УДК 004.91 http://doi.org/10.15407/pp2025.01.024

Б.О. Семьонов, С.Д. Погорілий

ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ GPGPU ТА TPU ДЛЯ МЕТОДУ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ КОМЕНТАРІВ У СИСТЕМАХ КОНТРОЛЮ ВЕРСІЙ

У роботі обґрунтовано актуальність вирішення задачі забезпечення якості описів до внесених змін у вихідних текстах програм для систем контролю версій. Для здійснення фільтрації коментарів використовуються методи машинного навчання: нейронні мережі різної архітектури. Доцільним є використання нейронних мереж у зв'язку з необхідністю пошуку описів до внесених змін, які відображають їхню мету. Створено рекурентні нейронні мережі та здійснено їх навчання на множині описів до внесених змін, отриманих за допомогою спеціального програмного інтерфейсу GitHub REST API. Для покращення швидкодії навчання застосовано різні програмно-апаратні платформи на кшталт CPU, TPU та GPGPU. Проведено аналіз точності моделей за допомогою метрик: точності (Accuracy) та середнього гармонійного (F1-score).

Ключові слова: GPGPU, RNN, TPU, опис внесених змін, репозиторій, система контролю версій.

B.O. Semonov, S.D. Pogorilyy

RESEARCH OF THE APPLICATION OF GPGPU AND TPU TECHNOLOGIES FOR ENSURING COMMENT QUALITY IN VERSION CONTROL SYSTEMS

The study substantiates the relevance of solving the issue of ensuring the quality of descriptions for changes made in source code files within version control systems. Machine learning methods, particularly neural networks of various architectures, are employed for comment filtering. Neural networks are deemed appropriate due to the necessity of identifying descriptions that accurately reflect the purpose of the changes made. Recurrent neural networks were developed and trained on a dataset of change descriptions obtained through the GitHub REST API. To enhance training performance, various hardware and software platforms such as CPU, TPU, and GPGPU were utilized. The accuracy of the models was analyzed using metrics like Accuracy and the harmonic mean (F1-score).

Keywords: commit message, GPGPU, repository, RNN, TPU, version control system.

Вступ

У сучасному цифровому світі, де розвиток програмного забезпечення стає невід'ємною частиною багатьох галузей, системи контролю версій є ключовим інструментом для ефективної розробки та управління вихідними текстами програм. Постійні зміни ринку вимагають від розробників не лише швидкості та якості, а й систематизованого та надійного підходу до керування версіями застосунків [7].

Завдяки системам контролю версій розробники можуть ефективно працювати над проєктами, вносити зміни та експериментувати з новими функціями, водночас маючи можливість у разі необхідності повертатися до попередніх версій. Це дозволяє уникнути втрат даних, забезпечує стабільність роботи системи та сприяє кращій співпраці в команді розробників.

Опис внесених змін у системі контролю версій є важливим елементом для розуміння, відстеження та збереження історії змін у кодовій базі. Кожне повідомлення про внесені зміни є відображенням конкретної зміни, внесеної розробником. Деталізований та розгорнутий опис внесених змін допомагає не лише поточним, а й майбутнім членам команди легко розібратися в тому, як і чому змінено вихідний текст програми [3].

Зрештою деталізовані описи внесених змін роблять проєкт більш документованим та полегшують процеси технічного супроводу, допомагають відслідковувати

етапи його розвитку, логіку ухилення рішень та, кінець кінцем, роблять проєкт прозорішим.

Створення методу забезпечення якості коментарів до внесених змін у вихідних текстах програм для систем контролю версій є актуальним у зв'язку із зростанням обсягу проєктів та ускладненням структури кодової бази.

Враховуючи те, що до сучасних проєктів можуть залучатися розробники із різним досвідом та стилями програмування, фільтр описів внесених змін, який оцінюватиме зміст та контекст опису, буде сприяти розумінню іншими членами команди внесених у вихідний текст програм змін та буде спрощувати технічну підтримку.

Отже, опис внесених змін – це повідомлення, зміст якого відповідає загальним правилам написання повідомлень про внесені зміни, є лаконічним, описує характер змін, їхні ефект та причину.

1. Правила щодо оформлення повідомлення про внесені зміни [6]:
   а. опис повинен мати заголовок і може мати тіло. Ці частини мають бути розділені порожнім рядком;
   б. заголовок не має бути довгим, не більше 50-70 символів;
   в. дієслова у заголовку повинні бути доконаного виду.
2. Правила щодо подання інформації в повідомленні про внесені зміни:
   а. має бути описана причина зроблених змін;
   б. який ефект має це повідомлення про внесені зміни;
   в. якщо внесені зміни виправляють якусь проблему, то необхідно її вказати;
   г. подання інформації має бути таким, щоб фахівець, який не розуміється на проблемі чи структурі коду, збагнув, що було зроблено і який вплив це має на проєкт (програму) Приклади описів наведено в Таблиці 1.

Постановка задачі

Всі, наведені у попередньому розділі правила написання повідомлень про внесені зміни, можна опрацювати засобами будь-якої мови програмування. Проте для правила 2.г потрібно використовувати саме машинне навчання, бо формальних правил для визначення, що було зроблено і чому, не існує – для цього потрібне «людське» розуміння контексту та змісту повідомлення про внесені зміни. Звідси випливає постановка задачі: створити метод забезпечення якості повідомлень про внесені зміни вихідних текстів програм у системах контролю версій, який аналізуватиме опис внесених розробником змін у вихідний текст програми та повертатиме мітку, чи відповідає цей коментар до внесених змін на питання «що було зроблено і чому?», тобто «так» чи «ні».

Таблиця 1

Повідомлення про внесені зміни, які відповідають правилу 2.г

№	Приклад опису
1	Add array identifiers for generating routes. This allows for resources with multi-column keys to generate the related routes by only passing an instance of the object in question.
2	Increases the select timeout in start reactor() endless loop. This small patch greatly reduces CPU time (for instance for backgroundrb).
3	Add a ’variance’ method and a ’sum’ method to the Array class to stay DRY.
4	Fixed autocomplete with scrollbar. IE has problems when the ul has a scrollbar and the user clicks there. The activate event is thrown and the list disappears.

Формалізація задачі фільтра. Нехай **X** – множина описів внесених змін (генеральна сукупність), кожне із цих повідомлень про внесені зміни описується m-вимірним вектором слів:

$$\vec{x} = \{x_1, x_2, \ldots, x_m\}, \vec{x} \in \mathbf{X}, \quad (1)$$

де m – розмірність вектору слів.

Y – множина можливих відгуків (міток) у вигляді M-вимірного вектору:

$$\vec{y} = \{y_1, \ldots, y_M\}, \vec{y} \in \mathbf{Y}, \quad (2)$$

де M = 2 – це кількість відгуків (класів), які потрібно отримати: перший відгук – чи задовольняє поточне повідомлення вимоги, другий – не задовольняє. Звідси випливає, що шуканий фільтр – це задача бінарної класифікації: Y = {0, 1}.

Отже, модель фільтра повідомлень до внесених змін можна описати таким сюр'єктивним, але не ін'єктивним, відображенням:

$$f : \mathbf{X} \to \mathbf{Y} \quad (3)$$

Збір та підготовка навчальних корпусів повідомлень

Для збору даних для навчання було використано один із найбільших веб-сервісів для розміщення IT-проєктів та спільної розробки програмного забезпечення GitHub, в основі якого лежить популярна система контролю версій Git. Цей веб-сервіс має спеціальний програмний інтерфейс для застосунків, який називається GitHub REST API [4]. Через те, що дослідження пов’язане з перевіркою змісту повідомлень (commit messages) про внесені зміни (differences, скорочено «diffs») для систем контролю версій, знадобилися такі інтерфейси:

1. отримання переліку репозиторіїв (проєктів);
2. отримання повідомлень до внесених змін для кожного репозиторію.

Наведені вище інтерфейси мають тип GET http-запиту і повинні формувати та передавати на сервер відповідні GET-параметри та http-заголовки. У свою чергу відповідь від сервісів надходить у текстовому форматі структурованих даних JSON.

Вказані у Таблиці 2 http-запити мають однакові набори http-заголовків, а саме:

1. 'Accept': 'application/vnd.github+json' – тип відповіді від програмного інтерфейсу GitHub REST API;
2. 'Authorization': 'Bearer [token]' – авторизація користувача API,

   де token – спеціальний набір символів, який можна згенерувати в налаштуваннях користувача вебсервісу GitHub;
3. 'X-GitHub-Api-Version': '2022-11-28' – версія програмного інтерфейсу GitHub REST API.

Після завантаження було виконано ручне маркування цих даних, тобто віднесення кожного повідомлення до відповідного класу згідно з правилом 2.г. Такий процес довготривалий, тому в цій роботі використовується 8 тис. повідомлень з 1 млн. завантажених.

Аналіз промаркованих описів внесених змін показав, що дані є нерівномірними. В отриманій вибірці описи, які не відповідають правилу 2.г, кількісно переважають повідомлення, які відповідають вимогам. На Рис. 1 зображено розподіл навчальних корпусів повідомлень про внесені зміни. Такий розподіл впливає на точність навчання нейронних моделей, тому наступний методи було використано задля боротьби з вказаною проблемою: зсув порогу (threshold moving) з вибором коефіцієнту 0.57.

Перед подачею повідомлень про внесені зміни на вхід досліджуваних нейронних мереж було виконано [11]:

1. нормалізацію (стандартизацію) повідомлень, а саме: приведення літер тексту у малі;
2. токенізацію повідомлень на навчальній вибірці;
3. векторизацію повідомлень. Під час створення об'єкту векторизації повідомлень було зафіксовано довжину вихідної послідовності токенів, тому що форма вхідного для нейронної мережі тензору має бути фіксованою. А також обмежено розмір словника, чим нехтують найменш вживані токени для прискорення обчислень.

Таблиця 2

Особливості сервісів програмного інтерфейсу GitHub REST API

№	Назва сервісу	URL-посилання	GET-параметри	«Корисні» поля для дослідження
1	Отримання переліку репозиторіїв	https://api.github.com/repositories	**since** – параметр, значення якого повинне бути цілим числом та відповідає ідентифікатору репозиторія. Він не є обов'язковим. Якщо цей параметр присутній, то у відповіді буде повернуто перелік репозиторіїв, ідентифікатори яких більші за вказаний у параметрі.	**id** – ідентифікатор репозиторію; **full_name** – назва репозиторію; **description** – детальний опис репозиторію.
2	Отримання повідомлень до внесених змін для кожного репозиторію	https://api.github.com/repos/[full_name]/commits?per_page=100, де full_name – назва репозиторію з пункту № 1	**per_page** – кількість повідомлень про внесені зміни у відповіді на один запит. Необов'язковий параметр. За замовчуванням встановлено 30 повідомлень. **sha** – символьна послідовність (відбиток) повідомлення про внесені зміни, з якого потрібно починати пошук.	**sha** – символьна послідовність (відбиток) повідомлення про внесені зміни; **message** – повідомлення про внесені зміни. Знаходиться в JSON-об'єкті «commit»; **date** – дата та час, коли були внесені зміни до репозиторію. Знаходиться в JSON-об'єкті «author», який, у свою чергу, розміщений у JSON-об'єкті «commit». **sha** з JSON-масиву **parents** – символьна послідовність (відбиток) повідомлення про внесені зміни попередніх внесених змін.

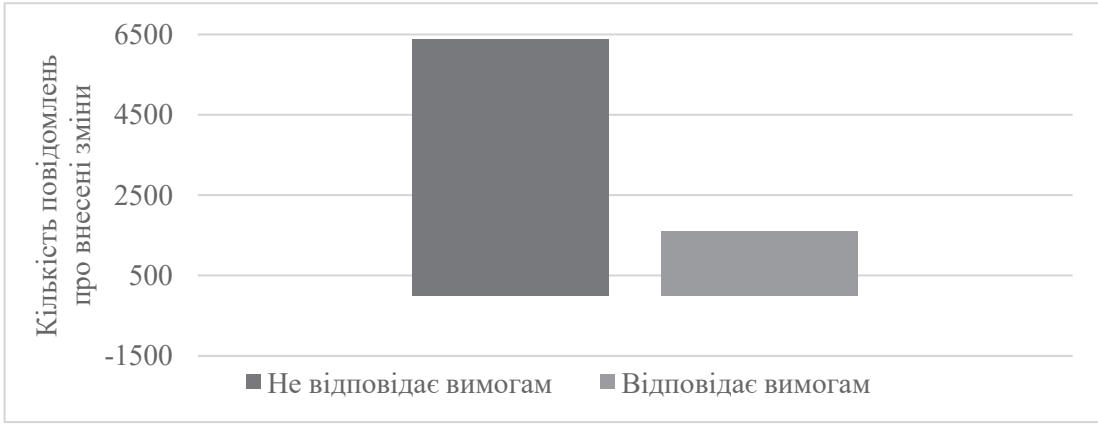

Рис. 1. Розподіл повідомлень про внесені зміни

Метод забезпечення якості коментарів до внесених змін у системах контролю версій

Перший фільтр (Рис. 2) було побудовано на основі моделі з використанням повнозв'язного (Dense) шару, який складається з:

1. двох вагових шарів (weighted layers):
   а. вхідного (Embedding) шару;
   б. вихідного повнозв'язного шару (Dense) [14] з кількістю нейронів, яка дорівнює кількості бажаних відгуків (M = 2);
2. проміжного шару без ваг (Global Average Pooling) [14].

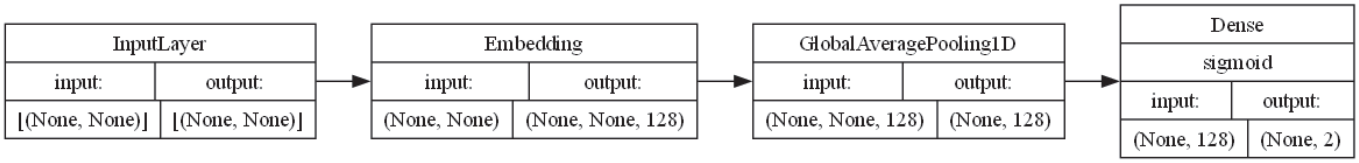

Рис. 2. Архітектура моделі з повнозв'язним шаром без методів регуляризації

Як алгоритм оптимізації ваг обрано AdamW [9], що так само є алгоритмом Adam зі спеціальним варіантом регуляризації на основі пониження ваг (weight decay). Обраний алгоритм демонструє швидше навчання і краще узагальнення, основними кроками якого є:

1. ініціалізація параметрів:
   а. встановлення початкових значень для параметрів моделі θ_0 (вектор ваг) випадковим чином;
   б. ініціалізація першого моменту (оцінка середнього градієнта) $m_0 = 0$ та другого моменту (оцінка дисперсії градієнта) $v_0 = 0$;
   в. встановлення гіперпараметрів:
   η – швидкість навчання (learning rate);
   β_1 і β_2 – коефіцієнти згасання для першого та другого моментів;
   ε – деяке мале значення для запобігання діленню на нуль;
   λ – коефіцієнт вагового спаду (weight decay);
2. обчислення градієнта: для кожної ітерації t обчислюється градієнт функції втрат:
$$g_t = \nabla_\theta f(\theta_t), \quad (4)$$
де $f(\theta)$ – це функція втрат,
g_t – градієнт для поточних параметрів;
3. оновлення моментів:
   а. оновлення першого моменту (оцінка середнього градієнта):
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad (5)$$
   б. оновлення другого моменту (оцінка дисперсії градієнта):
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad (6)$$
4. коригування зміщення моментів: для того, щоб компенсувати зміщення на початкових етапах (оскільки $m_0 = 0$ та $v_0 = 0$), відбувається коригування:
$$m_t = \frac{m_t}{1-\beta_1^t}, \quad (7)$$
$$v_t = \frac{v_t}{1-\beta_2^t} \quad (8)$$
5. оновлення параметрів з урахуванням вагового спаду та адаптивної швидкості навчання:
$$\theta_t = \theta_{t-1} - \eta\left(\frac{m_t}{\sqrt{v_t + \varepsilon}} + \lambda\theta_{t-1}\right), \quad (9)$$
де ваговий спад $\lambda\theta_{t-1}$ застосовується окремо від основного процесу оновлення градієнта, що робить AdamW відмінним від класичного Adam;
6. повторення кроків 2 – 5 для кожної ітерації t, поки не буде досягнуто зупинки (залежно від кількості епох або критеріїв зупинки).

У моделі другого фільтру описів внесених змін (Рис. 3) для зменшення прояву явища перенавчання використано методи регуляризації:

1. введено додатковий шар Dropout з коефіцієнтом 0.3 – випадкове виключення нейронів шару із заданою

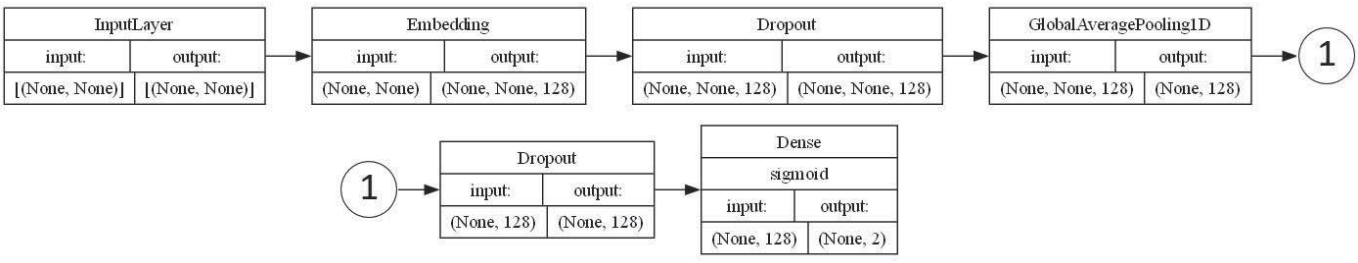

Рис. 3. Архітектура моделі з повнозв'язним шаром та методами регуляризації

ймовірністю p під час кожної ітерації навчання, метою якого є перешкоджання спільній адаптації ваг нейронів шару, яка у свою чергу призводить до перенавчання [10];
2. використано метод ранньої зупинки.

Окрім того, до частини, яка видобуває ознаки тексту для подальшої класифікації, було додано різні варіанти рекурентних шарів, в результаті чого вдалося побудувати 5 різних моделей:

1. один LSTM-шар з 48-ма нейронами (Рис. 4);
2. один двонаправлений (bidirectional) LSTM/GRU-шар з 24-ма нейронами (загалом 48 нейронів) (Рис. 5);
3. два двонаправлені (bidirectional) LSTM/GRU-шари з 12-ма нейронами кожний шар (загалом 48 нейронів) (Рис. 6).

Слід зауважити, що для навчання моделей було використано методологію, яка базується на основі перевірної підмножини. Вона полягає у тому, що весь доступний набір маркованих даних поділяється на частини, що не перетинаються, а саме: з навчальної множини (training set) було виділено 20% на перевірну підмножину (validation subset), а решту – для навчання моделей-кандидатів.

Для імплементації наведених вище моделей було використано мову програмування Python (версії 3.10.12) та застосовано бібліотеку TensorFlow (2.15.0), тому що вона має підтримку GPGPU та TPU технологій, які є однією зі складових дослідження. У Таблиці 3 наведено порівняльну характеристику між TensorFlow та іншими популярними бібліотеками для машинного навчання, такими як PyTorch [12], Keras [8] та Scikit-learn [13]. Кожна з цих бібліотек має свої унікальні можливості та підходить для різних типів завдань.

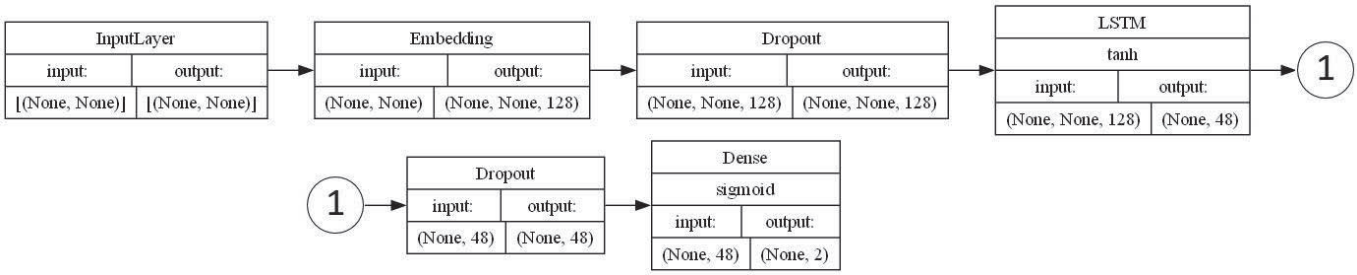

Рис. 4. Архітектура моделі з рекурентним LSTM-шаром та методами регуляризації

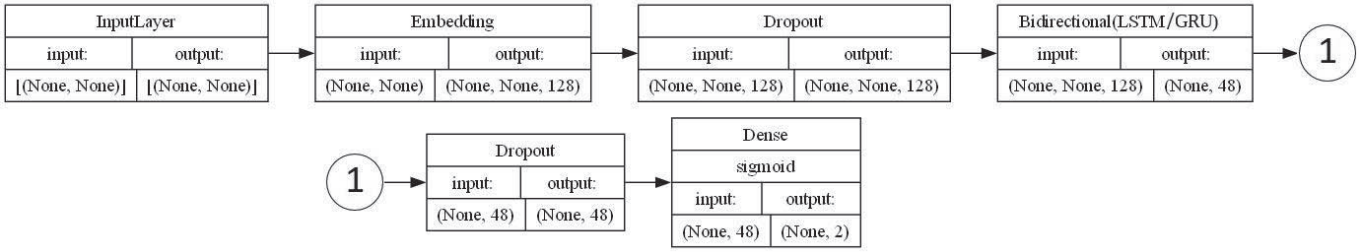

Рис. 5. Архітектура моделі з одним рекурентним BiLSTM/BiGRU-шаром та методами регуляризації

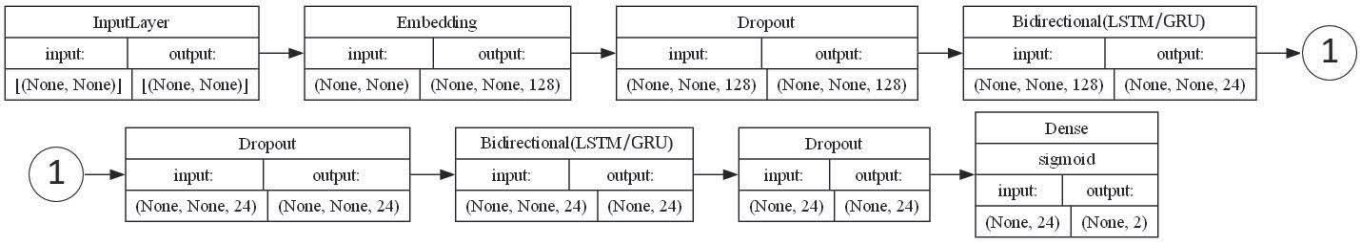

Рис. 6. Архітектура моделі з двома рекурентними BiLSTM/BiGRU-шарами та методами регуляризації

Таблиця 3

Особливості бібліотек для машинного навчання

Характеристика	**TensorFlow**	**PyTorch**	**Keras**	**Scikit-learn**
Рік випуску	2015	2016	2015	2007
Розробник	Google	Facebook (Meta)	Google (підмодуль TensorFlow)	Інститут Франсуа Шоле
Мова програмування	Python, C++, Java, Go, JavaScript, Swift	Python, C++	Python	Python, C++
Основне використання	Глибоке навчання, нейронні мережі	Глибоке навчання, нейронні мережі	Інтерфейс для TensorFlow/високорівневе API	Класичні ML-алгоритми, регресія, класифікація
Підтримка CPU/GPU	Так (підтримує GPU через CUDA, TPU)	Так (GPU через CUDA)	Так (через TensorFlow або Theano)	Лише CPU (GPU підтримується через обгортки)
Модульність	Висока (TensorFlow 2.0 спрощений)	Висока (динамічні обчислювальні графи)	Висока (зручний API поверх TensorFlow)	Середня (для традиційних ML-моделей)
Динамічний граф	Ні (але є функція Eager Execution у TF 2.0)	Так (основа PyTorch)	Ні (покладається на TensorFlow)	Ні
Простота використання	Відносно складна, але зростає з TF 2.0	Простий для прототипування, інтуїтивний	Дуже простий (високий рівень абстракції)	Дуже простий (для класичних задач ML)
Гнучкість	Висока (низький рівень контролю)	Висока (простий контроль над обчисленнями)	Обмежена (високий рівень абстракції)	Середня (зосереджена на класичних алгоритмах)
Обчислювальні графи	Статичні (але є динамічна можливість)	Динамічні (створюються під час виконання)	Статичні (через TensorFlow або Theano)	Ні
Розподілені обчислення	Так, відмінна підтримка (TPU, кластеризація)	Обмежена підтримка	Через TensorFlow	Ні
Підтримка мобільних пристроїв	Так (TensorFlow Lite)	Ні	Так (через TensorFlow Lite)	Ні
Експорт моделей	TensorFlow SavedModel, HDF5, TF.js, TF Lite	TorchScript, ONNX	HDF5, TensorFlow SavedModel	Pickle, ONNX
Спільнота та ресурси	Велика, багато офіційної документації, підтримка від Google	Велика, активна спільнота, багато прикладів	Велика спільнота через TensorFlow/Keras	Дуже велика для класичного ML
Продуктивність	Висока продуктивність, особливо на великих моделях	Висока продуктивність на GPU	Висока (через TensorFlow)	Висока продуктивність для традиційного ML
Використання в індустрії	Широко використовується, особливо у великих проєктах (Google, Uber, Airbnb)	Використовується для наукових досліджень і прототипування	Широко використовується для швидкої розробки	Дуже популярний у наукових дослідженнях і стартапах

Як середовище виконання було обрано *Google Colab* [5], тому що воно надає користувачеві (досліднику, розробнику) «в хмарі» апаратні ресурси технологій GPGPU та TPU, а також в ньому можна використовувати всі наведені бібліотеки для машинного навчання.

Під час використання платформи *Google Colab* було визначено наступні переваги:

1. Доступ до потужних GPU і TPU. Як відомо, графічні та тензорні процесори значно прискорюють тренування моделей глибокого навчання. *Google Colab* дозволяє використовувати ці обчислювальні потужності, що важливо для проєктів, які потребують великих ресурсів;
2. Це дає змогу тренувати моделі, які потребують багато обчислень, без необхідності купувати дорогий апаратний ресурс;
3. Хмарне середовище: *Google Colab* працює у браузері, тому користувачеві не потрібно нічого встановлювати локально на свій комп'ютер. Всі обчислення виконуються на серверах *Google*, а дані та написані сценарії автоматично зберігаються на хмарному носії *Google Drive*, що полегшує збереження та доступ до проєктів;
4. Інтеграція з *GitHub*: ця можливість дозволяє відкривати, редагувати та зберігати вихідні тексти програм безпосередньо в репозиторіях *GitHub*;
5. Легкий обмін та співпраця: користувачі можуть ділитися своїми сценаріями через посилання, як це відбувається з *Google Docs* або *Google Sheets*. Інші ж користувачі можуть редагувати або переглядати ваші зміни вихідних текстів у режимі реального часу. Це зручно для командної роботи, наукових досліджень або навчання, коли існує необхідність працювати над одним проєктом одночасно з кількома користувачами;
6. Підтримка основних бібліотек для машинного навчання. *Google Colab* має попередньо встановлені популярні бібліотеки, такі як *TensorFlow*, *Keras*, *PyTorch*, *Scikit-learn*, *Pandas*, *NumPy* та інші. Це дозволяє швидко почати роботу, не гаючи часу на встановлення пакетів;
7. Інтерактивне *Python*-середовище. *Colab* використовує *Jupyter Notebook* у хмарі, що дозволяє писати *Python*-сценарії, виконувати його по блоках, а також додавати візуалізації, текстові блоки та графіки. Це інтерактивне середовище ідеально підходить для експериментів з даними та розробки моделей машинного навчання, наукових досліджень і навчання;
8. Підтримка інших мов програмування. Хоча основною мовою в *Google Colab* є *Python*, також існує можливість виконувати сценарії, написані іншими мовами, за допомогою спеціальних команд. Наприклад, R, JavaScript, SQL тощо;
9. Розширення для візуалізації: Colab підтримує бібліотеки для візуалізації, такі як Matplotlib, Seaborn, Plotly та інші, що дозволяє створювати графіки, діаграми та візуалізації результатів безпосередньо в Jupyter-блокнотах;
10. Безперервне навчання моделей, тобто існує можливість виконувати процеси навчання моделей протягом кількох годин на серверах Google Colab, що зручно для довгих тренувань, особливо у роботі з великими нейронними мережами

Аналіз архітектур паралельних обчислень та їх застосування для навчання моделей фільтра

Основною особливістю сучасних графічних відеоадаптерів, які використовують графічні процесори (GPU), є наявність набору потокових мультипроцесорів (SM), що використовувалися раніше лише в алгоритмах і задачах, пов'язаних з обробкою графічних зображень. Програмні технології (інтерфейси), що застосовуються розробни-

ками програмного забезпечення для створення програм такого напрямку, використовують пам'ять відеоадаптера для розміщення структур даних, таких як текстури, буфери, визначають конвеєр обробки, кожен етап якого відповідає за свої специфічні дії, а саме: растеризацію, інтерполяцію, блендинг, теселяцію [2] тощо.

Технологія обчислень загального призначення на графічних процесорах (GPGPU) ґрунтується на використанні великої кількості процесорів GPU, що працюють паралельно, для обробки даних за допомогою алгоритмів загального призначення (наукових чи інших, але не обов'язково пов'язаних з обробкою зображень).

Потоковий процесор на GPU має простішу структуру порівняно з обчислювальним вузлом CPU. Тобто такі вузли менш універсальні і виконують менший набір функцій, аніж вузли процесора. Проте, оскільки їхня кількість велика, то для певних типів задач можна значно підвищити швидкодію. Найкращого прискорення вдається досягти для алгоритмів, що підтримують концепцію паралелізму за даними (один паралельний потік обробляє свою область у пам'яті). Тому зазвичай GPGPU застосовують у наступних сферах: обробка зображень (Reduction, Histogram, Fast Fourier Transform, Summed Area Table); обробка відеоданих (Transcode, Digital Effects, Analysis); лінійна алгебра; моделювання (Technical, Finance, Academic, Some Databases) [1] тощо.

TPU (Tensor Processing Unit) – це спеціалізований інтегральний процесор, розроблений компанією Google, який оптимізований для виконання операцій глибокого навчання, зокрема, обчислень з тензорами в рамках машинного навчання. Він використовується для прискорення роботи моделей штучного інтелекту, особливо в системах, що працюють з великими обсягами даних і потребують високої продуктивності.

У Таблиці 4 наведено порівняльну характеристику технологій GPGPU та TPU:

Отже, основна відмінність полягає в тому, що TPU розроблені спеціально для швидкої обробки великих обсягів даних, які використовуються в машинному навчанні (зокрема, нейронних мережах), тоді як GPU є універсальнішими і застосовуються як у графічних, так і в обчислювальних задачах, включно із тренуванням моделей машинного навчання. Кожна з технологій має свої переваги, і вибір між ними визначається конкретними вимогами до продуктивності та характером обчислювальних задач.

Ефективність застосування технологій GPGPU та TPU до задачі забезпечення якості коментарів до внесених змін у вихідних текстах програм для систем контролю версій показано в Таблиці 5.

Результати цього експерименту показали, що найбільшого прискорення навчання моделей вдалося досягти саме з використанням GPU.

Оцінка методу забезпечення якості коментарів

Для оцінки якості забезпечення коментарів до внесених змін систем контролю версій використано такі метрики [13]:

1. частка правильних відповідей (*Accuracy*):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \quad (10)$$

де TP (True Positive) – вірні позитивні відповіді;

TN (True Negative) – вірні негативні відповіді;

FP (False Positive) – невірні позитивні відповіді;

FN (False Negative) – невірні негативні відповіді.

2. частка вірних позитивних відповідей серед усіх позитивних відповідей класифікатора (*Precision*):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (11)$$

3. частка вірних спрацювань на позитивних об'єктах (*Recall*):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (12)$$

4. середнє гармонійне (F1-score):

$$\text{F}_1 = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (13)$$

Таблиця 4

Порівняльна характеристика технологій GPGPU та TPU

Характеристика	TPU	GPU
Призначення	Оптимізовано для виконання операцій з тензорами, орієнтованих на обчислення в нейронних мережах (машинне навчання, зокрема TensorFlow)	Широко застосовується для паралельних обчислень, графіки, рендерингу і тренування нейронних мереж
Виробник	Google	Nvidia, AMD
Архітектура	Спрямована на матричні операції (використовує матричні множники)	Паралельна обробка, тисячі ядер для обчислень
Тип обчислень	Високоефективні для операцій з низькою точністю (FP16, INT8)	Гнучка точність обчислень (FP32, FP64), підходить для широкого спектру обчислень
Програмне забезпечення	Спеціалізоване для TensorFlow, Google Cloud	Підтримує TensorFlow, PyTorch, Keras та інші фреймворки
Швидкість обробки	Висока продуктивність при тренуванні нейронних мереж	Залежить від моделі, проте найсучасніші GPU мають високу швидкість, але відстають від TPU у певних задачах
Енергоспоживання	Оптимізовані для низького енергоспоживання при виконанні ML задач	Високе енергоспоживання, особливо у потужних моделях для машинного навчання
Ціна	Доступний лише як хмарна послуга Google (не доступний для індивідуального продажу)	Можна придбати як окремі пристрої різного рівня продуктивності
Застосування	Виключно для машинного навчання, нейронних мереж	Широке застосування: графіка, симуляції, машинне навчання, рендеринг
Пам'ять (тип і розмір)	Спеціалізована швидка пам'ять для ML (HBM)	GDDR5, GDDR6, HBM, різні об'єми пам'яті
Гнучкість у використанні	Менш універсальні, спеціалізовані для ML	Висока універсальність, використовується в багатьох сферах

Таблиця 5

Порівняльна таблиця часу виконання навчання моделей за допомогою CPU, TPU та GPU

Модель	**CPU (Intel Xeon з 2-ма ядрами)**	**TPU другого покоління з 8-ма ядрами**	**GPU (Nvidia Tesla T4 з 320-ма тензорними ядрами)**
Dense (без методів регуляризації)	44 хв 7 с	9 хв 47 с	2 хв 26 с
Dense (з регуляризацією)	57 хв 52 с	7 хв 33 с	1 хв 55 с
LSTM (48 нейронів)	59 хв 34 с	15 хв 56 с	1 хв 4 с
BiLSTM (1 шар з 24-ма нейронами)	1 год 10 хв 59 с	11 хв 38 с	1 хв 33 с
BiLSTM (2 шари з 12-ма нейронами кожний)	1 год 14 хв 34 с	16 хв 56 с	2 хв 35 с
BiGRU (1 шар з 24-ма нейронами)	1 год 3 хв 19 с	10 хв 36 с	1 хв 34 с
BiGRU (2 шари з 12-ма нейронами кожний)	1 год 5 хв 55 с	18 хв 53 с	2 хв 49 с

Хоча варто врахувати, що для фільтра повідомлень про внесені зміни важливіша саме повнота (*Recall*), тому що краще

відхилити повідомлення та порекомендувати розробнику доповнити його або перефразувати.

Отже, результати роботи створених моделей наведені у Таблиці 6.

Таблиця 6

Порівняння ефективності побудованих моделей для імплементації фільтра повідомлень про внесені зміни

Модель	**Метрика**	**Без використання методів, що долають проблему нерівномірних даних**	**Зсув порогу (threshold moving)**
Dense (без методів регуляризації)	Accuracy F1-score	0.786 0.622	0.788 0.616
Dense (з регуляризацією)	Accuracy F1-score	0.816 0.648	**0.817** 0.637
LSTM (48 нейронів)	Accuracy F1-score	0.801 0.445	0.801 0.445
BiLSTM (1 шар з 24-ма нейронами)	Accuracy F1-score	0.796 0.664	0.798 0.655
BiLSTM (2 шари з 12-ма нейронами кожний)	Accuracy F1-score	0.794 **0.677**	0.796 0.675
BiGRU (1 шар з 24-ма нейронами)	Accuracy F1-score	**0.808** **0.651**	0.811 0.629
BiGRU (2 шари з 12-ма нейронами кожний)	Accuracy F1-score	0.791 0.635	0.790 0.630

Динаміку навчання моделей представлено на Рис. 7-9, а саме зображено графіки значення точності та середнього гармонійного залежно від епохи.

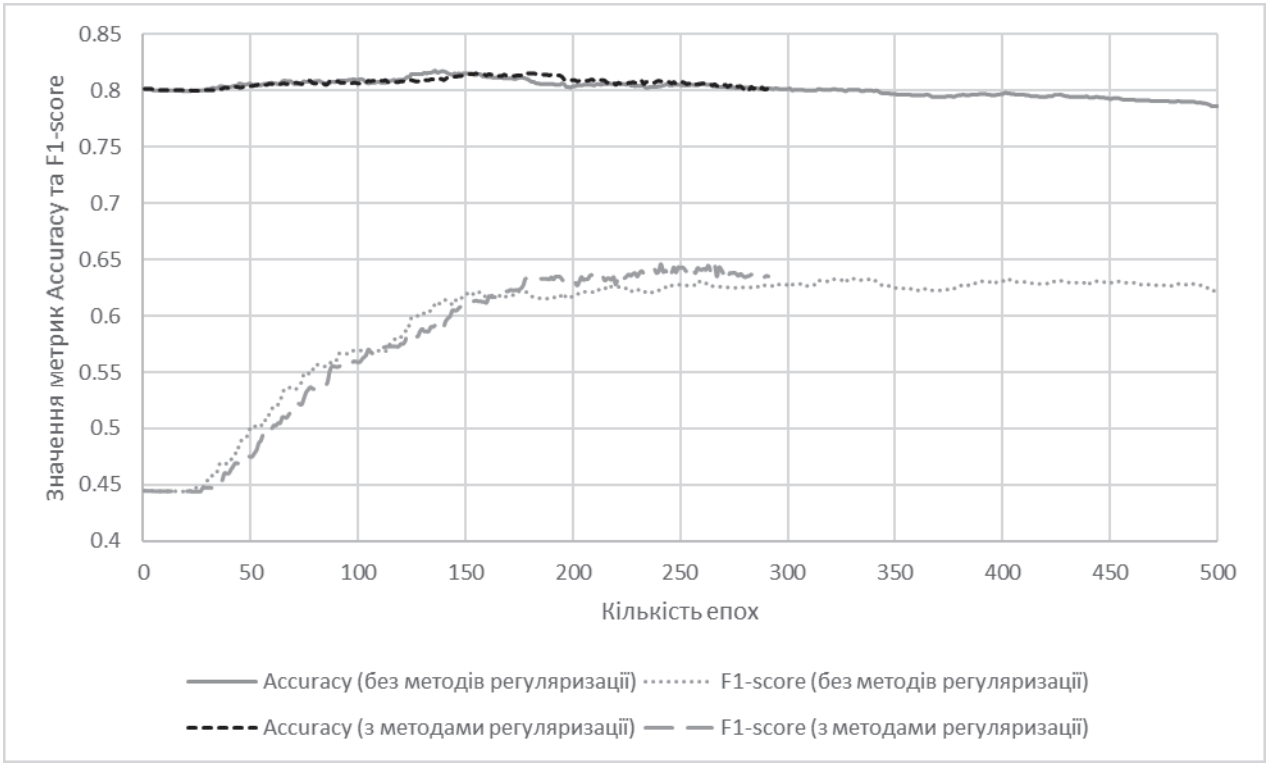

Рис. 7. Графік залежності значення різних метрик оцінювання якості навчання нейронних мереж з Dense-шаром від порядкового номера епохи навчання

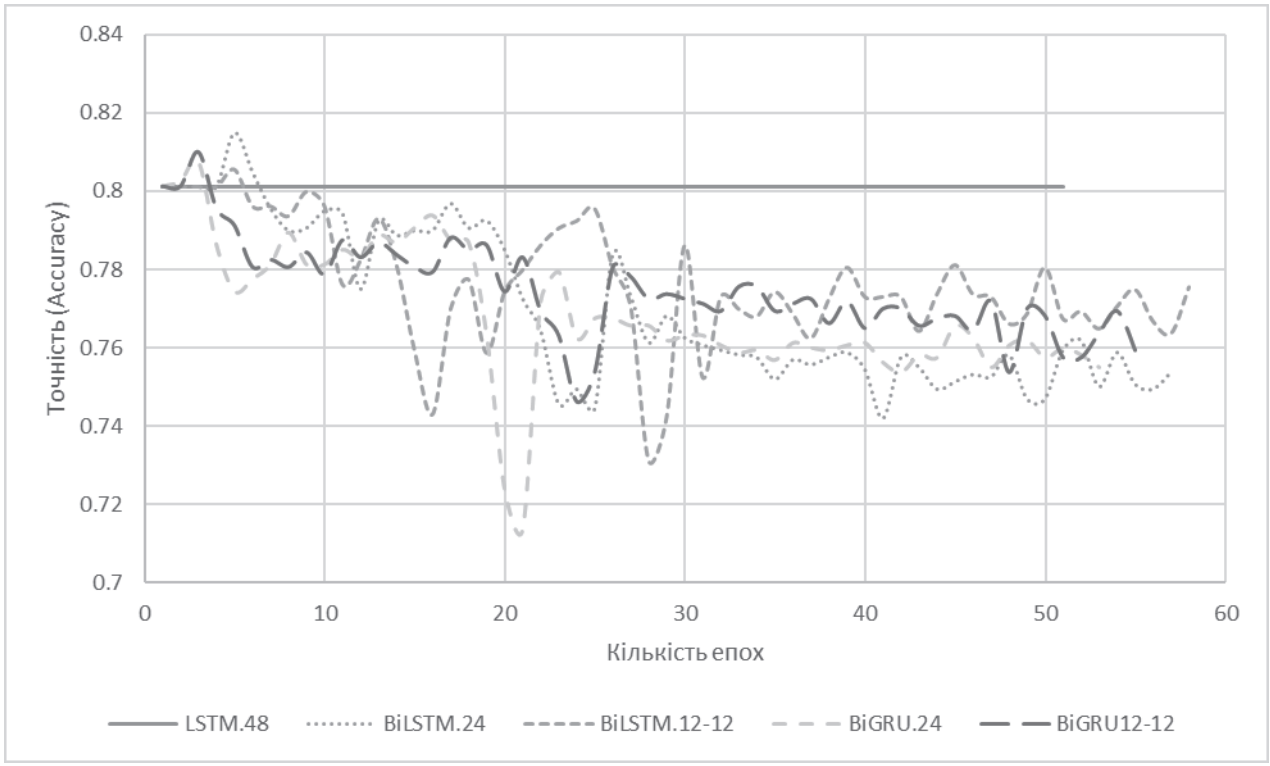

Рис. 8. Графік залежності значення точності рекурентних нейронних мереж від порядкового номера епохи навчання

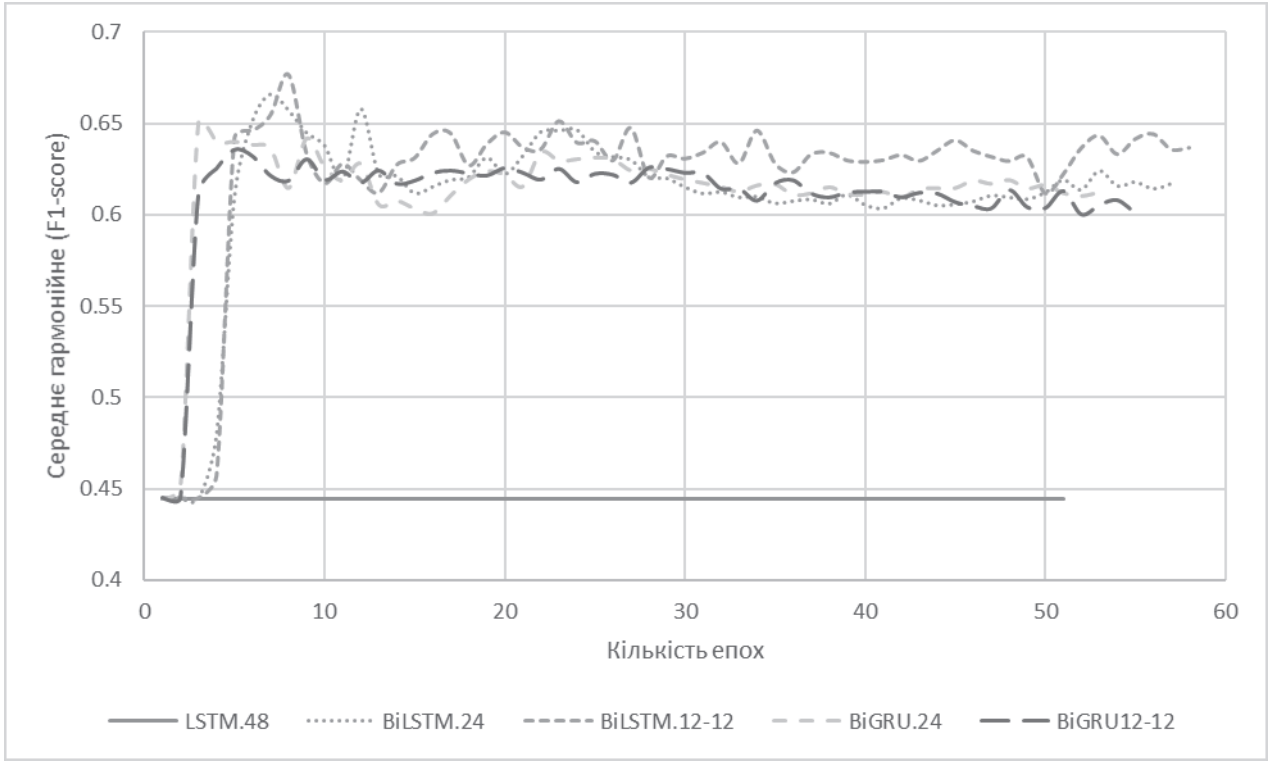

Рис. 9. Графік залежності значення метрики середнього гармонійного рекурентних нейронних мереж від порядкового номера епохи навчання

Графіки точності RNN-мереж мають дещо стрибкоподібний характер через особливості рекурентних нейронних мереж, тому що вони «працюють» із послідовними даними, і точність може різко змінюватися залежно від того, наскільки добре модель вловила послідовні зв'язки в певних наборах даних. Якщо модель починає «вчитися» на певних зразках і покращує точність для конкретних послідовностей, то вона може втрачати здатність узагальнювати інші зразки, що призводить до стрибків.

Висновки

У роботі запропоновано різні підходи до імплементації фільтра повідомлень до внесених змін. Цей фільтр є одним з етапів аналізу, обробки та підготовки даних для методу, який буде мати можливість створювати повідомлення про внесені зміни на їхній основі.

Проведено порівняльний аналіз основних програмно-апаратних платформ, запропонованих у хмарному середовищі Google Colab задля прискорення навчання моделей фільтра, а саме: CPU, TPU та GPU. Часові заміри тривалості навчання показують доцільність використання тензорного процесору (TPU) від корпорації Google та графічного адаптеру (GPU) від Nvidia у порівнянні з центральним процесором (CPU) від Intel. Результати показують домінування графічного адаптеру Nvidia Tesla T4 над Google TPU v2 у 15 разів, а над Intel Xeon – у 45 разів. У свою чергу TPU перевершує CPU у 6 разів.

Виконано оцінку методу забезпечення якості описів до внесених змін. З таблиці результатів видно, що найкращий результат має RNN-мережа, яка містить один двонаправлений (bidirectional) GRU-шар з 24-ма нейронами для видобутку ознак тексту: середнє гармонійне становить 65.1 %, а повнота – 80.8 %. Окрім того, така архітектура мережі навчається з методами регуляризації за 1 хвилину та 34 секунди за допомогою GPU. Хоча послідовна та інші рекурентні моделі (окрім, LSTM) також мають порівняні результати, як щодо точності, так і щодо швидкості.

Показано, що середовище *Google Colab* є потужним інструментом для швидкого прототипування та розробки моделей машинного навчання. Завдяки своїм можливостям (GPU, TPU та інтеграції з хмарними сервісами) він стає одним із найзручніших інструментів для студентів, дослідників та інженерів програмного забезпечення.

Література

1. Погорілий С.Д., Семьонов Б.О. Дослідження паралельних алгоритмів мовою Python з використанням різних платформ. *Наукові записки НаУКМА*. Вип. 198, 2017. С. 14–21.
2. Погорілий С.Д., Семьонов Б.О. Дослідження програмної бібліотеки Linpack на архітектурі CUDA мовою програмування Python. *Наукові праці ДонНТУ*. Вип. 23, № 2. С. 98–106.
3. Buse R., Weimer W. Automatically documenting program changes. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*(2010).
4. GitHub Docs GitHub REST API. *GitHub Docs*. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28 (дата звернення: 07.10.2024).
5. Google Colaboratory – Google. *research.google.com*. 2024. URL: https://research.google.com/colaboratory/faq.html (дата звернення: 07.10.2024).
6. How to Write a Git Commit Message. *cbeams*. 30.08.2014. URL: https://cbea.ms/git-commit/#seven-rules (дата звернення: 07.10.2024).
7. Jiang S., Armaly A., McMillan C. Automatically generating commit messages from diffs using neural machine translation. *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*(2017).
8. Keras Home - Keras Documentation. *Keras.io*. 2019. URL: https://keras.io/ (дата звернення: 07.10.2024).
9. Loshchilov I., Hutter F. Decoupled weight decay regularization. *Proceedings of the ICLR*(2019).
10. N.K. Nissa Text Messages Classification using LSTM, Bi-LSTM, and GRU. *Medium*. 23.08.2022. URL: https://nzlul.medium.com/theclassification-of-text-messages-using-lstm-bi-lstm-and-gruf79b207f90ad (дата звернення: 07.10.2024).
11. N.V. Otten How To Use Text Normalization Techniques In NLP With Python [9 Ways. *Spot Intelligence*. 25.01.2023. URL: https://spotintelligence.com/2023/01/25/text-normalization-techniquesnlp/ (дата звернення: 07.10.2024).
12. PyTorch PyTorch documentation — PyTorch master documentation. *Pytorch.org*. 2019. URL: https://pytorch.org/docs/stable/index.html (дата звернення: 07.10.2024).
13. Scikit-Learn scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation.

Scikit-learn.org. 2019. URL: https://scikit-learn.org/ (дата звернення: 07.10.2024).
14. TensorFlow TensorFlow. *TensorFlow*. 2019. URL: https://www.tensorflow.org/ (дата звернення: 07.10.2024).

Одержано: 26.12.2024
Внутрішня рецензія отримана: 08.01.2025
Зовнішня рецензія отримана: 08.01.2025

Про авторів:

[1]*Семьонов Богдан Олександрович*,
аспірант.
https://orcid.org/0009-0001-3692-9415.

[2]*Погорілий Сергій Дем'янович*,
доктор технічних наук, професор.
https://orcid.org/0000-0002-6497-5056.

Місце роботи авторів:

[1]Київський національний
університет імені Тараса Шевченка,
тел. +380 (66) 451-97-00
E-mail: bohdan.semonov@gmail.com

[2]Київський національний
університет імені Тараса Шевченка,
Тел. +380 (66) 434-27-86
E-mail: sdp7799@gmail.com