УДК 004.83

Б. О. Семьонов, С. Д. Погорілий
Київський національний університет імені Тараса Шевченка
проспект Академіка Глушкова, 4,Г, 03022 Київ, Україна

Створення методу забезпечення якості коментарів у системах контролю версій на основі трансформерних моделей

Обґрунтовано важливість розв'язання задачі підвищення якості описів до змін у вихідних текстах програм у контексті систем контролю версій. Для фільтрації коментарів застосовано методи машинного навчання, зокрема нейронні мережі різних архітектур. Використання нейронних мереж є доцільним через потребу в автоматичному виявленні описів, що точно відображають призначення внесених змін. Проведено порівняльний аналіз моделей на основі Transformer-архітектур, таких як BERT, RoBERTa та DistilBERT, та їхнє застосування у бінарних класифікаторах для фільтрації змін. Здійснено навчання моделей на множині описів до внесених змін, отриманих за допомогою спеціального програмного інтерфейсу GitHub REST API. Проведено оцінювання точності моделей через використання метрик: точності (Accuracy) та середнього гармонійного (F1-score). Також підтверджено ефективність середовища Google Colab для прототипування моделей машинного навчання.

Ключові слова: *AdamW-алгоритм, BERT, commit message, DistilBERT, GitHub REST API, RoBERTa, Transformer, вихідний текст програми, ПЗ, повідомлення про внесені зміни, програмне забезпечення, репозиторій, середнє гармонійне, система контролю версій.*

Вступ

В епоху цифрових технологій, коли розробка програмного забезпечення відіграє важливу роль у різних галузях, системи контролю версій стають невід'ємним інструментом для управління вихідним кодом. Динамічні зміни ринку вимагають від розробників не лише швидкості та якості роботи, а й структурованого та надійного підходу до керування версіями програм [1].

Системи контролю версій дають розробникам змогу гнучко працювати над проєктами, вносити зміни та тестувати нові функції, зберігаючи, за потреби, можливість повернення до попередніх версій. Це допомагає запобігти втраті даних, підтримувати стабільність системи та покращувати командну співпрацю

© Б. О. Семьонов, С. Д. Погорілий

Збереження змін у системі контролю версій відіграє ключову роль у розумінні, відстеженні та збереженні історії модифікацій у кодовій базі. Кожне повідомлення про зміни відображає конкретну правку, внесену розробником. Детальний опис змін сприяє тому, щоб як поточні, так і майбутні члени команди могли зрозуміти, які коригування було зроблено та з якою метою [2].

У підсумку, детальні описи змін покращують документацію проєкту, спрощують технічну підтримку, дозволяють відстежувати етапи його розвитку та логіку прийняття рішень, а також роблять проєкт більш прозорим.

Розробка методу забезпечення якості коментарів до внесених змін у вихідному коді для систем контролю версій стає особливо важливою через збільшення обсягів проєктів і ускладнення структури кодової бази.

Оскільки до сучасних проєктів можуть бути залучені розробники з різним досвідом та підходами до програмування, фільтр описів змін, що оцінює зміст і контекст коментарів, сприятиме кращому розумінню змін іншими членами команди та полегшить технічну підтримку [3].

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

1. Правила щодо оформлення повідомлення про внесені зміни [4]:

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 вживатися у доконаній формі.

2. Правила щодо подання інформації у повідомленні про внесені зміни:

а) має бути описано, чому були зроблені зміни;

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, щоби фахівець, який не має розуміння щодо проблеми чи структури коду, зрозумів, що було зроблено, і який вплив це має на проєкт (програму). Приклади описів наведено в табл. 1.

Таблиця 1. Коментарі до внесених змін, які відповідають правилу 2.г

№ за/п	Приклад повідомлення	Мета повідомлення
1	Refactored authentication logic to improve code readability and reduce duplication	Покращити читабельність коду та усунути надлишкові фрагменти
2	Fixed race condition in data loader to prevent intermittent test failures	Усунути причину нестабільного проходження тестів
3	Optimized image rendering to improve page load performance on mobile devices	Пришвидшити завантаження на мобільних пристроях
4	Updated password hashing algorithm to enhance security compliance	Підвищити рівень безпеки відповідно до вимог
5	Added error messages to login form to improve user feedback	Забезпечити більш інтуїтивну взаємодію з користувачем
6	Rewrote unit tests to align with updated API behavior	Актуалізувати тести відповідно до оновленої логіки роботи API

Постановка задачі

Усі, 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, так як 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».

Формалізація задачі фільтра. Нехай $\boldsymbol{X}$ — множина описів внесених змін (генеральна сукупність), кожне із цих повідомлень про внесені зміни описується m-вимірним вектором слів:

$$\vec{x} = \{x_1, x_2, \dots, x_m\}, \vec{x} \in \boldsymbol{X}, \tag{1}$$

де m — розмірність вектора слів.

$\boldsymbol{Y}$ — множина можливих відгуків (міток) у вигляді M-вимірного вектора:

$$\vec{y} = \{y_1, \dots, y_M\}, \vec{y} \in \boldsymbol{Y}, \tag{2}$$

де $M = 2$ —— це кількість відгуків (класів), які потрібно отримати: перший відгук — чи задовольняє поточне повідомлення вимогам, другий – не задовольняє. Звідси випливає, що шуканий фільтр – це задача бінарної класифікації: $Y = \{0, 1\}$ [5].

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

$$f: \boldsymbol{X} \to \boldsymbol{Y}. \tag{3}$$

Збір і підготовка навчальних корпусів повідомлень

Для збору даних для навчання було використано один із найбільших вебсервісів для розміщення IT-проєктів і спільної розробки програмного забезпечення GitHub, в основі якого лежить відома система контролю версій Git. Цей веб-сервіс має спеціальний програмний інтерфейс для застосунків, який називається GitHub REST API [6]. Так як 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 сервіси мають тип GET http-запиту, 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.

Указані у табл. 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.

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

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

Після аналізу промаркованих описів внесених змін було зроблено висновок, що дані є нерівномірними. В отриманій вибірці описи, які не відповідають правилу 2.г, кількісно переважають повідомлення, які відповідають вимогам.

Перед подачею повідомлень про внесені зміни на вхід досліджуваних нейронних мереж (трансформерів) було виконано:

1) токенізацію повідомлень на навчальній вибірці [7];

2) векторизацію повідомлень лише для однієї моделі на основі encoder-архітектури трансформера, тому що інші попередньо навчені моделі мають вбудовану векторизацію [8].

Метод забезпечення якості коментарів до внесених змін у вихідних текстах програм для систем контролю версій

В основу першого фільтра (рис. 1) покладено модель, побудовану на основі «класичної» *encoder*-архітектури трансформера (t*ransformer-encoder*). Ця модель складається з:

1) трьох вагових шарів (weighted layers):

а) вхідного (Embedding) шару;

б) одного трансформерного блоку, який застосовує багатоголовий механізм уваги з 4 головами та має прихований шар розміром 512 нейронів;

в) вихідного Dense-шару [9] з двома нейронами, який повертатиме бажаний відгук (M = 2);

2) проміжного шару без ваг (Global Average Pooling), який «конденсує» всю вхідну послідовність в один вектор фіксованої довжини для подальшої передачі в Dense-шар [9].

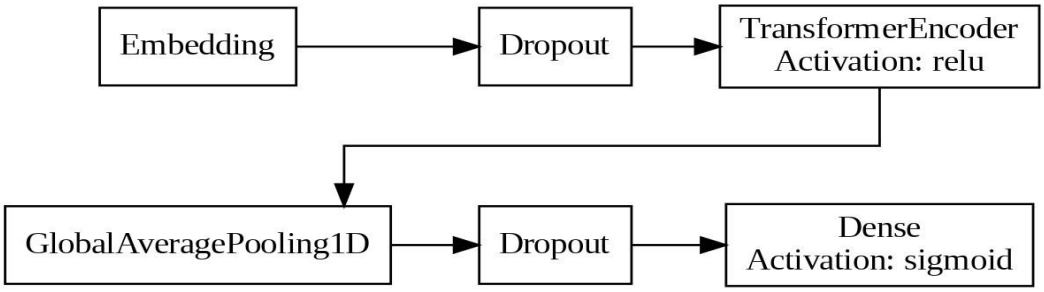

Рис. 1. Архітектура, побудована на основі encoder-частини «класичного» трансформера

Як алгоритм оптимізації ваг обрано AdamW [10], що, у свою чергу, 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 та 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:

$$g_t = \nabla_\theta f(\theta_t), \tag{4}$$

де $f(\theta)$ — це функція втрат; g_t — градієнт для поточних параметрів;

3) оновлення моментів:

а) оновлення першого моменту (оцінка середнього градієнта):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t; \tag{5}$$

б) оновлення другого моменту (оцінка дисперсії градієнта):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2; \tag{6}$$

4) коригування зміщення моментів: для того щоб компенсувати зміщення на початкових етапах (оскільки $m_0 = 0$ та $v_0 = 0$), відбувається коригування:

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}, \tag{7}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}; \tag{8}$$

5) оновлення параметрів з урахуванням вагового спаду та адаптивної швидкості навчання:

$$\theta_t = \theta_{t-1} - \eta \left(\frac{\widehat{m_t}}{\sqrt{\widehat{v_t} + \epsilon}} + \lambda \theta_{t-1} \right), \tag{9}$$

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).

Для зменшення прояву явища перенавчання, використано методи регуляризації:

1) введено додатковий шар Dropout з імовірністю 30 % — випадкове виключення нейронів шару із заданою ймовірністю p під час кожної ітерації навчання, метою якого є перешкоджання спільній адаптації ваг нейронів шару, яка, у свою чергу, призводить до перенавчання [11];

2) додано метод ранньої зупинки.

Щоб імплементувати наведену вище модель було використано мову програмування ***Python*** (версії **3.11.12**) та застосовано бібліотеку ***TensorFlow*** (**2.18.0**). У табл. 3 наведено порівняльну характеристику між *TensorFlow* та іншими популярними бібліотеками для машинного навчання, такими як *PyTorch* [12], *Keras* [13] та *Scikit-learn* [14]. Кожна із цих бібліотек має свої унікальні можливості та підходить для різних типів завдань.

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

Характеристика	TensorFlow	PyTorch	Keras	Scikit-learn
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

Для побудови наступних фільтрів було застосовано високорівневі обгортки з бібліотеки *TensorFlow*, які базуються на трансформених моделях інших типів, а саме: ***BERT*** [15], ***RoBERTa*** [16], ***DistilBERT*** [17] та ***XLM-RoBERTa*** [18]. Основні особливості цих моделей наведено в табл. 4.

Таблиця 4. Загальна порівняльна характеристика трансформерних моделей

Характеристика	BERT	RoBERTa	DistilBERT	XLM-RoBERTa
Повна назва	Bidirectional Encoder Representations from Transformers	Robustly Optimized BERT Pretraining Approach	Distilled BERT	Cross-lingual RoBERTa
Архітектура	Двонапрямлений трансформер (Encoder)	Покращений BERT	Спрощений (менше шарів) BERT	Мультимовний RoBERTa
Розмір	~110 млн параметрів	~125 млн параметрів	~66 млн параметрів	~270 млн параметрів
Підтримка мов	Англійська	Англійська	Англійська	Більше 100 мов
Дані для навчання	BooksCorpus та Wikipedia	Більший корпус: CommonCrawl, WebText, CC-News	Те саме, що BERT	CommonCrawl (більше 100 мов)
Маскування (MLM)	Маскування токенів (15%)	Динамічне маскування	Те саме, що в BERT	Те саме, що RoBERTa
NSP (Next Sentence Prediction)	Так	Ні	Ні	Ні

У роботі використовується **16** попередньо натренованих моделей, які були донавчені на корпусах повідомлень до внесених змін, зібраних за допомогою сервісів GitHub REST API. Таким чином, створено також шістнадцять фільтрів описів внесених змін. Дані щодо їхніх параметрів зведено в табл. 5, архітектуру зображено на рис. 2.

Таблиця 5. Основні параметри попередньо натренованих моделей

Назва моделі	Кількість параметрів, млн	Шари	Голови	Розмір вектора ознак	Мова	Збереження регістра літер
bert_tiny_en_uncased	~4	2	2	128	Англійська	Ні
bert_small_en_uncased	~29	4	4	256	Англійська	Ні
bert_medium_en_uncased	~42	8	8	512	Англійська	Ні

bert_base_en_uncased	~110	12	12	768	Англійська	Ні
bert_base_en	~110	12	12	768	Англійська	Так
bert_base_multi	~110	12	12	768	Більше 100 мов	Так
bert_large_en_uncased	~340	24	16	1024	Англійська	Ні
bert_large_en	~340	24	16	1024	Англійська	Так
bert_tiny_en_uncased_sst2	~4	2	2	128	Англійська	Ні
roberta_base_en	~125	12	12	768	Англійська	Так
roberta_large_en	~355	24	16	1024	Англійська	Так
distil_bert_base_en_uncased	~66	6	12	768	Англійська	Ні
distil_bert_base_en	~66	6	12	768	Англійська	Так
distil_bert_base_multi	~134	6	12	768	Більше 100 мов	Так
xlm_roberta_base_multi	~270	12	12	768	Більше 100 мов	Так
xlm_roberta_large_multi	~550	24	16	1024	Більше 100 мов	Так

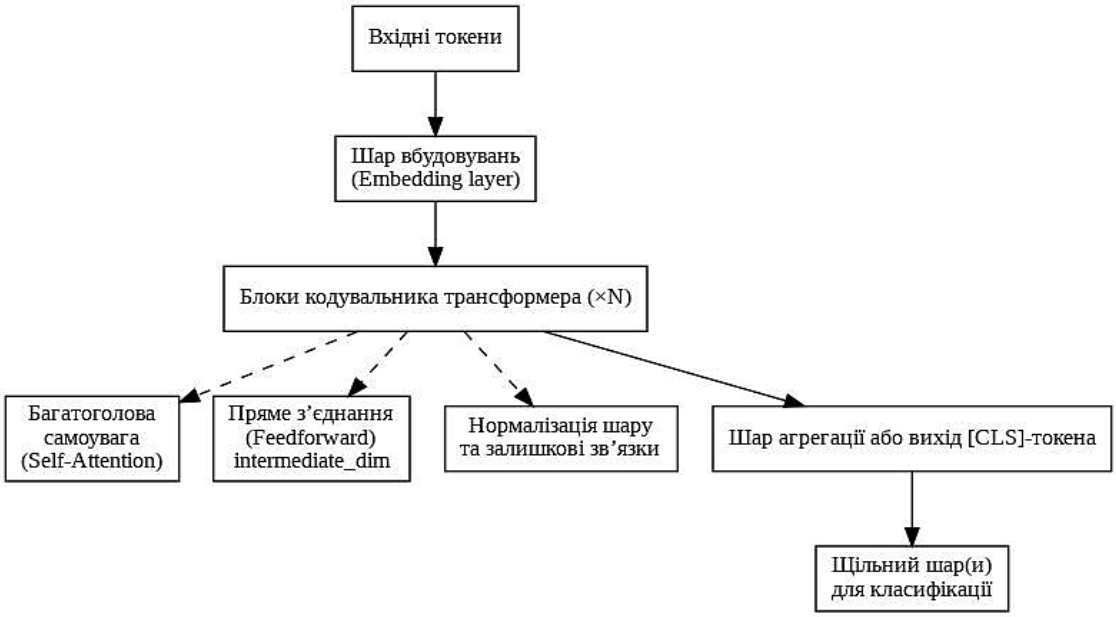

Рис. 2. Узагальнена архітектура моделей BERT, RoBERTa, DistilBERT та XLM-RoBERTa

Нижче наведено пояснення до позначень, використаних у табл. 5:

1) назва моделі — це міжфреймворкові стандартизовані позначення архітектур і конфігурацій трансформерів. Вони використовуються в *TensorFlow*, *PyTorch* та інших бібліотеках;

2) кількість параметрів — це орієнтовне значення, яке визначає складність моделі та впливає на її продуктивність і обсяг пам'яті;

3) шари / голови — характеристики моделі, які визначають глибину та здатність до обробки контексту;

4) розмір вектора ознак — це довжина векторного представлення кожного токена після обробки на кожному шарі трансформера;

5) мова — назва мови (набору мов), на яких тренувалася модель;

6) збереження регістру літер — індикатор, який означає, що вся текстова інформація перед обробкою приводиться до нижнього регістру (lowercase).

Основними елементами архітектури моделей BERT, RoBERTa, DistilBERT та XLM-RoBERTa є:

1) вхідні токени — послідовність вхідних елементів (токенів), отриманих після попередньої обробки тексту (токенізації);

2) шар вбудовувань (Embedding layer) — це шар, що перетворює кожен токен у відповідний вектор певної розмірності (hidden size), який слугує числовим поданням значення та позиції токена;

3) блоки кодувальника трансформера ($\times N$) — основна частина моделі, яка містить N однакових блоків. Кожен блок включає три ключові компоненти:

а) багатоголову самоувагу (Multi-Head Self-Attention) — це механізм, який дозволяє моделі одночасно звертати увагу на різні частини вхідної послідовності, порівнюючи кожен токен з усіма іншими;

б) пряме з'єднання (Feedforward layer) — нейронну мережу, яка застосовується до кожного токена окремо, має одну приховану (проміжну) проєкцію розміром intermediate_dim (наприклад, 2048 у BERT-base), що дозволяє моделі краще виражати складні залежності;

в) нормалізацію шару та залишкові зв'язки — механізми, які стабілізують навчання та пришвидшують його. Залишкові зв'язки допомагають уникнути втрати інформації, а нормалізація — зменшує залежність від масштабів вхідних даних;

4) шар агрегації або вихід [CLS]-токена — етап, на якому формується підсумкове подання всієї послідовності. Залежно від архітектури, застосовується один із варіантів:

а) CLS-токен — спеціальний токен, який агрегує інформацію з усієї послідовності (використовується в моделях: BERT, DistilBERT, RoBERTa);

б) Pooling (усереднення/максимум) — агрегування ознак усіх токенів (використовується в моделі XLM-RoBERTa);

5) шільний (Dense) шар або кілька шарів для класифікації —— один або кілька повнозв'язних (Dense) шарів, які перетворюють вихід з трансформера на конкретний прогноз.

До того ж, для навчання моделей було застосовано методологію, що базується на основі перевірочної підмножини. Вона полягає у тому, що весь доступний набір маркованих даних поділяється на такі частини, що не перетинаються, а саме: з навчальної множини (training set) було виділено 20 % на перевірочну підмножину (validation subset), а решту — для тренування моделей-кандидатів [19].

Середовищем для навчання було обрано ***Google Colaboratory*** [20], тому що воно надає користувачеві (досліднику, розробнику) «в хмарі» апаратні ресурси технологій GPGPU та TPU, а також в ньому можна застосовувати всі наведені бібліотеки для машинного навчання. Так як в роботі присутні моделі, які мають більше 100 млн параметрів, — існує потреба в значних обчислювальних ресурсах (процесорному часі та особливо пам'яті) для їхнього тренування. Такі ресурси надає графічний адаптер ***NVIDIA A100***, який є одним із найбільш потужних GPU, розроблений для обчислень у сфері глибокого навчання, HPC (високопродуктивних обчислень) та хмарних обчислень. Цей GPU часто використовується в Google Colab

Pro/Pro+, AWS, Azure, NVIDIA DGX-системах, а також у суперкомп'ютерах. Основні параметри наведено в табл. 6.

Таблиця 6. Основні параметри графічний адаптера NVIDIA A100

Параметр	**Значення**
Архітектура	NVIDIA Ampere
Кількість ядер CUDA	6912
Кількість тензорних ядер	432
Пам'ять	40 ГБ

У додаток до вищезазначеного, під час використання платформи *Google Colab* було виявлено низку її переваг, що забезпечують ефективне середовище для розробки та навчання моделей машинного навчання, а саме:

1) доступ до високопродуктивних графічних (GPU) і тензорних (TPU) процесорів: використання апаратного прискорення значно скорочує час тренування моделей глибокого навчання;

2) хмарна інфраструктура: *Google Colab* функціонує у веббраузері, що усуває необхідність встановлення спеціального програмного забезпечення на локальний комп'ютер. Усі обчислення виконуються на серверах *Google*, а збереження даних і робочих документів здійснюється автоматично у хмарному сховищі *Google Drive*. Це полегшує керування проєктами та доступ до них з різних пристроїв;

3) інтеграція з платформою GitHub: Colab дозволяє напряму відкривати, редагувати та зберігати вихідний код, розміщений у репозиторіях GitHub, що спрощує контроль версій і спільну розробку програмного забезпечення;

4) зручні засоби спільної роботи. За аналогією з *Google Docs*, користувачі можуть ділитися своїми проєктами через посилання, надаючи іншим користувачам можливість перегляду або редагування в режимі реального часу. Такий підхід сприяє ефективній командній роботі, особливо у навчальних або науково-дослідних проєктах;

5) попередньо встановлені бібліотеки для машинного навчання: *Colab* надає вбудовану підтримку найбільш поширених бібліотек, на кшталт, *TensorFlow*, *Keras*, *PyTorch*, *Scikit-learn*, *Pandas*, *NumPy* тощо. Це дозволяє миттєво розпочати реалізацію проєктів без додаткової підготовки середовища;

6) інтерактивне програмне середовище на основі *Jupyter Notebook*: платформа підтримує покрокове виконання *Python*-сценаріїв, додавання текстових пояснень, графіків і візуалізацій, що забезпечує зручність у дослідженні даних та експериментальній розробці моделей;

7) мультимовна підтримка: не зважаючи на те, що основною мовою програмування є *Python*, *Google Colab* також дозволяє виконання сценаріїв іншими мовами, такими як *R*, *JavaScript*, *SQL* тощо, що розширює сферу застосування платформи;

8) підтримка засобів для візуалізації даних: *Colab* сумісний із бібліотеками *Matplotlib*, *Seaborn*, *Plotly* та іншими, що дає змогу створювати якісні графіки та діаграми безпосередньо в середовищі блокнота;

9) можливість тривалого навчання моделей: платформа підтримує безперервне виконання обчислень протягом тривалого часу, що є важливою перевагою під час навчання складних моделей, які потребують значного обсягу обчислень.

Оцінка методу забезпечення якості коментарів

Для оцінки якості забезпечення коментарів до внесених змін систем контролю версій використано такі метрики [14]:

1) доля правильних відповідей (*Accuracy*):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{10}$$

де TP (True Positive) — вірні позитивні відповіді;

2) TN (True Negative) — вірні негативні відповіді; FP (False Positive) — невірні позитивні відповіді; FN (False Negative) — невірні негативні відповіді.

3) доля вірних позитивних відповідей серед усіх позитивних відповідей класифікатора (*Precision*):

$$Precision = \frac{TP}{TP + FP}; \tag{11}$$

4) доля вірних спрацювань на позитивних об'єктах (*Recall*):

$$Recall = \frac{TP}{TP + FN}; \tag{12}$$

5) середнє гармонійне (F1-score):

$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall}. \tag{13}$$

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 наведено у табл. 7.

Таблиця 7. Порівняння ефективності побудованих моделей
для імплементації фільтра повідомлень про внесені зміни

№ за/п	Модель	Розмір пакета	Accuracy	F1-score	Час навчання
1	transformer-encoder	128	**0.819**	**0.691**	29 сек
2	bert_tiny_en_uncased	128	**0.814**	**0.689**	2 хв 52 сек
3	bert_small_en_uncased	128	*0.199*	*0.166*	6 хв 30 сек
4	bert_medium_en_uncased	128	0.801	0.445	10 хв 35 сек
5	bert_base_en_uncased	64	*0.201*	*0.167*	24 хв 14 сек
6	bert_base_en	64	0.801	0.445	24 хв 6 сек
7	bert_base_multi	64	0.801	0.445	24 хв 4 сек
8	bert_large_en_uncased	16	0.799	0.444	1 год 15 хв 51 сек
9	bert_large_en	16	0.801	0.445	1 год 14 хв 9 сек
10	bert_tiny_en_uncased_sst2	128	**0.802**	**0.648**	3 хв 5 сек
11	roberta_base_en	64	0.801	0.445	27 хв 40 сек
12	roberta_large_en	16	0.801	0.445	1 год 18 хв 9 сек
13	distil_bert_base_en_uncased	128	0.799	0.444	12 хв 39 сек
14	distil_bert_base_en	128	0.799	0.444	12 хв 6 сек
15	distil_bert_base_multi	128	0.799	0.444	12 хв 20 сек
16	xlm_roberta_base_multi	64	0.801	0.445	25 хв
17	xlm_roberta_large_multi	16	0.799	0.444	1 год 18 хв 32 сек

У процесі навчання моделей виникла необхідність коригування розміру пакета (batch size — обсяг даних, що обробляється моделлю за одну ітерацію під час тренування), оскільки обрані моделі характеризуються значною кількістю параметрів, що унеможливлює їхнє повне завантаження в доступну пам'ять графічного адаптера обсягом 40 ГБ.

Висновки

Запропоновано низку підходів до реалізації фільтрації повідомлень щодо внесених змін, що становить один із етапів процесу аналізу, обробки та формування вхідних даних. Зазначений фільтр слугує підготовчим компонентом для подальшого застосування методу, здатного генерувати повідомлення про зміни на основі цих даних.

Здійснено порівняльний аналіз провідних моделей сімейства BERT (зокрема, BERT, RoBERTa, DistilBERT і XLM-RoBERTa), а також спеціалізованої моделі, побудованої на основі традиційної кодувальної архітектури трансформера.

Виконано оцінку побудованих фільтрів забезпечення якості описів до внесених змін. З таблиці результатів (див. табл. 7) видно, що найкращі результати мають: transformer-encoder (точність 81,9 %, середнє гармонійне 69,1 %), bert_tiny_en_uncased (точність 81,4 %, середнє гармонійне 68,9 %) та bert_tiny_en_uncased_sst2 (точність 80.2 %, середнє гармонійне 64,8 %).

Показано, що середовище *Google Colaboratory* 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 найбільш зручних інструментів для студентів, дослідників та інженерів програмного забезпечення.

1. Jiang S., Armaly A., and McMillan C., "Automatically generating commit messages from diffs using neural machine translation," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017.

2. Buse R. and Weimer W., "Automatically documenting program changes," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2010.

3. P. Xue *et al.*, "Automated Commit Message Generation With Large Language Models: An Empirical Study and Beyond," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3208–3224, 2024, doi: 10.1109/TSE.2024.3478317.

4. "How to Write a Git Commit Message," cbeams. Accessed: Oct. 07, 2024. [Online]. Available: https://cbea.ms/git-commit/#seven-rules

5. S. Pogorilyy and B. Semonov, "The Implementation of a Commit Messages Filter for Software Version Control Systems," in *The 9th International Conference on Control and Optimization with Industrial Applications*, 2024, pp. 175–179.

6. GitHub Docs, "GitHub REST API," GitHub Docs. Accessed: Oct. 07, 2024. [Online]. Available: https://docs.github.com/en/rest?apiVersion=2022-11-28

7. N.V. Otten, "How To Use Text Normalization Techniques In NLP With Python [9 Ways," Spot Intelligence. Accessed: Oct. 07, 2024. [Online]. Available: https://spotintelligence.com/2023/01/25/text-normalization-techniquesnlp/

8. A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification." 2016. [Online]. Available: https://arxiv.org/abs/1607.01759

9. TensorFlow, "TensorFlow," TensorFlow. Accessed: Oct. 07, 2024. [Online]. Available: https://www.tensorflow.org/

10. Loshchilov I. and Hutter F., "Decoupled weight decay regularization," in *Proceedings of the ICLR*, 2019.

11. N.K. Nissa, "Text Messages Classification using LSTM, Bi-LSTM, and GRU," Medium. Accessed: Oct. 07, 2024. [Online]. Available: https://nzlul.medium.com/theclassification-of-text-messages-using-lstm-bi-lstm-and-gruf79b207f90ad

12. PyTorch, "PyTorch documentation — PyTorch master documentation," Pytorch.org. Accessed: Oct. 07, 2024. [Online]. Available: https://pytorch.org/docs/stable/index.html

13. Keras, "Home - Keras Documentation," Keras.io. Accessed: Oct. 07, 2024. [Online]. Available: https://keras.io/

14. Scikit-Learn, "scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation," Scikit-learn.org. Accessed: Oct. 07, 2024. [Online]. Available: https://scikit-learn.org/

15. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

16. Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach." 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

17. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." 2020. [Online]. Available: https://arxiv.org/abs/1910.01108

18. A. Conneau *et al.*, "Unsupervised Cross-lingual Representation Learning at Scale." 2020. [Online]. Available: https://arxiv.org/abs/1911.02116

19. H. Jain, A. Agarwal, K. Shridhar, and D. Kleyko, "End to End Binarized Neural Networks for Text Classification," *ArXiv*, vol. abs/2010.05223, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:222290714

20. Google, "Colaboratory – Google," research.google.com. Accessed: Oct. 07, 2024. [Online]. Available: https://research.google.com/colaboratory/faq.html

Надійшла до редакції 03.06.2025