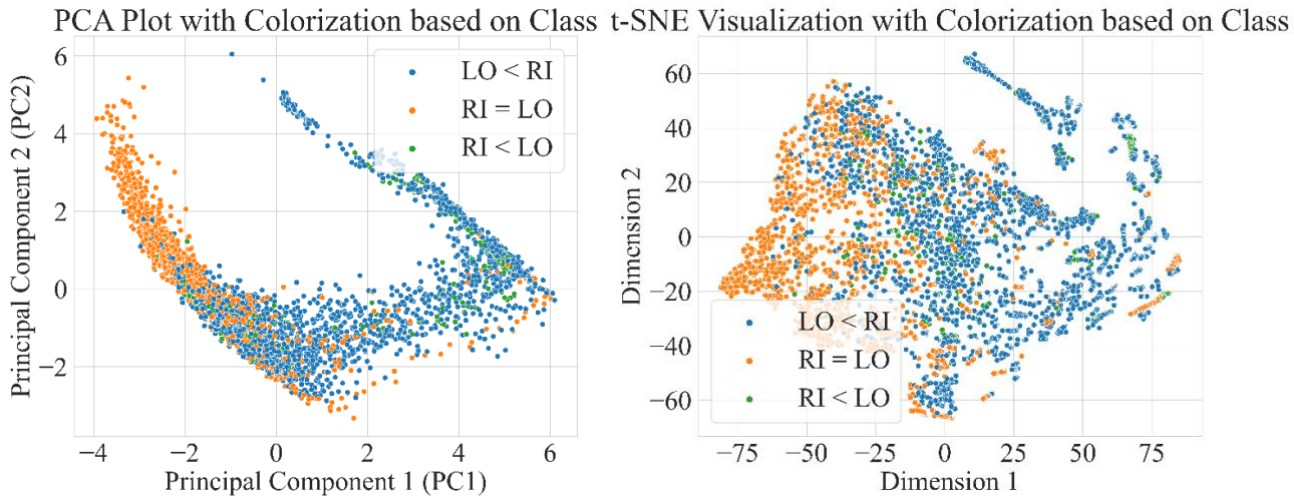
• Інтерпретація: Низьке значення WCSS вказує на щільні та добре розділені кластери, що є ознакою якісної кластеризації. Це важливо для методів, таких як k-means, де мета – мінімізувати WCSS для оптимальної кластеризації.

WCSS часто використовується для визначення оптимальної кількості кластерів через метод ліктя, коли кількість кластерів збільшується, поки зменшення WCSS не стає незначним, що свідчить про досягнення оптимальної кількості кластерів.

4.1.4 Результати дослідження вилучення ознак із лямбда-термів

Нижче представлені результати візуалізації простору лямбда-термів з використанням різних методів зменшення розмірності.

Після проведення кластерного аналізу було отримано набір можливих кластерів, та проведено аналіз чутливості змінних. Аналіз представляє собою тренування нейронної мережі на задачі класифікації де на вхід подаються векторні представлення термів а класами є зпрогнозовані кластери. Це дає змогу зрозуміти, які змінні найбільше впливають на утворення кластерів. Результати приведені нижче в табл. 4.2.



(a) (b)

Рис. 4.1. Порівняння значення усереднених вбудовувань із позначенням пріоритету стратегії за алгоритмами стиснення (a) PCA та (b) t-SNE.

Також було підраховано коефіцієнт перекриття, що представляє собою коефіцієнт співпадіння зпрогнозованих кластерів та реального розбиття по класах відповідно до пріоритетних стратегій. Результат показаний в табл. 4.3.

Таблиця 4.2.

Топ-5 найбільш інформативних змінних результатів кластерного аналізу векторів усереднених вбудовувань.

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

Таблиця 4.3.

Значення коефіцієнту перекриття результатів кластерного аналізу.

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

4.1.5 Аналіз результатів дослідження вилучення ознак із лямбда-термів

Особливістю описаного етапу роботи є використання передових методів машинного навчання, таких як LLM у вирішенні задачі пошуку та виділення значущими прихованих характеристик лямбда-термів. Дані ознаки потенційно можуть бути використані для виділення пріоритету однієї із стратегій редукції. Підхід для виявлення даних ознак у лямбда численні може бути в подальшому перенесений і для використання в інших функціональних мовах програмування. Також для майбутньої імплементації має сенс проаналізувати вартість

обчислювальних витрат для використання LLM в порівнянні з тим скільки економії в обчисленнях надає обрана оптимальна стратегія. Тож даний підхід може дозволити зрозуміти потенціал LLM у визначенні ознак, що впливають на пріоритет стратегій, та в подальшому використовуючи найпростіші техніки, як наприклад, пошук патернів можливо застосовувати їх на практиці.

Описана методологія дозволить в майбутньому підвищити продуктивність функціональних мов програмування їх інтерпретаторів та компіляторів. Проведені експерименти дозволили перевірити можливість застосування LLM у якості інструменту дослідження.

Описані методи машинного навчання мають ряд переваг, що обґрунтовуються більшими можливостями у розрізі роботи з великою кількістю лямбда-термів, та потенціалу для використання натренованих LLM для вирішення задачі вилучення ознак. Цей підхід є відмінним та більш перспективним, порівняно з [104], де було застосовано підхід експертної логіки для отримання ознак функціональної програми для оптимізації інтерпретатора.

В подальшому були перевірені залежності між вилученими характеристиками лямбда-термів, та кластерами, на які вони були розбиті, це дозволило отримати набір найбільш інформативних змінних, що представлені в табл. 4.2. Варто зазначити факт, що існує деяких набір змінних, що є ідентичним для більшості результатів кластеризації. Схожість отриманих результатів може бути причиною цього.

В ході роботи також було перевірено залежності між побудованими кластерами та реальною пріоритетністю стратегій відносно лямбда термів. В табл. 4.3 продемонстровано, що терми з пріоритетом стратегії RI дуже важко або навіть неможливо виділити із множини інших термів, однак терми з іншою пріоритетністю стратегій могли бути розділені з цілком прийнятною точністю. Отримані результати можуть бути пояснені тим фактом, що модель Microsoft CodeBERT немає достатньої бази знань по відношенню до лямбда-термів, і через це була втрачена інформативність усереднених вбудовувань. Однією із можливих

причин може бути природа використаного для тренування штучно згенерованого набору даних лямбда-термів, що насправді може не відображати всіх внутрішніх залежностей реального людського коду та програм, а також недостатньою точністю виділених пріоритетів стратегій.

Візуальна презентація отриманих результатів зменшення розмірності простору усереднених вбудовувань (Рис. 4.1 та Рис 4.2) та метрики перекриття (табл. 4.3) демонструють достатньо високий потенціал запропонованого підходу. Представлені методи машинного навчання можуть бути використані у майбутньому для екстракції певних ознак та оптимізації процесу редукції функціональних мов програмування.

Одним з обмежень дослідження можна виділити недостатню чіткість розрізнення стратегій, що полягає у підрахунку різниці у кількості кроків між двома стратегіями, оскільки для моделей не відомо чи пріоритетність стратегії в покроковому представленні являє собою різницю в 5 або в 20 кроків, це може викликати деякі складнощі для процесу тренування моделі.

Також одним із обмежень можна виділити використання в якості моделі Microsoft CodeBERT, що початково була натренована для роботи з мовами програмування загального призначення, як наприклад (Go, Java, Python та інших), даний нюанс може зменшити якість представлення лямбда-термів у вигляді вбудовувань, а також переведення вбудовувань в більш універсальне представлення для майбутньої роботи.

Одним із обмежень можна виділити гіпотезу про те, що використання LLM у якості інструменту вилучення ознак може бути достатньо, а завдяки вилученим ознакам можна буде в подальшому визначити пріоритет редукування лямбда-терму. Для врахування зазначених вище недоліків та обмежень в майбутньому має сенс провести дослідження з використанням LLM, що були спеціально натреновані для вирішення задач, пов’язаних із лямбда-численням.

Одним із можливих методів застосування у майбутньому може бути використання техніки підсумовування тексту, що дозволить зменшити об’єм втрати даних в процесі переходу до усереднених вбудовувань.

1) В ході описаного етапу дослідження завдяки використанню методології ідентичній Word2Vec, спрощене текстове представлення лямбда-термів було перетворено на матриці вбудовувань завдяки використанню моделі Microsoft CodeBERT, що подальшому були усереднені до векторів вбудовувань розмірністю 768. Було проведено аналіз побудованого простору вбудовувань, а також проведена робота з деякими альтернативними підходами у вигляді латентного простору. В ході використанням методів зменшення розмірності PCA та t-SNE та проведення візуального аналізу було помічено ознаки розділення лямбда-термів на кластери в просторі усереднених вбудовувань. Даний факт служить підтвердженням гіпотези щодо використання кластерного аналізу для виявлення внутрішніх структур та взаємозв’язків термів.

2) Також було проведено кластерний аналіз з використанням таких алгоритмів як DBSCAN та HAC та використанням різних метрик, таких як евклідова, косинусна та L1. Проведений кластерний аналіз підкреслив потенціал використання та ефективність моделі Microsoft CodeBERT у вилученні вагомих характеристик лямбда-термів. Незважаючи на продемонстровано ефективність, слід зазначити, що модель Microsoft CodeBERT навчалася на мовах програмування загального призначення, а не безпосередньо на лямбда-численні, що також вносило деякі складнощі під час дослідження. Також варто зазначити, що особливість моделі Microsoft CodeBERT також вносить деякі складнощі в процес перетворення отриманих матриць вбудовувань у значущі та зрозумілі усереднені вектори латентного простору, та при подальшому методів підходів зменшення розмірності, зокрема автокодувальників.

3) Було проведено аналіз чутливості та інформативності змінних для оцінки їх ролі та впливу в середині отриманих кластерів. Даний факт дозволив окреслити деякий набір найбільш інформативних змінних, що є ідентичними в незалежності

до обраного методу кластеризації, та які вносять найбільший вплив на побудову існуючих кластерів. Таку особливість можна пояснити існуванням певних багатовимірних структур в середині простору лямбда-термів, які різні методи кластеризації визначають приблизно однаково.

4) Аналіз та впровадження коефіцієнту перекриття, також позитивно вплинув на процес оцінки взаємозв'язків між утвореними кластерами та пріоритетами стратегій редукції відносно термів. Робота з оцінкою перекриття дозволила виявити відсутність вагомої кореляції між отриманими кластерами та фактичними пріоритетами стратегій редукції для кожного терму. Отримані результати вказують на необхідність аналізу альтернативних моделей машинного навчання, що навчені безпосередньо на областях суміжних з лямбда-численням або тонкого налаштування моделі Microsoft CodeBERT на тренувальних даних лямбда-термів.

4.2. Кластеризація лямбда-термів за допомогою вбудовувань

4.2.1. Використання вбудовувань у проблемі кластеризації

На даному етапі роботи, як і в підрозділі 4.1 основною ідеєю є застосування кластерного аналізу для дослідження внутрішніх залежностей лямбда-термів, та вилучення їх важливих характеристик що впливають на процес редукції або можуть надати можливості для його оптимізації.

Представлена частина дослідження зосереджена на використанні передових методів машинного навчання для трансформації текстового представлення лямбда-термів у значущі вектори ознак та проведенні кластерного аналізу, що в подальшому дозволить покращити розуміння та вибір більш ефективної стратегії для конкретного терму.

Було використано комплексний підхід, що є деякою комбінацією попередніх етапів, що включає в собі використання згенерованого датасету лямбда-термів, кластерного аналізу та сучасних моделей машинного навчання для вилучення вбудовувань.

На поточному етапі дослідження було використано ідентичні моделі, що були представлені в підрозділі 3.4, що являють собою моделі вбудовувань OpenAI Embeddings.

В якості методу кластеризації було використано DBSCAN. В якості методів візуалізації та зменшення розмірності було використано PCA та t-SNE. Для візуального аналізу та виявлення шаблонів та можливих явних окремих кластерів даних.

В результаті поточного етапу дослідження було виявлення явні відмінності між побудованими кластерами, та характеристиками термів всередині кластерів, це підтверджує гіпотезу про те, що такого роду аналіз може допомогти виділити шаблони та окремі підгрупи лямбда-термів.

Однак, треба зазначити, що через те, що моделі вбудовувань OpenAI Embeddings, є вбудовуваннями загального призначення, тобто були натреновані для роботи з людським текстом та програмним кодом, а не безпосередньо для роботи з лямбда численням, це вносить деякі складнощі в процес побудови представлення лямбда-термів.

Даний етап дослідження підкреслює обмеження поточних моделей машинного навчання та потребу в більш адаптованих алгоритмах, демонструючи складність визначення оптимальної стратегії редукції лямбда-термів.

Продемонстроване розуміння моделями машинного навчання будови лямбда-термів та можливості вдосконалення в майбутньому завдяки цьому процесу редукції, незважаючи, на складнощі пов’язані з базовою природою та, потенційно, адаптивністю моделей OpenAI Embeddings.

4.2.2 Аналіз результатів проведених експериментів

Як вже обговорювалося раніше, за отриманими результатами даного етапу дослідження можна зробити висновок про те що виявлення більш продуктивної стратегії редукції по відношенню до конкретних лямбда термів постає складним завданням, що більш того є принципово нерозв’язним, з точки зору математичних засобів, як вказує робота [23]. Описана складність також підкреслюється

відсутністю універсального рішення для вибору оптимальної стратегії в усіх можливих умовах. Проте варто зазначити, що при виконанні деяких умов, та певних обмежень, залишається можливою розробка деяких життєздатних методів. Оскільки в описаному підході були використані штучно згенеровані лямбда-терми, що дозволяє зменшити витрати на збір даних, проте, як і було описано раніше, сформований датасет може не описувати достатньо чітко всіх існуючих залежностей реальних лямбда-термів та функціональних програм.

На даному етапі роботи було трансформовано лямбда-терми у вектори вбудовувань розмірністю 1536 з використання моделі OpenAI Embeddings. Було проведено аналіз з використання алгоритмів зменшення розмірності усереднених вбудовувань PCA та t-SNE для подальшої візуалізації даних, що дозволив візуально підтвердити початкову гіпотезу про роздільність даних термів та потенціалу у використанні кластерного аналізу. В описаній частині роботи було розглянуто алгоритм кластеризації DBSCAN з використання евклідової метрики та процес формування кластерів з його допомогою.

В ході даного етапу дослідження можна виділити здатність моделей вбудовувань OpenAI Embeddings для вилучення важливих характеристик та атрибутів лямбда-термів. Проте слід зазначити, що тренування моделей вбудовувань OpenAI Embeddings проводилося на явному представленні лямбда-термів, на загальнолюдському тексті та коді, що може ускладнювати точне представлення лямбда-термів у матрицях вбудовувань.

4.3. Редукція термів лямбда-числення: оцінка прогнозивних здатностей LLM

Даний етап дослідження також можна віднести до роду альтернативних підходів. Основна ідея полягає у використанні LLM для безпосередньо проведення процесу редукції лямбда термів. Тобто приймаючи на вхід лямбда терм LLM лямбда терм у текстовому представленні, що представляє собою наступний крок редукції. Мета описаного підходу полягає у кращому розумінні обчислювальних можливостей LLM відносно проблем лямбда-числення.

Великі мовні моделі (LLM), такі як GPT-4, є потужними аналітичними інструментами, які використовують їх здатність обробляти та генерувати текст, із великих наборів даних. Ці моделі чудово аналізують величезні обсяги текстових даних для вилучення закономірностей, настроїв і розуміння, що особливо корисно в таких сферах, як дослідження ринку для розуміння настроїв споживачів. LLM також автоматизують створення звітів шляхом узагальнення складних даних, вилучення ключової інформації та створення зв'язних наративів для легшого розуміння зацікавленими сторонами. Вони можуть передбачати майбутні тенденції, навчаючись на історичних даних, таким чином прогнозуючи рух ринку на основі минулих фінансових звітів і статей новин. Крім того, LLM покращує процес прийняття рішень, надаючи детальну довідкову інформацію, потенційні результати та наслідки рішень за допомогою швидкої обробки інформації та можливостей перехресних посилань. Як лідери в обробці природної мови (NLP), LLM надають необхідні інструменти для розпізнавання мовлення, мовного перекладу та семантичного пошуку, що має вирішальне значення для вилучення корисних даних із вхідних даних природною мовою. Крім того, їх можна інтегрувати з іншими моделями штучного інтелекту та аналітичними інструментами для розширення їхніх можливостей, таких як уточнення результатів алгоритмів інтелектуального аналізу даних або інтерпретація результатів статистичного аналізу в більш доступному форматі, що робить LLM безцінними для підвищення швидкості, точності та глибини аналіз даних у різних доменах.

Таблиця. 4.4.

Порівняння моделей GPT-3.5 і GPT-4.

Model name	**Weights Number**	**Price of input per million tokens**	**Price of output per million tokens**
GPT-3.5	~20 Billion	0.50$	1.50$
GPT-4	~220 Billion	30.00$	60.00$

Для проведення експериментів було обрано дві LLM – GPT-3.5 і GPT-4, що відрізняються за кількістю ваг, а отже і рівнем розуміння проблеми, а також за вартістю використання [105]. GPT-3.5 і GPT-4 є вдосконаленими ітераціями моделей Generative Pre-trained Transformer від OpenAI, розроблених для створення людиноподібного тексту, на основі промптів. GPT-3.5 є проміжним вдосконаленням порівняно з GPT-3 із розширеними можливостями аргументації та зменшенням упереджень, тоді як GPT-4 ще більше покращує це завдяки більшій кількості навчальних даних, більшому розміру моделі та покращеній продуктивності в широкому діапазоні мов і завдань, у тому числі складне міркування та глибше розуміння контексту. Дані моделі представлені у вигляді простого API, тож всі обчислення відбуваються на стороні серверів OpenAI. В табл. 4.4 наведено їх порівняльний аналіз.

4.3.1 Результати досліджень прогнозивних здатностей LLM відносно лямбда-числення

Описані моделі приймають на вхід промпт, що по своїй суті є деяким набором інструкцій та завдань у текстовому представленні. Від складності та точності промту напряму залежить якість відповіді моделей, треба також пам’ятати, про те, що чим більший є промпт, тим більшу вартість він складає для використання API. Нижче наведені три типи промптів, що були використані у ході експериментів. Вони відрізняються за своєю складністю та підходом. В кожному з промптів замінюється вхідний терм.

```
f"""
Please generate the next step of reduction a lambda term. Provide only term
expression. Use the rightmost innermost (RI) strategy.
The RI strategy use the latest redex for reduction. Pay attention on this fact.

Lambda term: '''{str_term}'''
"""
```

Рис. 4.2 Найпростіший промпт-команда для моделі GPT-4 для генерації наступного терму відповідно до стратегії LO.

```
f"""Given the lambda term, apply the leftmost-outermost (LO) strategy to perform the next step of reduction. The LO strategy, also known as normal order reduction, prioritizes the reduction of the leftmost-outermost redex first. This means that if there's a choice between reducing an expression inside a lambda abstraction or an application outside, the application takes precedence unless there's no other redex outside the abstraction.

Lambda Calculus Reduction Rules:
1. Alpha Conversion (α-conversion): Rename bound variables, ensuring no variable name conflicts. This step is essential for avoiding collisions between variables.
2. Beta Reduction (β-reduction): Apply the function to its argument. The formal rule is ((λx.M) N) → M[x:=N], where M[x:=N] denotes substituting N for x in M.
3. Eta Conversion (η-conversion): Simplify functions with unnecessary abstractions. The rule is λx.(M x) → M if x does not appear in M.

Prioritization in LO Strategy:
- Outermost First: Reduce the outermost redex before any inner redexes, even if the inner one is to the left of an outer one.
- Leftmost First: When faced with multiple outermost redexes, choose the leftmost one.

Examples:
- Given (λx.x x) ((λy.y) z), the LO strategy first reduces the outermost leftmost redex, resulting in (λx.x x) z.
- For ((λx.λy.x y) (λa.a)) b, the first step of reduction under LO strategy would yield (λy.(λa.a) y) b.

Procedure:
- Identify the leftmost-outermost redex in the term.
- Apply the appropriate reduction rule based on the structure of this redex.
- If multiple steps are available, choose the one that aligns with the LO strategy's prioritization.

Take your time to analyze the term <<<{str_term}>>. Consider each part of the term carefully and apply the reduction rules as described. Remember to use α-conversion to avoid variable naming conflicts, especially when dealing with nested lambda expressions. In the end provide the next reduction term in format: Result: next step term
"""
```

Рис. 4.3. Описовий промпт для моделі GPT-3.5 для генерації наступного терму відповідно до стратегії LO.

```
f"""Example of performing task #1:
Given term: (λx.((λy.((λz.z) x)) (λa.a))). Provide the next step of term reduction

Your output:
1. Identify the leftmost-outermost redex in the given term: ((λy.((λz.z) x)) (λa.a))
1.1. Where object of the redex is (λy.((λz.z) x)) (λa.a)
1.2. And subject of the redex is
2. Apply β-reduction: ((λy.((λz.z) x)) (λa.a)) [x:= (λa.a)]
3. Result: (λy.((λz.z) (λa.a)))
Example of performing task #2:
Given term: (((λx.x) (λy.(y (λz.z)))) (λa.a)). Provide the next step of term reduction.

Your output:
1. Identify the leftmost-outermost redex in the given term: (((λx.x) (λy.(y (λz.z)))) (λa.a))
2. Apply β-reduction: ((λy.(y (λz.z))) (λa.a)) [x:= (λy.(y (λz.z)))]
3. Result: ((λy.(y (λz.z))) (λa.a))

Given term: {str_term} Provide the next step of term reduction using example.

Your output:
"""
```

Рис. 4.4. Детальний покроковий промпт для моделі GPT-3.5 для генерації наступного терму відповідно до стратегії LO.

Таблиця 4.5.

Точність прогнозів наступних кроків редукції за допомогою моделей GPT-3.5 і GPT-4.

	GPT-3.5 to LO (77 tokens)	**GPT-3.5 to LO (40 tokens)**	**GPT-3.5 to RI (77 tokens)**	**GPT-3.5 to RI (40 tokens)**	**GPT-4 to LO (40 tokens)**	**GPT-4 to RI (40 tokens)**
Description prompt	9.5%	23.6%	6.47%	17.04%	–	–
Detailed step prompt	4.0%	10.82%	3.0%	9.83%	–	–
Simplest command prompt	10.0%	27.21%	3.23%	9.83%	41.3%	36.39%

В роботі були використані два типи датасетів, перший, коли наступний крок редукції було виконано з використанням LO стратегії, та другий з відповідно RI стратегією. Описані промти на Рис. 4.2, Рис. 4.3 та Рис. 4.4 були використані для різних моделей та різних датасетів, великі промти не були використані для моделі GPT-4 через високу вартість. В табл. 4.5 наведені результати експериментів.

4.3.3 Аналіз результатів досліджень прогнозивних здатностей LLM відносно лямбда-числення

Виходячи з низької точності моделі при використанні більш складних промптів, які включають в себе детальний покроковий опис, можна зробити висновок, що такі промти перенавантажують модель зайвим текстом та інформацією. Також можна сказати що модель GPT-3.5 не здатна достатньо глибоко зрозуміти та проаналізувати лямбда-терми, оскільки при зменшенні максимальної кількості вхідних токенів відбувалося підвищення точності. Оскільки справжні програми можуть містити десятки та сотні змінних, це може ускладнити подальший аналіз. Результати точності є нижчими для моделей при прогнозування стратегії RI на 5-7% порівняно зі стратегіями LO, це можна пояснити неповним розумінням моделями GPT-3.5 і GPT-4 розуміння концепції редексу та процесу редукції. Також на зменшення точності могло вплинути те, що в тренувальних даних моделей частіше зустрічалися приклади редукції за стратегією LO, оскільки моделі навчаються на великих об’ємах загального тексту та коду, дана стратегія є загальноприйнятою та більш популярною

В процесі дослідження вдалося збільшити точність передбачення наступного кроку редукції за стратегією RI завдяки використанню описового промту, що більш детально розкриває суть задачі. Тож можна сказати що моделі здатні покращити своє розуміння процесу редукції, але це залежить від побудови промту. Найкращі показники точності для моделі GPT-4 були на 10% вищими за найкращі показники для моделі GPT-3.5, з чого можна зробити висновок, що значна кількість ваг моделі не дає значного приросту у точності. Проте треба зауважити, що найточнішою

моделлю за результатами експериментів була найбільша протестована модель GPT-4. За результатами дослідження можна сказати, що LLM загального призначення, не є придатними у вирішенні задачі прогнозування наступного кроку редукції. В даному випадку варто звернути увагу на процес тонкого налаштування, наприклад у відношенні невеликих моделей загального призначення з відкритим кодом. З недоліків проведених експериментів можна виділити використання не всіх побудованих типів промптів у відношенні моделі GPT-4, оскільки через велику вартість генерації великі промти не були використані. Також через мінусів можна зазначити те, що робота проводилася лише з моделями OpenAI. Враховуючи все вищесказане, в якості наступного етапу дослідження можна розглянути використання підходу тонкого налаштування певних LLM для досягнення більш точних результатів.

В ході даного етапу дослідження були вирішені такі задачі:

– Підготовка навчального та тестового наборів даних, що представляють собою пари терм та його наступний крок редукції, з урахуванням обмежень кількості вхідних токенів для обраних LLM. Для проведення експериментів було обрано дві стратегії редукції LO та RI для більш глибокого аналізу. Це дозволило побачити більш широко розуміння моделями-лямбда числення та побачити різницю у розумінні стратегій редукцій.

– Були проведені експерименти з прогнозування наступного кроку редукції лямбда-термів з використанням моделей GPT-3.5 та GPT-4. Для перевірки надійності отримані прогнози були відфільтровані.

– За допомогою розробленого програмного забезпечення текстове представлення термів було переведено до представлення об’єктів програмного коду, що дозволило точно порівнювати фактичні та прогнозовані лямбда-терми з точністю до імен змінних. Застосовність моделей GPT-3.5 та GPT-4 для вирішення задачі прогнозування наступного кроку редукції було перевірено з використанням декількох варіантів промптів. Було зроблено висновок, що обрані LLM недостатні для вирішення даної задачі, зокрема у відношенні термів великого розміру. Даним

моделям не вистачає глибшого розуміння процесу редукції та відмінностей у стратегіях.

Можна зробити такі висновки:

1. Було підготовлено навчальний та тестовий набори даних з урахування особливостей та обмежень використаних архітектур LLM;

2. Згенерований набір даних дозволив проаналізувати, рівень розуміння моделями процесу редукції лямбда-термів та можливість працювати з двома різними обраними стратегіями редукції LO та RI;

3. Використовуючи моделі GPT-3.5 і GPT-4 були зпрогнозовані наступні кроки редукції, прогнози у текстовому представленні були в подальшому відфільтровані та трансформовані у об'єктне представлення для перевірки їх надійності.

За допомогою розробленого програмного оточення Lambda Calculus Environment було додатково оброблено прогнози та їх було порівняно з фактичними відповідями для точної відповідності по іменах змінних. В ході експериментів було розглянуто можливість застосування моделей GPT-3.5 та GPT-4, за результатами яких можна сказати що представлених моделей недостатньо для використання у якості інтерпретатора та розуміння процесу редукції лямбда-термів, та розуміння відмінностей у стратегіях редукції, особливо для більших термів.

4.4. Опис можливої імплементації методу оптимізації та верифікації в лямбда-численні.

Можливим варіантом імплементації підходу до підвищення продуктивності виконання функціональних програм може бути використання архітектур LLM для прогнозування більш вигідної стратегії редукції для обраного терму.

Оскільки функціональна програма може бути трансформована до рівня лямбда-числення та буде представляти собою складний лямбда-терм. Даний підхід може бути масштабований до рівня функціональних мов програмування. На даному етапі ми маємо можливість прогнозування кількості кроків редукції лямбда

терму для обраної стратегії, тож обравши набір доступних стратегій для редукції ми можемо прогнозувати кількість кроків для кожної стратегії та обрати найпродуктивнішу з точки зору прогнозу.

Варто також приділити увагу аналізу таких прогнозів з точки зору витрат обчислювальних ресурсів, оскільки використання LLM в деяких випадках може нести під собою виконання ресурсоємних операції та може потребувати використання спеціального технічного забезпечення.

Проте на даний момент використання LLM стає все більш дешевим та обчислювально ефективним.

З першого погляду можна сказати що перед кожною компіляцією програми на необхідно буде виконати операцію прогнозування кількості кроків для даної програми для певного набору стратегії.

В майбутньому можливо ми будемо працювати вже на рівні функціональних програм, а не лямбда-числення, проте зараз можливо знадобиться попередня обробка програми перетворенням її у лямбда-терм.

Перед компіляцією необхідно буде виконати процедуру прогнозування кількості кроків декілька разів, вибрати найефективнішу стратегію згідно з прогнозом, та провести саму компіляцію. Скоріше за все даний підхід має бути ефективним оскільки ми виконуємо всього N викликів LLM на один раз компіляції, де N відповідає кількості доступних стратегій, проте варто пам’ятати, що розмір лямбда терму що буде переданий на вхід може бути достатньо великим. Можна припустити що перетворення програми написаної стандартними мовами програмування у лямбда терм, може збільшити її об’єм в декілька разів.

Тож для вирішення даної задачі варто пам'ятати про доступний розмір входу LLM. Також однією із задач має бути збір набору даних реальних програм написаних людиною, даний набір має бути достатньо великим за розміром, орієнтовно, можливо мало би сенс використовувати для цього наприклад весь доступний відкритий код з репозиторію GitHub для певної мови програмування, та

в ідеалі навчити LLM з нуля на цих даних з або без перетворення вхідних даних на лямбда терми.

Даний підхід мав би показати себе краще ніж використання LLM загального призначення для роботи з текстовою інформацією та кодом.

В ідеалі наступним етапом могла би бути побудова інструменту, що підлаштовує свою параметри безпосередньо для коду певного користувача. Тобто загальний інструмент був навчений на усьому доступному коді, але коли використовується певним користувачем, то адаптується під його стиль написання коду, задачі що вирішує код, застосовані патерни та бібліотеки в коді, з кожною компіляцією.

Також варто приділити увагу ризикам, що можуть виникати на кожному з етапів імплементації представленого рішення. На Рис. 4.5 приведено діаграму можливого імплементації описаного підходу. Тут LLM представляє собою нейронну мережу, натреновану для класифікації функціональної програми, прогнозуючи найпродуктивнішу стратегію. Це також може бути декілька LLM що прогнозують кількість кроків редукції, з яких потім обирається стратегія з найменшою кількістю кроків. Виконавчими сутностями тут є компілятори і інтерпретатори. На вхід подається програма, а на виході отримується прогноз найпродуктивнішою стратегії.

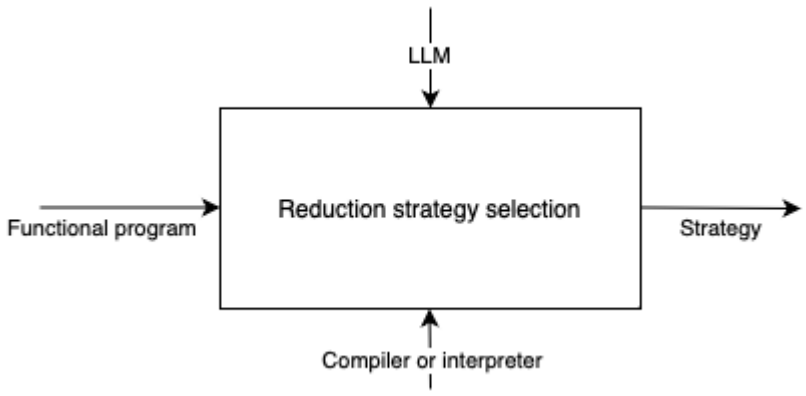


Рис. 4.5. Діаграма IDEF0 можливої імплементації описаного підходу.

В процесі роботи також були проаналізовані можливі методи використання великих мовних моделей у якості засобу верифікації програмного забезпечення, зокрема лямбда термів.

Великі мовні моделі (LLM), такі як GPT та його наступники, продемонстрували виняткові можливості у створенні на розумінні природної мови, а також структурованого коду. Оскільки LLM продовжують розвиватися, їх застосування в завданнях програмування розширилося, включаючи генерацію коду, оптимізацію та перевірку. Далі продемонстровано, як LLM можна використовувати для перевірки лямбда-термів, основоположної концепції лямбда-числення та функціонального програмування. Досліджено потенціал LLM для надання допомоги у формальній перевірці властивостей лямбда-терму, таких як правильність, завершення та еквівалентність, і як вони можуть автоматизувати частини послідовності перевірки. LLMs, з їхньою здатністю навчатися на величезних корпусах коду та математичних міркувань, пропонують новий підхід до перевірки лямбда-термів. Розуміючи та генеруючи структури, подібні до коду, LLM можуть допомогти у перевірці лямбда-термів у такі способи:

• LLM може бути навчена для генерації формальних доказів властивостей лямбда-термів шляхом розпізнавання загальних шаблонів у функціональному коді та застосування логічних міркувань. Наприклад, маючи лямбда-терм і бажану властивість (таку як безпека типу або еквівалентність), LLM може створити покрокове підтвердження або запропонувати перетворення, які зберігають правильність. Цей процес може включати перевірку редукцій, призначення типів і переписування термів, що забезпечує більш автоматизований робочий процес перевірки.

• Навчаючи LLM на великих наборах даних перевірених лямбда-термів і їх доказів, моделі можуть навчитися розпізнавати верифікаційні шаблони та надавати перевірені кроки для лямбда-термів. Це може скоротити час і зусилля, необхідні для перевірки вручну, зберігаючи при цьому формальні гарантії.

Одним із ключових аспектів перевірки лямбда-термів є забезпечення безпечності типів термів. Виведення типу — це процес автоматичного визначення типу лямбда-терму без явних анотацій типу. LLMs можуть допомогти в цьому процесі, аналізуючи лямбда-вирази та прогнозуючи найбільш імовірні типи для змінних і функцій на основі контексту. Крім того, LLM можуть виконувати перевірку типу, гарантуючи, що лямбда-терми правильно типізовані відповідно до правил лямбда-числення.

• LLM можуть допомогти у перевірці правильності анотацій типу та виявленні потенційних помилок типу в лямбда-термах. Таким чином вони оптимізували б процес перевірки відповідності лямбда-термів їх очікуваним типам, критичний крок у забезпеченні коректності програми.

• Лямбда-числення значною мірою покладається на рівняння, де різні лямбда-терми вважаються еквівалентними, якщо їх можна привести до тієї самої нормальної форми за допомогою бета-редукції або інших перетворень. LLM можуть полегшити міркування на основі рівнянь, визначаючи еквівалентності між лямбда-термінами, спрощуючи складні терми та пропонуючи перетворення, які призводять до еквівалентних, але більш ефективних форм. Наприклад, маючи два лямбда-терми, LLM може визначити, чи є вони еквівалентними, генеруючи низку скорочень і перетворень, які відображають один терм в інший. Це може допомогти перевірити оптимізацію у функціональних програмах, які покладаються на лямбда-числення, гарантуючи, що перетворення зберігають семантику оригінальних лямбда-термів.

• LLM можуть допомогти в аналізі складності лямбда-термів шляхом оцінки кількості кроків скорочення, необхідних для досягнення нормальної форми. Це надасть розробникам уявлення про характеристики продуктивності лямбда-виразів і допоможе виявити потенційну неефективність функціонального коду.

Незважаючи на те, що LLM пропонують значний потенціал для автоматизації перевірки лямбда-термів, їхні результати слід поєднувати з формальними інструментами перевірки для забезпечення правильності. LLM чудово

справляються з розпізнаванням образів і генерацією коду, але їм може бракувати строгості, необхідної для повних формальних доказів. Щоб вирішити цю проблему, LLM можна інтегрувати в існуючі офіційні верифікаційні підходи, де вони служать помічниками, а не заміною традиційних методів перевірки.

Висновок до розділу 4

В даному розділі було представлено декілька альтернативних підходів для застосування LLM у розрізі вирішення проблем лямбда-числення. Описані методи включають у себе кластерний аналіз з використанням моделей з відкритим кодом та пропріетарних моделей, а також використання моделей як засобу редукції лямбда-термів.

Було проведено обширний кластерний аналіз з використанням моделей з відкритим кодом, таких як Microsoft CodeBERT. Що були натреновані для роботи з мовами програмування загального призначення. Виходи середніх шарів даної моделі були використані для побудови вбудовувань, що в подальшому були використані для кластеризації з застосуванням декількох алгоритмів. Для кластеризації були використані такі методи як DBSCAN та HAC, результати кластеризації були візуалізовані з використанням методів зменшення розмірності, такими як PCA та t-SNE. В ході візуального аналізу було виявлено можливість розділення існуючого простору лямбда термів на окремі множини, що підтвердило початкову гіпотезу.

Також було використано моделі вбудовувань OpenAI Embeddings, з якими можна працювати через використання простого API. Слід зазначити, що моделі вбудовувань OpenAI Embeddings мають більший об’єм ваг, порівняно з Microsoft CodeBERT та були натреновані для вирішення задача загального призначення на людському тексті та мовах програмування. Виходи даних моделей також були використані для побудови вбудовувань та проведення кластерного аналізу. Кластерний аналіз показав доволі схожі результати з попередніми.

В підрозділі 4.3 було представлено аналіз методу використання LLM для прогнозування наступного кроку редукції лямбда термів, тобто фактично проведення редукції терму методами LLM. Було згенеровано датасет, що складався з терму у текстовому представленні, та його наступного кроку редукції також у текстовому представленні. Було використано звичайне представлення термів. Для аналізу були обрані дві мовні моделі GPT-3.5 та GPT-4. Були побудовані три типи промптів, що подаються на вхід моделям та відрізнялися за розміром і тим, наскільки точно вони описують поставлену задачу. Результати експериментів показали недостатню точність LLM у вирішенні цієї задачі. Найкращі результати були продемонстровані моделлю GPT-4. Основною ідеєю даного підходу можна назвати аналіз розуміння та обчислювальних можливостей LLM відносно лямбда-числення.

В підрозділі 4.4 було представлено підхід для імплементації методів використання існуючих розробок для вирішення задач та проблем пов’язаних з лямбда-численням, зокрема для вибору більш ефективної стратегії по відношенню до лямбда-терму. В майбутньому ціла програма буде представлена у вигляді лямбда-терму, що дозволить перед етапом компіляції або інтерпретації вибрати більш вигідну стратегію редукції саме для цієї програми. При імплементації даного підходу варто також брати до уваги обчислювальні витрати на виклики моделей машинного навчання, оскільки в деяких випадках занадто велика кількість викликів LLM, може нівелювати переваги отримані з точки зору обчислювальних вартостей в розрізі лямбда-числення. Також у підрозділі 4.4 продемонстровано підхід для використання великих мовних моделей у якості інструменту верифікації лямбда термів, та оптимізації їх роботи та редукції. Було запропоновано декілька можливих варіантів імплементації, описано їх сильні сторони. Поєднання оптимізації та верифікації функціональних програм є дуже вигідною комбінацією з точки зору використання великих мовних моделей та засобів штучного інтелекту в цілому.

ВИСНОВКИ

1. У процесі аналізу функціональних мов програмування були виявлені їх особливості та нерозв'язані проблеми компіляції та інтерпретації. Встановлено, що лямбда-числення служить ідеальним тестовим майданчиком для дослідження процесу виконання функціональних програм. Лямбда-числення дозволяє робити це через процес редукції, і незважаючи на свій спрощений синтаксис, відтворює складність функціональних мов програмування.

2. В дисертації висвітлені наступні наукові результати.

– Вперше розроблено параметричну випадкову стратегію редукції, яка на відміну від існуючих дозволяє регулювати ймовірності вибору редексу. Це дозволяє контролювати процес редукції, тим самим роблячи його більш ефективним для специфічних підмножин редексів. Ця властивість відкриває нові можливості для оптимізації обчислювальної поведінки лямбда-термів у різних сценаріях.

– Удосконалено метод прогнозування складності редукції лямбда термів, що на відміну від існуючих забезпечує використання методів машинного навчання. Це дозволяє підвищити точність прогнозу обчислювальної складності кожного кроку редукції лямбда-термів.

– Удосконалено експериментальне середовище емуляції лямбда-числення завдяки впровадженню процедури багатоетапної верифікації розробленого програмного забезпечення та автоматичної генерації лямбда-термів. Це дозволило забезпечити точну реалізацію цього програмного забезпечення та створювати набори даних лямбда-термів для забезпечення теоретично необхідного покриття контрольованими конфігураціями лямбда-термів.

– Дістали подальшого розвитку методи представлення лямбда-термів у вигляді вбудовувань (embeddings), що на відміну від існуючих забезпечує використання штучних нейронних мереж на базі архітектури Transformer. Це

дозволяє покращити глибину аналізу коду лямбда-термів та підвищити точність екстрації характеристик, що впливають на процес редукції.

– Дістав подальшого розвитку комплексний підхід до оптимізації стратегій редукції, що відрізняється від існуючих прогнозуванням кількості кроків редукції для обраної стратегії з можливістю подальшого вибору стратегії нормалізації терму. Цей комбінований підхід є новим та має потенціал скоротити загальний час нормалізації, що напряму впливає на продуктивність виконання функціональних програм.

3. Розроблено програмну реалізацію оточення лямбда-числення та методології для його верифікації та забезпечення надійності виконання, що дозволяє інтерактивну роботу з лямбда-термами в середовищі Python та емуляцію виконання функціональних програм. Це оточення дозволяє штучну генерацію лямбда-термів для проведення експериментів та покрокової редукції лямбда-термів, трансляцію лямбда термів в текстові представлення різних структур, реалізацію необхідних стратегій редукції для лямбда-термів.

4. За допомогою розробленого оточення була оцінена можливість оптимізації стратегій редукції у лямбда-численні, що відображають процеси компіляції та інтерпретації функціональних мов програмування. Результати дослідження показали, що можливо оцінити обчислювальну нерівнозначність редексів лямбда термів та можливо розробити стратегію редукції, що враховувала б цю нерівнозначність. Показано, що використання передових методів машинного навчання для аналізу текстової інформації дозволяє екстрагувати дані, що впливають на кількість кроків редукції терму, та можливість подальшого вибору стратегії з найменшою кількістю кроків. Також виявлено можливість зменшення обчислювальних витрат на компіляцію та інтерпретацію, використовуючи інформацію про стан програми, залежно від обраної стратегії. Досліджено обчислювальні витрати та шляхи до їх оптимізації у процесах нормалізації. Показано, що можливо використати штучні нейронні мережі для розрізнення лямбда-термів за стратегіями редукції також було проаналізовано їх потенціал у

виборі оптимальних стратегій редукції для термів лямбда-числення. Також в дослідженні проаналізовано можливість редукції лямбда-термів за допомогою сучасних моделей машинного навчання. Було показано що використання великих мовних моделей, таких як GPT-3.5, GPT-4 для прогнозування наступного кроку редукції є недостатньо точним, та потребує подальшого аналізу. Було запропоновано можливий варіант імплементації існуючого рішення для забезпечення оптимізації та верифікації програм лямда-числення.

5. Наукові положення, висновки та рекомендації щодо подальшого дослідження мають достовірне викладення, що аргументоване отриманими результатами дослідження. Аргументація дослідження стало можливою завдяки використанню у роботі комплексного підходу, що включає широкий спектр методів математичної оптимізації, методів імітаційного та математичного моделювання, теорії множин, теорії графів, та методів машинного навчання для вирішення проблем редукції термів у безтиповому лямбда-численні.

Основні результати роботи було реалізовано в Харківському національному університеті імені В. Н. Каразіна у рамках НДР «Моделювання інформаційних процесів у складних і розподілених системах» за 2021 – 2023 рр. (ДР No 0121U109183).

Одним з практичних результатів застосування сучасних методів обробки текстової інформації було застосування моделі Microsoft CodeBERT, що була обрана з поміж інших моделей для роботи з програмним кодом, для екстракції даних з лямбда термів. Для цього було використано набір даних із 4-х тисяч штучно згеренованих лямбда термів, що розрізняються за кількістю кроків редукції по стратегіям Leftmost Outermost та Rightmost Innermost. До моделі CodeBERT подавалися терми у текстовому представленні, токенізація та підготовка проходила засобами самої моделі. На виході були отримані матриці вбудовувань (embeddings), що були усереднені до векторів розмірністю 768. Ці вектори вбудовувань були проаналізовані методами t-SNE та PCA на предмет їх роздільності. Також було проведене неінформоване розділення даних засобами кластеризації, а саме

DBSCAN та HAC, результати кластеризації було порівняно з очікуваними пріоритетами стратегії редукції та була отримана точність в 86.44% в розділенні отриманих вбудовувань термів за пріоритетом стратегії. Зважаючи на це зроблено висновок, що додаткове навчання моделі Microsoft CodeBERT може покращити точність виявлення найкращої стратегії для терму. Також подібні експерименти були проведені з моделями OpenAI широкого спектру задач, про їх застосування не показало значного покращення в порівнянні з вузькоспеціалізованою моделлю.

Результати роботи було впроваджено в якості матеріалів курсу «Особливості застосування методів машинного навчання» для студентів другого курсу магістратури факультету математики та інформатики Харківського національного університету ім. В.Н. Каразіна.

6. Отримані результати дослідження можуть бути рекомендовані до застосування в розробці програмного забезпечення та науково-дослідних організаціях для розроблення компіляторів та інтерпретаторів функціональних мов програмування.

7. Викладені в дисертації науково-практичні результати разом із підтвердженням їх достовірності, наукової та практичної значущості дозволяють вважати, що сформульована наукова задача оптимізації функціональних програм методами штучного інтелекту вирішена, а поставлена мета досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhenjiang Hu, John Hughes and Meng Wang. How functional programming mattered. National Science Review 2. 2015. Pp. 349-370. DOI: https://doi.org/10.1093/NSR%2FNWV042

2. J. Neumann. Advantages and disadvantages of functional programming. 2022. NEW IT Engineering, Medium. URL: https://medium.com/twodigits/advantages-and-disadvantages-of-functional-programming-52a81c8bf446

3. Adam L. Davis. Functional Programming. Learning Groovy 3. 2019. DOI: https://doi.org/10.1007/978-1-4842-5058-7_9

4. Thomas Bandt. Who Cares About Functional Programming? 2024. URL: https://thomasbandt.com/who-cares-about-functional-programming

5. David Whitney. Does Functional Programming Make Your Code Hard To Read? 2020. URL: https://davidwhitney.co.uk/Blog/2020/10/14/functional_code_hard_to_read

6. Robert Butler. Why Are Developers Falling in Love (Again) With Functional Programming? URL: https://www.bairesdev.com/blog/developers-love-functional-programming/

7. Nikolay Mozgovoy. The Pillars Of Functional Programming. 2019. URL: https://sigma.software/about/media/pillars-functional-programming-part-1F

8. Pavel Ryzhov. Haskell Financial Data Modeling and Predictive Analytics. 2013.

9. Roman Alterman. Why Fintech Companies Use Haskell. 2020. URL: https://serokell.io/blog/functional-programming-in-fintech

10. Jason Mcclellan. Scala is Ready to Power the Future of Financial Services Today. 2023. URL: https://articles.xebia.com/scala-is-ready-to-power-the-future-of-financial-services-today

11. Maria Kucharczyk. FinTech software development with Scala. 2020. URL: https://softwaremill.com/fintech-software-development-scala/F

12. Joe Armstrong. A history of Erlang. Proceedings of the third ACM SIGPLAN conference on History of programming languages. 2007. Pp. 1–26. DOI: https://doi.org/10.1145/1238844.1238850

13. Simon St. Laurent and J. David Eisenberg. Introducing Elixir: Getting Started in Functional Programming. Computer Science. O'Reilly Media, Inc. 2014.

14. Sandy Ryza, Uri Laserson, Sean Owen and Josh Wills. Advanced Analytics with Spark. O'Reilly Media, Inc. 2015.

15. Stuart Halloway. Programming Clojure. Pragmatic Bookshelf. 2009.

16. Ricardo Lanziano. IoT Complexity Made Simple with the Versatility of Erlang and Elixir. 2023. Erlang Solutions Blog. 2023. URL: https://www.erlang-solutions.com/blog/iot-complexity-made-simple-erlang-elixir/

17. Dr Jon Harrop. Disadvantages of purely functional programming. Devmio Blog. 2016. URL: https://devm.io/programming/disadvantages-of-purely-functional-programming-126776

18. Ties van de Ven. The Problem with Functional Programming. Foojay.io. 2022. URL: https://foojay.io/today/the-problem-with-functional-programming/

19. A. Aaby. Functional Programming. 1996. URL: https://www.cs.jhu.edu/~jason/465/readings/lambdacalc.html

20. Greg Michaelson. An Introduction To Functional Programming Through Lambda Calculus. URL: https://www.macs.hw.ac.uk/~greg/books/gjm.lambook88.pdf

21. Jesse Alama and Johannes Korbmacher. The Lambda Calculus. The Stanford Encyclopedia of Philosophy (Winter 2023 Edition). 2023. URL: https://plato.stanford.edu/archives/win2023/entries/lambda-calculus/

22. Franz Baader and Tobias Nipkow. Term rewriting and all that. 1998. DOI: https://doi.org/10.1017/CBO9781139172752.008

23. H.P. Barendregt. The Lambda Calculus: Its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics. 2012. URL: https://books.google.com.ua/books?id=b8jsMQEACAAJ

24. Johan van Benthem. A manual of intensional logic. Stanford: CSLI Publications. 1989. DOI: https://doi.org/10.2307/2274837

25. S. Wildekker, N Richardstatma, Henk Barendregt, Wil Dekkers, Richard Statman, and others. Lambda Calculus with Types. Computer Science, Mathematics. 2014. DOI: https://doi.org/10.5860/choice.51-5067

26. Gottlob Frege. Grundgesetze der Arithmetik (The Basic Laws of Arithmetic). Jena: Verlag Hermann Pohleю 1893.

27. Biernacka, Małgorzata, Witold Charatonik and Tomáš Dráb. The Zoo of Lambda-Calculus Reduction Strategies, And Coq. International Conference on Interactive Theorem Proving. 2022. DOI: https://doi.org/10.4230/LIPIcs.ITP.2022.7

28. Tomáš Dráb. Reduction Strategies in the Lambda Calculus and Their Implementation through Derivable Abstract Machines: Introduction. ArXiv, Computer Science. 2024. DOI: https://doi.org/10.48550/arXiv.2405.12586

29. Chuck C. Liang, Gopalan Nadathur and Xiaochu Qi. Choices in Representation and Reduction Strategies for Lambda Terms in Intensional Contexts. Journal of Automated Reasoning 33. Pp. 89–132. 2004. DOI: https://doi.org/10.1007/s10817-004-6885-1

30. Giulio Guerrieri. Head reduction and normalization in a call-by-value lambda-calculus. 2nd International Workshop on Rewriting Techniques for Program Transformations and Evaluation. 2015. DOI: https://doi.org/10.4230/OASIcs.WPTE.2015.3

31. Vladyslav Shramenko, Victoriya Kuznietcova and Grygoriy Zholtkevych. Studying Mixed Normalization Strategies of Lambda Terms. International Workshop of IT-professionals on Artificial Intelligence. Vol. 3348. 2022. URL: https://ceur-ws.org/Vol-3348/paper5.pdf

32. K. Berkling. Head order reduction: A graph reduction scheme for the operational lambda calculus. In: Fasel, J.H., Keller, R.M. (eds) Graph Reduction. GR 1986. Lecture Notes in Computer Science, vol 279. Springer, Berlin, Heidelberg. 1987. DOI: https://doi.org/10.1007/3-540-18420-1_48

33. Eric Walkingshaw. CS 583: Advanced Functional Programming. Course of lectures in Oregon State University. 2021. URL: https://web.engr.oregonstate.edu/~walkiner/teaching/cs583-sp21/

34. Haskell/Lazy evaluation. Haskel Documentation. 2021. URL: https://wiki.haskell.org/Haskell/Lazy_evaluation

35. Simona Kasterovic and Michele Pagani. The Discriminating Power of the Let-In Operator in the Lazy Call-by-Name Probabilistic lambda-Calculus. International Conference on Formal Structures for Computation and Deduction. 2019. DOI: https://doi.org/10.48550/arXiv.2205.14916

36. Dimitur Nikolaev Krustev. Optimizing Program Size Using Multi-result Supercompilation. Proceedings 8th International Workshop on Verification and Program Transformation and 7th Workshop on Horn Clauses for Verification and Synthesis, VPT/HCVS@ETAPS 2020, Dublin, Ireland. 2020. DOI: https://doi.org/10.4204/EPTCS.320.9

37. Peter Battyanyi and Karim Nour. Normalization in the simply typed λμμ'ρθε-calculus Mathematical Structures in Computer Science. No. 32. Pp. 1066–1098. 2022. DOI: https://doi.org/10.1017/S096012952200041X

38. Michele Pagani and Paolo Tranquilli. Parallel Reduction in Resource Lambda-Calculus. Asian Symposium on Programming Languages and Systems. 2009. DOI: https://doi.org/10.1007/978-3-642-10672-9_17

39. R. Mammadli, A. Jannesari, and F. A. Wolf. Static Neural Compiler Optimization via Deep Reinforcement Learning. IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar). Pp. 1–11. 2020. DOI: https://doi.org/10.1109/LLVMHPCHiPar51896.2020.00006

40. Margherita Zorzi. On quantum lambda calculi: a foundational perspective. Mathematical Structures in Computer Science. No. 26. Pp. 1107–1195. 2014. DOI: https://doi.org/10.1017/S0960129514000425

41. C. A. Soares, Lucas De Souza Batista, Felipe Campelo and Frederico Gadelha Guimarães. Computation of Mixed Strategy Non-dominated Nash Equilibria in Game Theory. 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence. Pp. 242–247. 2013. DOI: https://doi.org/10.1109/BRICS-CCI-CBIC.2013.47

42. M. Mitzenmacher, E. Upfa. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. 2005.

43. O. Deineha, V. Donets, G. Zholtkevych. On Randomization of Reduction Strategies for Typeless Lambda Calculus. In: Antoniou, G., et al. Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI 2023. Communications in Computer and Information Science. Springer, Cham. Vol. 1980. 2023. DOI: https://doi.org/10.1007/978-3-031-48325-7_3

44. Mariangiola Dezani-Ciancaglini, Simona Ronchi Della Rocca, and Lorenza Saitta. Complexity of lambda-term reductions. RAIRO Theor. Informatics Appl. 13. Pp. 257-287. 1979. DOI: https://doi.org/10.1051/ita/1979130302571

45. Katarzyna Grygiel, Pierre Lescanne. Counting and Generating Terms in the Binary Lambda Calculus. 2015. DOI: https://doi.org/10.1017/S0956796815000271

46. 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

47. Eyal Wirsansky. Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems. Packt Publishing Ltd. 346p. 2020.

48. 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. Vol. 15. Feb. 2019. DOI: https://doi.org/10.23638/LMCS-15(1:16)2019.

49. Clemens Grabmayer. Linear Depth Increase of Lambda Terms along Leftmost-Outermost Beta-Reduction. ArXiv. Nov. 2019. DOI: https://doi.org/10.48550/arXiv.1604.07030

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

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

52. 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

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

54. Oleksandr Deineha, Volodymyr Donets and Grygoriy Zholtkevych. Estimating Lambda-Term Reduction Complexity with Regression Methods. International Conference "Information Technology and Interactions". Vol. 3624. No. 13. Pp. 147–156. 2023. URL: https://ceur-ws.org/Vol-3624/Paper_13.pdf

55. 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: https://doi.org/10.38094/jastt1457

56. Chang Liu, Zhenhua Hu, Yan Li, Shaojun Liu. Forecasting copper prices by decision tree learning. Resources Policy, Vol. 52, Pp. 427-434. ISSN 0301-4207. 2017. DOI: https://doi.org/10.1016/j.resourpol.2017.05.007

57. Smola and B. Schölkopf. A tutorial on support vector regression. Statistics and Computing 14. Pp. 199–222. 2004. DOI: https://doi.org/10.1023/B:STCO.0000035301.49549.88

58. T. Cover, and P. Hart. Nearest neighbor pattern classification. In IEEE Transactions on Information Theory. Vol. 13. No. 1. Pp. 21–27. Jan. 1967. DOI: https://doi.org/10.1109/TIT.1967.1053964

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

60. Aki Vehtari, Andrew Gelman and Jonah Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing. No. 27. Pp. 1413–1432. 2015. DOI: https://doi.org/10.1007/s11222-016-9696-4

61. Jasper Snoek, H. Larochelle and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. Neural Information Processing Systems. 2012. URL: https://www.semanticscholar.org/reader/2e2089ae76fe914706e6fa90081a79c8fe01611e

62. Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. No. 374. 2016. DOI: https://doi.org/10.1098/rsta.2015.0202

63. Edith Heiter, Bo Kang, Ruth Seurinck and Jefrey Lijffijt. Revised Conditional t-SNE: Looking Beyond the Nearest Neighbors. International Symposium on Intelligent Data Analysis. 2023. DOI: https://doi.org/10.1007/978-3-031-30047-9_14

64. 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). Pp. 3286–3292. Prague, Czech Republic, 2022. DOI: https://doi.org/10.1109/SMC53654.2022.9945302

65. 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. 2023. URL: https://arxiv.org/abs/2309.03241

66. 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

67. 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. 2023. URL: https://arxiv.org/abs/2309.07062

68. Ugo Dal Lago, Gabriele Vanoni. On randomised strategies in the λ-calculus. Theoretical Computer Science. Vol. 813. Pp. 100–116. 2020. DOI: https://doi.org/10.1016/j.tcs.2019.09.033

69. Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep Learning. Nature. No. 521. Pp. 436–444. 2015. DOI: https://doi.org/10.1038/nature14539

70. Liang Yao, Chengsheng Mao and Yuan Luo. Graph Convolutional Networks for Text Classification. AAAI Conference on Artificial Intelligence. 2018. DOI: https://doi.org/10.1609/aaai.v33i01.33017370

71. 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: https://doi.org/10.3390/app10175841

72. 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. 2020. DOI: https://doi.org/10.1145/3437963.3441667

73. Oleksandr Deineha, Volodymyr Donets and Grygoriy Zholtkevych. 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. URL: https://ceur-ws.org/Vol-3641/paper12.pdf

74. M. M. Krell, Bilal Wehbe. A First Step Towards Distribution Invariant Regression Metrics. 2020. URL: https://arxiv.org/abs/2009.05176

75. Yoshua Bengio, Paolo Frasconi. Input-output HMMs for sequence processing. IEEE transactions on neural networks. No. 7. Vol. 5. Pp. 1231–1249. 1996. DOI: https://doi.org/10.1109/72.536317

76. Bhavya Mor, Sunita Garhwal and Ajay Kumar. A Systematic Review of Hidden Markov Models and Their Applications. Archives of Computational Methods in

Engineering. No. 28. Pp. 1429–1448. 2020. DOI: https://doi.org/10.1007/s11831-020-09422-4

77. 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.

78. Josey Mathew, Chee Khiang Pang, Ming Luo and Weng Hoe Leong. Classification of Imbalanced Data by Oversampling in Kernel Space of Support Vector Machines. IEEE Transactions on Neural Networks and Learning Systems. No. 29. Pp. 4065–4076. 2018. DOI:https://doi.org/10.1109/TNNLS.2017.2751612

79. Adrian A. Hopgood. Rule-Based Systems. Encyclopedia of Social Network Analysis and Mining. 2021. DOI: https://doi.org/10.1007/978-1-4614-6170-8_100878

80. Long-Hao Yang, Jun Liu, Fei-Fei Ye, Yingming Wang, Chris D. Nugent, Haibo Wang and Luis Martínez. Highly explainable cumulative belief rule-based system with effective rule-base modeling and inference scheme. Knowl. Based Syst. No. 240. 2022. DOI: https://doi.org/10.1016/j.knosys.2021.107805

81. Vaswani, A., Shazeer, N.M., Parmar, N., Usz-koreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosu-khin, I. Attention is All you Need. Neural Information Processing Systems. Vol. 30. 2017

82. 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: https://doi.org/10.48550/arXiv.2303.18223

83. Ormerod, M., Devereux, B., & Martínez Del Rincón, J. How is a “Kitchen Chair” like a “Farm Horse”? Exploring the Representation of Noun-Noun Compound Semantics in Transformer-based Language Models. Computational Linguistics. Pp 1–30. 2023. DOI: https://doi.org/10.1162/coli_a_00495

84. Kowsari, Kamran, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text Classification Algorithms: A Survey. Information. No. 10(4). Vol. 150. 2019. DOI: https://doi.org/10.3390/info10040150

85. Sebastian Ruder. An overview of gradient descent optimization algorithms. ArXiv. Computer Science. Machine Learning. 2016. URL: https://arxiv.org/abs/1609.04747.

86. 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. 2023. URL: https://arxiv.org/abs/2304.10500.

87. Styawati Styawati, Andi Nurkholis, Ahmad Ari Aldino, Selamet Samsugi, Emi Suryati and Ryan Puji Cahyono. Sentiment Analysis on Online Transportation Reviews Using Word2Vec Text Embedding Model Feature Extraction and Support Vector Machine (SVM) Algorithm. 2021 International Seminar on Machine Learning, Optimization, and Data Science (ISMODE). Pp. 163–167. 2022. DOI: https://doi.org/10.1109/ISMODE53584.2022.9742906

88. Md. Shihab Mahmud, Md. Touhidul Islam, Afrin Jaman Bonny, Rokeya Khatun Shorna, Jasia Hossain Omi and Md. Sadekur Rahman. Deep Learning Based Sentiment Analysis from Bangla Text Using Glove Word Embedding along with Convolutional Neural Network. 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT). Pp. 1–6. 2022. DOI: https://doi.org/10.1109/ICCCNT54827.2022.9984392

89. Ghaith Alomari, Et al.. Transforming Text Generation in NLP: Deep Learning with GPT Models and 2023 Twitter Corpus Using Transformer Architecture. International Journal on Recent and Innovation Trends in Computing and Communication. 2023. DOI: https://doi.org/10.17762/ijritcc.v11i9.9463

90. New embedding models and API updates. Blog OpenAI. 2024. URL: https://openai.com/blog/new-embedding-models-and-api-updates

91. Hahsler, M., Piekenbrock, M., & Doran, D. DBSCAN: Fast Density-Based Clustering with R. Journal of Statistical Software. 2019. DOI: https://doi.org/10.18637/jss.v091.i01

92. Monath, N., Dubey, K.A., Guruganesh, G., Zaheer, M., Ahmed, A., McCallum, A., Mergen, G., Najork, M., Terzihan, M., Tjanaka, B., Wang, Y., & Wu, Y.

Scalable Hierarchical Agglomerative Clustering. Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2020. DOI: https://doi.org/10.1145/3447548.3467404

93. Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M.P., Ferrer, C.C., Grattafiori, A., Xiong, W., D'efossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., & Synnaeve, G. Code Llama: Open Foundation Models for Code. 2023. DOI: https://doi.org/10.48550/arXiv.2308.12950

94. Elnaggar, A., Ding, W., Jones, L., Gibbs, T., Fehér, T.B., Angerer, C., Severini, S., Matthes, F., & Rost, B. CodeTrans: Towards Cracking the Language of Silicone's Code Through Self-Supervised Deep Learning and High Performance Computing. ArXiv. 2021 URL: www.arxiv.org/abs/2104.0244

95. Replit. (n.d.-b). replit-code-V1-3B. Hugging Face. https://huggingface.co/replit/replit-code-v1-3b

96. Wang, Y., Wang, W., Joty, S.R., & Hoi, S.C. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. 2021. DOI: https://doi.org/10.18653/v1/2021.emnlp-main.685

97. 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. 2020. DOI: https://doi.org/10.18653/v1/2020.findings-emnlp.139

98. Hartigan, J.A., & Wong, M.A. A k-means clustering algorithm. 1979. DOI: https://doi.org/10.2307/2346830

99. Zhang, Y., Li, M., Wang, S., Dai, S., Luo, L., Zhu, E., Xu, H., Zhu, X., Yao, C., & Zhou, H. Gaussian Mixture Model Clustering with Incomplete Data. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM). No. 17. Pp. 1–14. 2021. DOI: https://doi.org/10.1145/3408318

100. Shahapure, K.R., & Nicholas, C.K. Cluster Quality Analysis Using Silhouette Score. 2020 IEEE 7th International Conference on Data Science and Advanced

Analytics (DSAA). Pp. 747–748. 2020. DOI: https://doi.org/10.1109/DSAA49011.2020.00096

101. Ros, F., Riad, R., & Guillaume, S. PDBI: A partitioning Davies-Bouldin index for clustering evaluation. Neurocomputing. No. 528. Pp. 178–199. 2023. DOI: https://doi.org/10.1016/j.neucom.2023.01.043

102. Lima, S.P., & Cruz, M.D. A genetic algorithm using Calinski-Harabasz index for automatic clustering problem. Revista Brasileira de Computação Aplicada. 2020. DOI: https://doi.org/10.5335/rbca.v12i3.11117

103. Li, X., Liang, W., Zhang, X., Qing, S., & Chang, P. A cluster validity evaluation method for dynamically determining the near-optimal number of clusters. Soft Computing. No. 24. Pp. 9227–9241. 2020. DOI: https://doi.org/10.1007/s00500-019-04449-7

104. Chlipala, A. An optimizing compiler for a purely functional web-application language. Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming. 2015. DOI: https://doi.org/10.1145/2784731.2784741

105. Deliyannis, E.P., Paul, N., Patel, P.U., & Papanikolaou, M. A comparative performance analysis of Chat GPT3.5, Chat GTP4.0 and Bard in answering common patient questions on melanoma. Clinical and experimental dermatology. 2023. DOI: https://doi.org/10.1093/ced%2Fllad409

Додаток А.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях, що входять до міжнародних наукометричних баз

1. Deineha, O., Donets, V., & Zholtkevych, G. (2024). The approach development of data extraction from lambda terms. Eastern-European Journal of Enterprise Technologies.

(*Особистий внесок здобувача: розробка основних ідей підходу екстракції даних з лямбда термів, проведення експериментів, написання частини тексту та його переклад.*

Особистий внесок Volodymyr Donets проведення експериментів та аналіз підходу екстракції даних з лямбда термів, написання частини тексту та його переклад.

Особистий внесок Grygoriy Zholtkevych розробка основних ідей підходу та концепції дослідження, аналіз отриманих результатів.)

2. Deineha, O. (2024). Supervised data extraction from transformer representation of Lambda-terms. *Radioelectronic and Computer Systems, 2024*(2), 19-29.

(*Особистий внесок: розробка ідей підходу екстракції даних з лямбда термів, дослідження існуючих моделей трансформерів, проведення експериментів, аналіз отриманих результатів, написання тексту та його переклад*)

Статті у наукових фахових виданнях України

3. Deineha O. The Clustering of Lambda Terms by Using Embeddings. Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». 2023. вип. 59. С.16-23.

(*Особистий внесок здобувача: аналіз підходів кластеризації лямбда термів за допомогою вбудовувань, проведення експериментів, дослідження типів вбудовувань, написання тексту та його переклад*).

4. Deineha, O. (2024). Lambda calculus term reduction: Evaluating LLMS' predictive capabilities. *Information Technology and Society*, 1(12), 51-55.

(*Особистий внесок: розробка та дослідження підходів оцінки прогнозивних здатностей LLM в розрізі роботи за лямбда-численням, аналіз існуючих моделей, написання тексту та його переклад*).

Наукові праці, які засвідчують апробацію матеріалів дисертації:

5. Deineha, O., Donets, V., Zholtkevych, G. (2023). On Randomization of Reduction Strategies for Typeless Lambda Calculus. In: Antoniou, G., et al. Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI 2023. Communications in Computer and Information Science, vol 1980. Springer, Cham. **(Scopus)**.

6. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Estimating Lambda-Term Reduction Complexity with Regression Methods. International Conference "Information Technology and Interactions". **(Scopus)**.

7. Deineha, O., Donets, V., & Zholtkevych, G. (2023). Deep Learning Models for Estimating Number of Lambda-Term Reduction Steps. International Workshop of IT-professionals on Artificial Intelligence. **(Scopus)**.

Онлайн сервіс створення та перевірки кваліфікованого та удосконаленого електронного підпису

ПРОТОКОЛ
створення та перевірки кваліфікованого та удосконаленого електронного підпису

Дата та час: 09:58:02 25.09.2024

Назва файлу з підписом: Deineha_diss.pdf.asice
Розмір файлу з підписом: 3.7 МБ

Перевірені файли:
Назва файлу без підпису: Deineha_diss.pdf
Розмір файлу без підпису: 5.8 МБ

Результат перевірки підпису: Підпис створено та перевірено успішно. Цілісність даних підтверджено

Підписувач: Дейнега Олександр Андрійович
П.І.Б.: Дейнега Олександр Андрійович
Країна: Україна
РНОКПП: 3570203554
Час підпису (підтверджено кваліфікованою позначкою часу для підпису від Надавача): 09:57:57 25.09.2024
Сертифікат виданий: "Дія". Кваліфікований надавач електронних довірчих послуг
Серійний номер: 382367105294AF9704000000FB962000F3A7E502
Тип носія особистого ключа: ЗНКІ криптомодуль ІІТ Гряда-301
Алгоритм підпису: ДСТУ 4145
Тип підпису: Кваліфікований
Тип контейнера: Підпис та дані в архіві (розширений) (ASiC-E)
Формат підпису: З повними даними ЦСК для перевірки (CAdES-X Long)
Сертифікат: Кваліфікований

Версія від: 2024.04.15 13:00