DOI: https://doi.org/10.36910/6775-2524-0560-2024-56-21
УДК 004.4`4:004.85
Іваненко Антон Романович, аспірант,
https://orcid.org/0000-0002-9846-537X
Марченко Олександр Іванович, к.т.н., доцент,
https://orcid.org/0000-0002-4537-3420
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ, Україна

МЕТОД КОМПІЛЯЦІЇ ОГОЛОШЕННЯ КЛАСІВ МОВ WEB-ПРОГРАМУВАННЯ У МОВИ БАЙТ-КОДОВОГО ТИПУ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Іваненко А.Р., Марченко О.І. Метод компіляції оголошення класів мов WEB-програмування у мови байт-кодового типу на основі машинного навчання. У даній статті пропонується метод, який дозволяє компілювати декларування класів мов WEB-програмування у мови байт-кодового типу на основі машинного навчання. Запропонований метод базується на ідеї використання двох видів штучного інтелекту. Для виконання перевірки введеної програми користувачем, запропоновано використати нейронну мережу, яка навчена для виконання задачі бінарної класифікації введеної програми на коректну і не коректну. Для навчання мережі було обрано алгоритм SDCA, що добре зарекомендував себе для розв'язку задач бінарної класифікації. З метою генерації інструкцій цільової мови на основі введеної програми у даній статті було використано генеративний штучний інтелект на основі LLM від компанії OpenAI. Для розв'язку задачі модель ChatGPT була донавчена на відповідних прикладах методом fine-tuning. Дослідження було апробоване на прикладі створення тестового компілятора, що перевіряє на коректність введену програму, що написана мовою TypeScript, та генерує відповідний код мовою CIL з достатньо високою точністю. Отриманий результат доводить, що використання методів машинного навчання для створення компіляторів є можливим. Такий підхід дозволить звести розробку компілятора тільки до підготовки правильного набору даних для донавчання відповідних моделей, що є значно простішим і менш часовитратним у порівнянні з класичним підходом, коли для кожного способу чи методу компіляції підмножини мови, що компілюється, необхідно вносити зміни і доопрацьовувати лексичний, синтаксичний, семантичний аналізатори та генератор коду для кожної нової чи зміненої конструкції вхідної мови програмування.
Ключові слова: компілятор, машинне навчання, генеративний штучний інтелект, бай-код, CIL, .NET, TypeScript, LLM, ChatGPT, ML.NET

Ivanenko A., Marchenko O. Method of compiling declarations of classes of WEB-programming languages into byte-code languages based on machine learning. This article proposes a method that allows you to compile class declarations of WEB programming languages into byte-code languages based on machine learning. The proposed method is based on the idea of using two types of artificial intelligence. In order to check the entered program by a user, it is proposed to use a neural network trained to perform the task of binary classification of the entered program into correct and incorrect. The SDCA algorithm was chosen for network training, which has proven itself well for solving binary classification problems. In order to generate CIL instructions based on the entered program, this article used generative artificial intelligence based on LLM from the OpenAI company. To solve the problem, the ChatGPT model was retrained on relevant examples using the fine-tuning method. The study was tested on a developed test compiler that checks the entered program written in TypeScript for correctness and generates the corresponding code in the Common Intermediate Language (CIL) with sufficiently high accuracy. The obtained result proves that the use of machine learning methods to create compilers is possible. This approach will reduce the development of the compiler only to the preparation of the correct data set for retraining the corresponding models, which is much simpler and less time-consuming compared to the classical approach, when for each method or method of compiling a subset of the language being compiled, it is necessary to make changes and refine the lexical, syntactic, semantic analyzers and a code generator for each new or changed construct of the input programming language.
Keywords: compiler, machine learning, generative artificial intelligence, byte-code CIL, .NET, TypeScript, ChatGPT, LLM, ML.NET

Постановка наукової проблеми.

Розробка компілятора з однієї мови програмування у іншу – досить складна задача, що складається з багатьох підзадач, зокрема і розробка компілятора з мов WEB-програмування у мови байт-кодового типу. У попередніх дослідженнях, [1] та інших, ми дослідили способи і методи компіляції певних конструкцій мови TypeScript у проміжну мову CIL використовуючи класичний підхід при розробці компілятора, що складається з лексичного, синтаксичного та семантичного аналізу, а також з написання генератора, який перетворює проміжне представлення програми у CIL інструкції.

Сьогодні все більшу популярність набирає використання різних засобів штучного інтелекту для розв'язання прикладних задач, в тому числі і генеративний штучний інтелект, який використовується для створення різних текстів, зображень, відео та музичних композицій за запитами користувача. Виходячи з цього, виникла ідея використання машинного навчання для створення компілятора мови WEB-програмування у мову байт-кодового типу. Такий підхід дозволив

би узагальнити і спростити вирішення проблеми розширення функціональності компілятора лише проводячи донавчання відповідної моделі штучного інтелекту.

Для апробації запропонованого методу і проведення експериментів у цьому напрямку було вирішено обрати невелику підмножину мови TypeScript, а саме оголошення класів, та розробити компілятор за запропонованим методом для цієї підмножини конструкцій мови у проміжну мову CIL платформи .NET. Зазначимо, що цей метод є придатним не тільки для мови Typescript та мови CIL, а й для компіляції оголошення класів будь-яких мов WEB-програмування у будь-які мови байт-кодового типу.

Аналіз останніх досліджень і публікацій.

На сьогодні є декілька різних досліджень та розробок, що пов'язані з використанням штучного інтелекту, а саме методів машинного навчання, для вирішення різних задач програмування. Розглянемо декілька найпопулярніших рішень використання машинного навчання для розв'язання задач генерації коду та трансляції коду з однієї мови в іншу.

У своїй статті англійські дослідники Юджіа Лі, Девід Чої та інші [2] дослідили, яким чином можливо навчити AlphaCode [3] вирішувати різні прикладні задачі шляхом генерації відповідного коду. З метою навчання штучного інтелекту було проаналізовано мільйони рішень різних задач. Автори статті використали підхід навчання, що складався з 3 етапів (рис. 1) [2]:

1. Pre-training (попереднє навчання) – на цьому етапі було проаналізовано сотні тисяч рішень на платформі Github на різних мовах програмування шляхом розділення кожного файлу на 2 частини, які використовувались як вхідні дані для енкодера для декодера відповідно, щоб навчити модель прогнозувати та генерувати продовження коду.

2. Fine-tuning (точне налаштування) – на цьому етапі на вхід енкодера був поданий опис задачі англійською мовою, а на вхід декодера – її рішення на відповідній мові програмування.

3. Sampling & Evaluation (семплінг та розрахунок) – на цьому етапі AlphaCode генерує багато зразків рішення для кожного опису проблеми, потім виконує їх, щоб відфільтрувати погані зразки та згрупувати ті, що залишилися. Після чого залишає невеликий набір кандидатів на правильне рішення заданої проблеми.

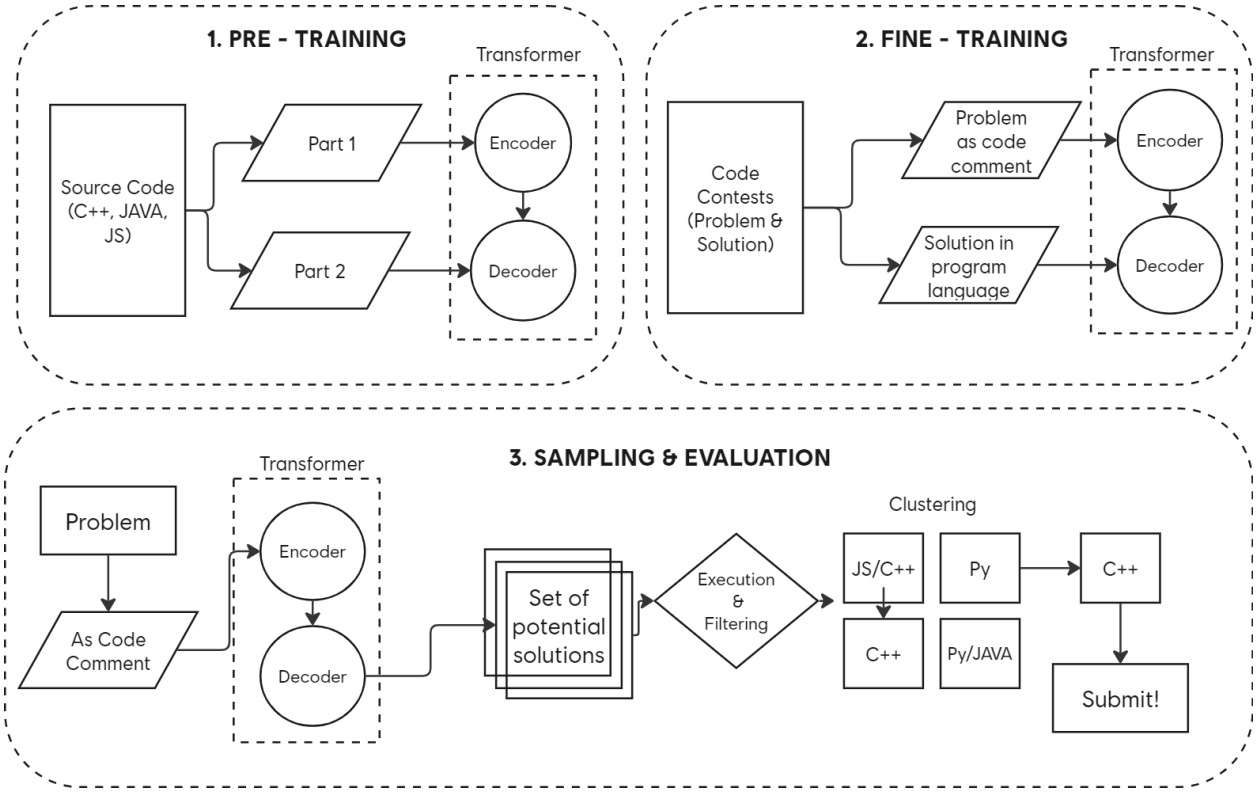

Рис. 1. Метод навчання AlphaCode для розв'язання задач програмування

Завдяки такому методу навчання, AlphaCode став спроможним розв'язувати задачі з

програмування краще, ніж 56% користувачів платформи Codeforces.

Розглянемо статтю [4], у якій китайські вчені Кьюші Сун, Нуо Чен та інші дослідили трансляцію програми з однієї мови програмування високого рівня в іншу. Вони назвали своє рішення TransCoder, що скорочено від Transferable Code Representation Learning. Цей метод базується на використанні Code Pre-Trained Models (далі CodePTM) та складається з двох етапів (рис. 2) [4]:

1. Source Task Training (навчання на вхідній задачі). На цьому етапі створюються універсальні префікси знань випадковим чином та додаються до CodePTM, що використовується для класифікації можливих розв'язків задачі певною мовою програмування.

2. Target Task Specification (специфікація цільової задачі). На цьому етапі універсальні префікси знання додаються до нової CodePTM, що відповідає розв'язку поставленої задачі іншими мовами програмування високого рівня.

Цей підхід дозволив реалізувати підтримку компіляції функцій написаних такими мовами програмування як Python, C++, JavaScript, Java. Наразі TransCoder є частиною загального рішення CodeGen від дослідницького центру компанії Meta.

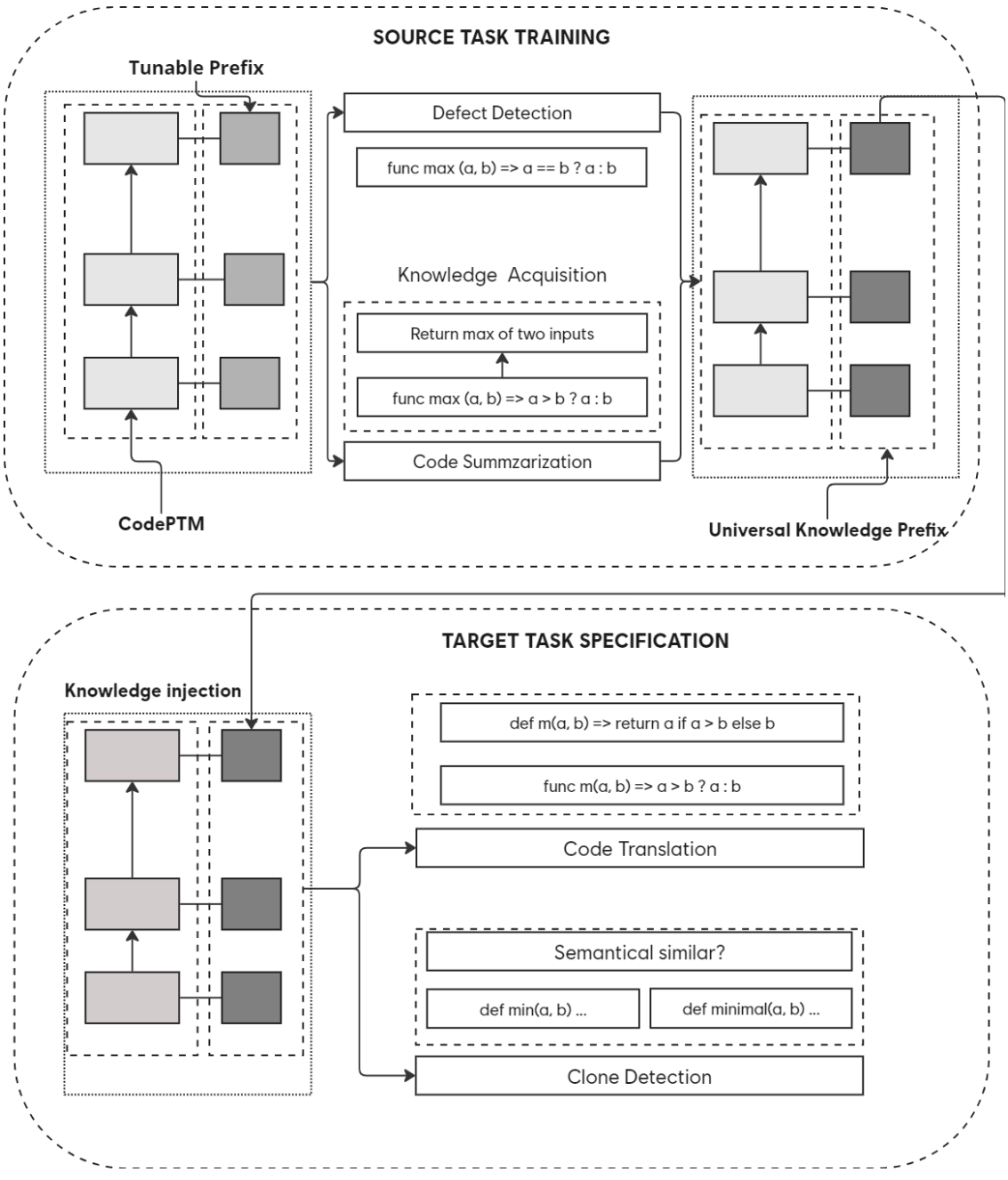

Рис. 2. Метод навчання TransCoder для компіляції програми з однієї мови високого рівня в іншу

В результаті проведеного аналізу існуючих досліджень, можна зробити два висновки.

Перший висновок – використання штучного інтелекту для вирішення різних задач програмування, починаючи від генерації коду для рішення задачі, описаної звичайною мовою, до трансляції коду, написаного однією з мов високого рівня в іншу, є можливим.

Другий висновок – дослідження компіляції мов програмування високого рівня у мови програмування низького рівня на основі машинного навчання наразі відсутні, і тому дослідження методів компіляції мов WEB-програмування у мови байт-кодового типу на основі машинного навчання матиме наукову новизну і практичну цінність.

Метою даної статті є розробка методу компіляції оголошення класів мов WEB-програмування у мови байт-кодового типу на основі машинного навчання.

Термінологія.

Large Language Model (LLM, Велика мовна модель) – це обчислювальна модель, що була навчена на великій кількості немаркованого тексту та здатна виконувати різні завдання обробки природної мови, наприклад розуміння і генерація текстів.

Fine-tuning (точне налаштування) – це підхід глибокого машинного навчання, за якого параметри попередньо навченої моделі додатково навчаються на нових даних.

Запропонований метод компіляції на основі машинного навчання.

Для апробації запропонованого методу, який є загальним і придатним для будь-яких мов WEB-програмування, було вирішено обрати мову TypeScript і виконати її компіляцію у проміжну мову CIL платформи .NET.

Розглянемо опис підграматики декларації класів мови TypeScript [5], яку було обрано для дослідження методу компіляції:

<class-declaration> --> class <class-name> <class-block>

<class-block> --> { <property-declaration-list>? }

<property-declaration-list> --> <class-property-declaration>|<class-property-declaration> <class-property-declaration-list>

<property-declaration> --> <access-modifier-keyword><property-name>:<attribute>;

<access-modifier-keyword> -> private | protected | public

<attribute> -> number | boolean | string | any

<class-name> -> <identifier>

<property-name> -> <identifier>

Приклад оголошення класу, що містить всі зазначені правила обраної підграматики, зображено на рис. 3, а очікуваний результат компіляції у проміжну мову CIL платформи .NET – на рис. 4.

```
class Test {
        private privateNumber: number;
        protected protectedString: string;
        public publicBoolean: boolean;
        public publicAny: any;
}
```

Рис. 3. Приклад оголошення класу мовою TypeScript

```
.class public auto ansi beforefieldinit Test

    extends [System.Private.CoreLib]System.Object
{
    .field private int32 privateNumber
    .field family string protectedString
    .field public bool publicBoolean
    .field public object publicAny

    .method public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        .maxstack 8

        IL_0000: ldarg.0
        IL_0001: call instance void [System.Private.CoreLib]System.Object::.ctor()
        IL_0006: nop
        IL_0007: ret
    }
}
```

Рис. 4. Очікувана програма мовою CIL, що містить приклад оголошення класу з рис. 3

У мові CIL ключове слово .class відповідає класу. Ключове слово beforefieldinit означає, що тип можна ініціалізувати в будь-який момент до того, як будуть посилатися на будь-які його статичні поля (компіляція статичних класів у даній статті не розглядається). За оголошення поля відповідає ключове слово .field. Відповідність області видимості полів мови TypeScript до мови CIL наступна:

- private – private
- protected – family
- public – public.

А відповідність типів даних є такою:

- number – int32
- string – string
- boolean – bool
- any – object.

У проміжній мові CIL кожен тип даних повинен успадковуватись від загального типу-посилання System.Object (якщо дані типу зберігаються у кучі) чи типу-значення System.Value (якщо дані зберігаються на стеку) та мати конструктор за замовчуванням без параметрів. У мові TypeScript клас є типом-посиланням, тому реалізуємо успадкування від типу System.Object.

Як відомо, основний процес компіляції виконується у 4 етапи:

1) лексичний аналіз;
2) синтаксичний аналіз;
3) семантичний аналіз;
4) генерація коду.

У підмножині мови TypeScript, що розглядається в експерименті, відсутні правила, які потрібно перевіряти на семантичну некоректність, і таким чином компіляцію можна звести то задачі лексичного і синтаксичного аналізу та задачі генерації коду. Реалізацію перших двох етапів можна виконати одним об'єднаним етапом перевірки на коректність введеної програми. Таким чином, будемо розглядати дві підзадачі машинного навчання – перевірку введеної програми на помилки та генерацію вихідної програми мово CIL. Перша підзадача є задачею класифікації, де головною метою є класифікація введеного тексту на коректний і некоректний. Друга підзадача є задачею генерації тексту, для чого можна використати генеративний штучний інтелект на основі LLM (наприклад ChatGPT), який донавчено виконувати таку підзадачу. Схема компілятора, що пропонується для рішення цих підзадач, зображена на рис. 3.

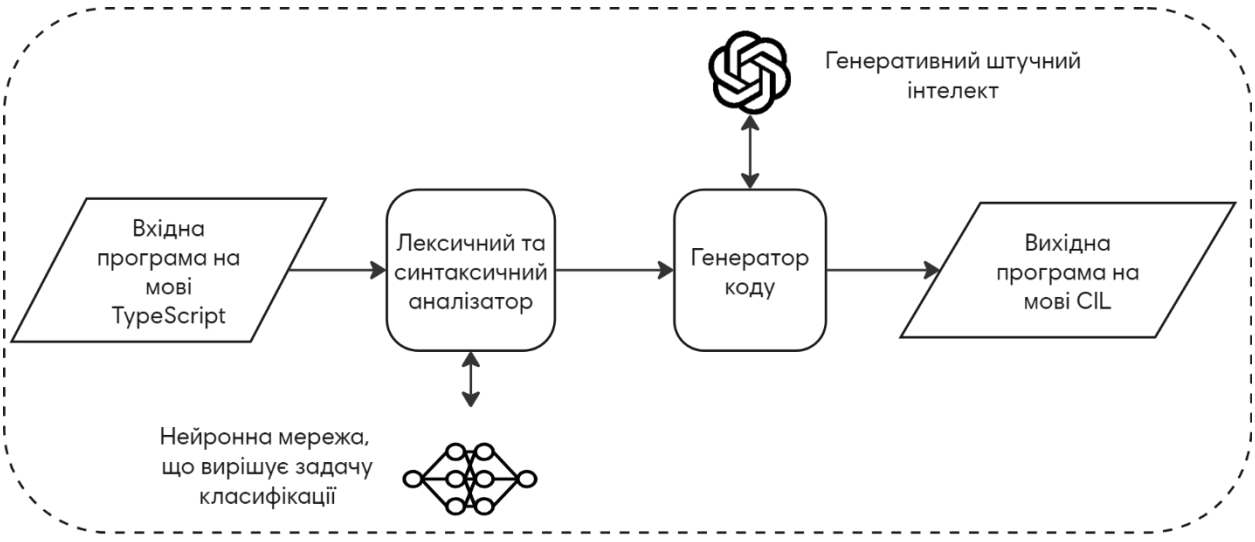

Рис. 3. Схема запропонованого компілятора

Для написання модуля, що відповідає за аналіз програми шляхом класифікації, використаємо платформу ML.NET [6].

Для початку напишемо консольну програму, що буде створювати та тренувати нашу модель на тестових даних. Ця програма має виконувати наступні етапи:

1) завантаження тестових даних з файлу та їх розподіл на дані для навчання та дані для тестування;
2) побудова і навчання моделі на відповідних даних з пункту 1;
3) тестування точності моделі на тестових даних;
4) збереження моделі в локальний файл.

Для навчання моделі було вирішено обрати алгоритм Stochastic Dual Coordinate Ascent (далі SDCA). Цей алгоритм навчання добре зарекомендував себе для вирішення задач бінарної класифікації, що висвітлено у відповідних дослідженнях [7, 8]. Схема запропонованого методу навчання зображена на рис. 4.

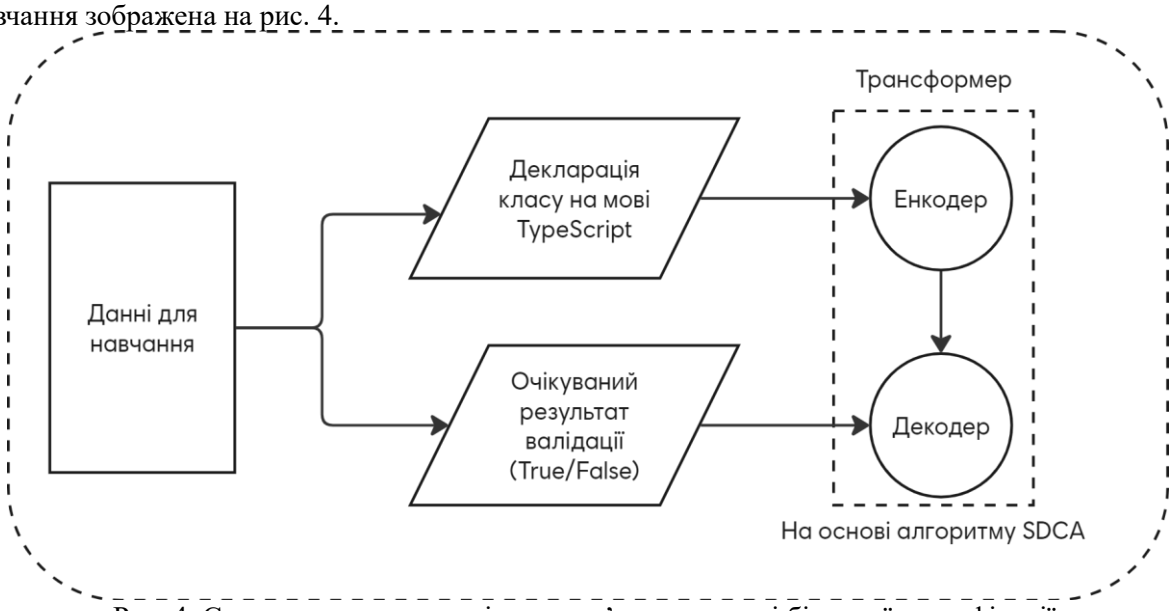

Рис. 4. Схема навчання моделі для розв'язання задачі бінарної класифікації.

Після того, як модель для виконання задачі класифікації коректного тексту налаштована і збережена у файл, у самій реалізації компілятора достатньо лише завантажити її та виконувати перевірку вхідної програми.

Наступним етапом після аналізу на коректність вхідної програми є генерація коду мовою CIL. Для виконання задачі використаємо генеративний штучний інтелект на основі LLM для генерації текстів ChatGPT від компанії Open AI. Оскільки відповідна модель ніколи не навчалась виконувати поставлену нами задачу, необхідно виконати налаштування шляхом донавчання цієї моделі за методом fine-tuning. Open AI надає відповідне API [9] для налаштування власної моделі, використовуючи підготовлені тестові дані й відповідні запити (promts). За основу власної моделі візьмемо найновішу версію gpt-4o-mini.

Створимо відповідну консольну програму, що буде завантажувати підготовлені тестові дані, та надсилати їх за протоколом HTTP для налаштування власної моделі. Ця програма виконуватиме наступні кроки:

1) завантаження даних у форматі json, що містить інформацію, як має поводити себе модель, приклад запиту та очікуваний результат на цей запит (рис. 7);
2) надсилання тестових даних у вигляді спеціальних запитів (promts);
3) перевірку результату, отриманого на тестових даних.

```
{
  "messages": [
      {
          "role": "system",
          "content": "Compiler is a chatbot that returns only generated code."
      },
      {
          "role": "user",
          "content": "Compile typescript code to common intermediate language: class
Test { private privateNumber: number…"
      },
```

```
        {
            "role": "assistant",
            "content": ".class public auto ansi beforefieldinit Test...."
        }
    ]
}
```

Рис. 7. Приклад контенту json файлу, що містить дані для навчання за методом fine-tuning

Після того, як модель налаштована і протестована, її можливо інтегрувати у наш тестовий компілятор. Для цього достатньо створити клієнт чат-бота, використовуючи той самий арі ключ, що було використано для донавчання власної моделі і зробити запит до чат-бота відповідним запитом (promt), формат якого було використано на етапі навчання (рис. 7, повідомлення role user).

Аналіз отриманих результатів.

Оскільки для виконання компіляції шляхом використання машинного навчання була взята досить мала підграматика мови TypeScript, то аналіз отриманих результатів слід проводити не у порівнянні з існуючими аналогами (як було проаналізовано у попередніх дослідженнях авторів), а дослідити точність роботи запропонованого методу на тестових даних.

За метрику точності роботи алгоритму було обрано відношення кількості коректних результатів роботи тестового компілятора до загальної кількості тестових даних (у %), в залежності від загальної кількості даних, що були використані для навчання і налаштування моделей.

Для моделі, що виконує задачу класифікації, діапазон даних для навчання був обраний від 100 до 1000000, а для моделі, що відповідає за генерацію CIL інструкцій, – від 100 до 10000. Така розбіжність діапазонів пояснюється далі в аналізі.

Для тестування було згенеровано 1000 декларацій класів з параметрами, які відрізняються за такими параметрами: кількість полів, різноманітність типів полів, імен полів та імен класів. Половина з цих даних містила помилки, які повинні були бути відсіяні на етапі лексичного і синтаксичного аналізу. Перевірка коректності згенерованих CIL інструкцій була виконана компілятором ILasm[10], який компілює проміжну мову CIL у мову асемблера. Результати тестування продемонстровані на рис. 5 та рис. 6.

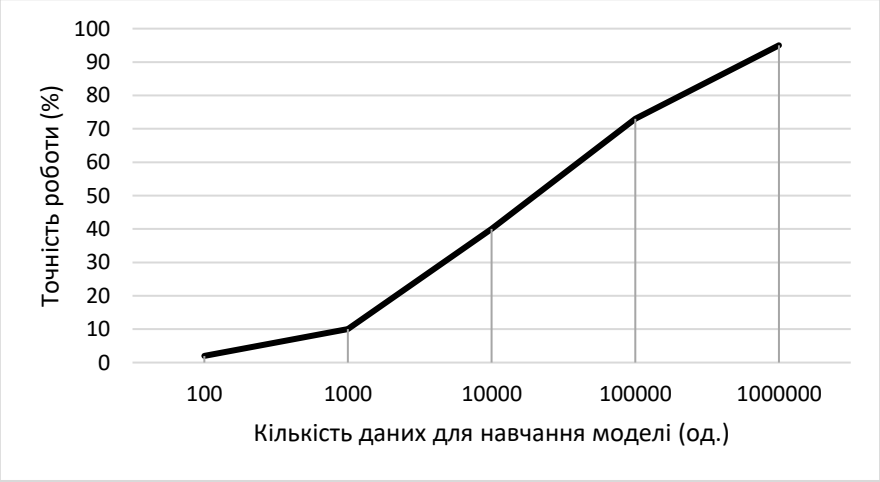

Рис. 5. Графік залежності коректності роботи лексичного та синтаксичного аналізу від загальної кількості даних для навчання моделі

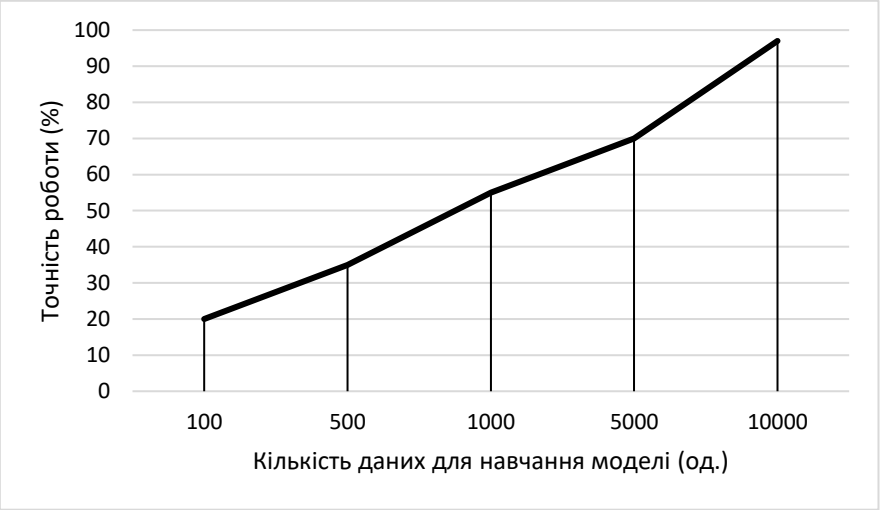

Рис. 6. Графік залежності коректності згенерованого коду від загальної кількості даних для навчання моделі

Як бачимо з отриманих результатів, для підвищення точності роботи компілятора необхідно збільшувати кількість даних для навчання. Чим ближче точність наближається до 100%, тим в рази більше даних необхідно використати для навчання.

Варто зазначити, що у випадку навчання моделі для виконання задачі класифікації (рис. 8) необхідно в рази більше тестових даних у порівнянні з донавчанням генеративної мережі (рис.9). Така розбіжність тестових даних пояснюється тим, що генеративна мережа (у нашому випадку ChatGPT) вже навчена виконувати задачу генерації тексту на великій кількості різноманітних даних і в ході експерименту ми лише донавчали її на різних прикладах, щоб відкоригувати роботу для виконання поставленої нами задачі.

Висновки та перспективи подальшого дослідження.

В цій статті отримало подальший розвиток дослідження методів компіляції мов WEB-програмування у мов байт-кодового типу. Зокрема було розроблено і досліджено метод компіляції оголошення класів на основі машинного навчання, який був апробований на прикладі компіляції оголошення класів мови TypeScript у проміжну мову CIL платформи .NET, яка є мовою байт-кодового типу.

Для реалізації тестового компілятора етапи лексичного і синтаксичного аналізу процесу компіляції були об'єднані в одну задачу машинного навчання, яка виконувала аналіз програми на коректність, а генерація вихідного коду вирішувалася як друга задача машинного навчання. Для розв'язку першої задачі було вирішено використати нейронну мережу, що була навчена на основі алгоритму SDCA виконувати бінарну класифікацію тексту введеної користувачем програми на коректну і некоректну, а для розв'язку другої задачі – використати LLM модель (ChatGPT), яку було донавчено за методом fine-tuning на відповідних прикладах генерувати CIL інструкції.

Отримані результати показують, що запропонований метод може бути застосований на практиці, але для реалізації компілятора навіть достатньо малої підмножини мови WEB-програмування необхідно мати велику кількість тестових даних для навчання. Крім того, у порівнянні з класичними підходами трансляції запропонований метод не завжди показує достатню точність роботи. Для підвищення точності роботи необхідно збільшувати кількість тестових даних, а для реалізації компілятора усієї мови кількість таких тестових даних повинна бути збільшена в рази.

Отриманий результат доводить, що використання методів машинного навчання для створення компіляторів є можливим. Такий підхід дозволить звести розробку компілятора тільки до підготовки правильного набору даних для донавчання відповідних моделей, що є значно простішим і менш часовитратним у порівнянні з класичним підходом, коли для кожного способу чи методу компіляції підмножини мови, що компілюється, необхідно вносити зміни і доопрацьовувати лексичний, синтаксичний, семантичний аналізатори та генератор коду для кожної нової чи зміненої конструкції вхідної мови програмування.

Напрямком подальших досліджень може бути розвиток запропонованого методу для інших конструкцій мов WEB-програмування, його апробація на інших вхідних мовах та мовах байт-

кодового типу, а також узагальнення цього методу компіляції на основі інших видів машинного навчання.

Список бібліографічного опису

1. Іваненко А.Р., Марченко О.І. «Метод компіляції типів об'єднання мови TypeScript у проміжну мову CIL платформи .NET», *Комп'ютерно-інтегровані технології: освіта, наука, виробництво,* 2023, № 52, с.77-84. Режим доступу: http://surl.li/mxrufv. Дата звернення: 11.09.24.
2. Yujia Li, David Choi, , Junyoung Chung and others. «Competition-level code generation with AlphaCode», *Science*, 2022, Vol 378, pp. 1092-1097. Available: http://surl.li/jbchsl. Accessed on: 11.09.24.
3. AlphaCode Attention Visualization, Available: https://alphacode.deepmind.com. Accessed on 11.09.24
4. Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, Ming Gao. «TransCoder: Towards Unified Transferable Code Representation Learning Inspired by Human Skills», *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, 2024, pp. 16713–16726, Available: http://surl.li/sxmmcw. Accessed on: 11.09.24.
5. Документація TypeScript. Режим доступу: https://www.typescriptlang.org/docs/handbook/unions-and-intersections.html. Дата звернення 11.09.24.
6. Документація ML.NET. Режим доступу: https://learn.microsoft.com/en-us/dotnet/machine-learning. Дата звернення: 11.09.24.
7. Shai Shalev-Shwartz, Tong Zhang. «Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization», *Journal of Machine Learning Research*, 2013, №14, pp. 567-599. Available: http://surl.li/vcaisp. Accessed on: 11.09.24.
8. Kenneth Tran, Saghar Hosseini, Lin Xiao, Thomas Finley, Mikhail Bilenko. «Scaling Up Stochastic Dual Coordinate Ascent», *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1185 – 1194. Available: http://surl.li/cdxvze. Accessed on: 11.09.24.
9. Документація OpenAI Platform. Режим доступу: http://surl.li/rnmkpo. Дата звернення: 11.09.24.
10. Документація ILAsm. Режим доступу: http://surl.li/dtqbcv. Дата звернення: 11.09.24.

References

1. Ivanenko A.R., Marchenko O.I. «Method of compilation union types of TypeScript into Common Intermediate Language of .NET platform», *Computer-Integrated Technologies: Education, Science, Production*, 2023, №52, pp. 77-84. Available: http://surl.li/mxrufv. Accessed on: 11.09.24.
2. Yujia Li, David Choi, , Junyoung Chung and others. «Competition-level code generation with AlphaCode», *Science*, 2022, Vol 378, pp. 1092-1097. Available: http://surl.li/jbchsl. Accessed on: 11.09.24.
3. AlphaCode Attention Visualization, Available: https://alphacode.deepmind.com/. Accessed on 11.09.24.
4. Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, Ming Gao. «TransCoder: Towards Unified Transferable Code Representation Learning Inspired by Human Skills», *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, 2024, pp. 16713–16726, Available: http://surl.li/sxmmcw. Accessed on: 11.09.24.
5. TypeScript documentation. Available: https://www.typescriptlang.org/docs/handbook/unions-and-intersections.html. Accessed on: 11.09.24.
6. ML.NET documentation. Available: https://learn.microsoft.com/en-us/dotnet/machine-learning/. Accessed on: 11.09.24.
7. Shai Shalev-Shwartz, Tong Zhang. «Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization», *Journal of Machine Learning Research*, 2013, №14, pp. 567-599. Available: http://surl.li/vcaisp. Accessed on: 11.09.24.
8. Kenneth Tran, Saghar Hosseini, Lin Xiao, Thomas Finley, Mikhail Bilenko. «Scaling Up Stochastic Dual Coordinate Ascent», *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1185 – 1194. Available: http://surl.li/cdxvze. Accessed on: 11.09.24.
9. OpenAI Platform documentation. Available: http://surl.li/rnmkpo. Accessed on: 11.09.24.
10. ILAsm documentation. Available: http://surl.li/dtqbcv. Accessed on: 11.09.24