УДК 004.056.53:004.85
DOI https://doi.org/10.32782/2521-6643-2026-2-72.23

Тазетдінов О. В., аспірант
Черкаського державного технологічного університету
ORCID: 0000-0003-4387-0500

Бабенко В. Г., доктор технічних наук, професор,
професор кафедри інформаційної безпеки та комп'ютерної інженерії
Черкаського державного технологічного університету
ORCID: 0000-0003-2039-2841

МЕТОД НАВЧАННЯ НА НЕЗБАЛАНСОВАНИХ ДАНИХ З АДАПТИВНИМ ЗВАЖУВАННЯМ ДЛЯ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО КОДУ

Стаття присвячена розробці методу навчання нейронних мереж на незбалансованих даних для задачі автоматичного виявлення вразливостей програмного коду. У реальних програмних проєктах частка вразливого коду зазвичай не перевищує 5–10 % від загального обсягу, що створює значний дисбаланс класів із коефіцієнтом від 10 до 100. Такий дисбаланс призводить до градієнтного домінування класу більшості та суттєво знижує здатність моделей розпізнавати вразливості. Проведено аналіз існуючих підходів до роботи з незбалансованими даними, зокрема методів передискретизації (Random Oversampling, SMOTE, Random Undersampling), статичного зважування класів та Focal Loss. Показано, що ці методи мають суттєві обмеження при застосуванні до програмного коду: методи передискретизації некоректно працюють із дискретними структурами коду, а статичні ваги не адаптуються до зміни складності прикладів у процесі навчання. Запропоновано метод адаптивного зважування, що поєднує три компоненти: зважування класів із параметричним контролем сили корекції на основі оберненої частоти класу, динамічне оцінювання складності прикладів на основі ентропії передбачення моделі та компонент навчального плану (curriculum learning), який забезпечує поступове введення складних прикладів через експоненціальну функцію темпу. Додатково розроблено стратегію адаптивного семплювання мінібатчів, що динамічно регулює співвідношення класів залежно від прогресу навчання та гарантує представництво класу меншості в кожному мінібатчі. Наведено математичну формалізацію всіх компонентів методу, включаючи механізми нормалізації ваг для збереження масштабу градієнтів та часткової компенсації зміщення семплювання через налаштовуваний параметр балансу. Результати дослідження можуть бути використані для підвищення ефективності систем автоматичного аналізу безпеки програмного забезпечення.

Ключові слова: *вразливості програмного коду, незбалансовані дані, адаптивне зважування, нейронні мережі, curriculum learning, Focal Loss, семплювання мінібатчів, кібербезпека.*

Tazetdinov O. V., Babenko V. H. Training method on imbalanced data with adaptive weighting for software code vulnerability detection

The article is devoted to the development of a neural network training method on imbalanced data for the task of automatic software code vulnerability detection. In real-world software projects, the proportion of vulnerable code typically does not exceed 5–10 % of the total codebase, resulting in a significant class imbalance with ratios ranging from 10 to 100. Such imbalance leads to gradient dominance of the majority class and substantially reduces the ability of models to detect vulnerabilities, which constitutes their primary objective. An analysis of existing approaches to handling imbalanced data has been conducted, including resampling methods (Random Oversampling, SMOTE, Random Undersampling), static class weighting, and Focal Loss. A comparative evaluation of these approaches is presented with respect to their applicability to source code analysis tasks. It has been demonstrated that resampling methods do not work correctly with the discrete structures of program code, static weights fail to adapt to changes in sample difficulty during the training process, and Focal Loss exhibits sensitivity to hyperparameters under conditions of extreme class imbalance. An adaptive weighting method is proposed that combines three components: class weighting with parametric control of correction strength based on the inverse class frequency raised to a tunable power, dynamic assessment of sample difficulty based on prediction uncertainty measured through classification entropy, and a curriculum learning component that ensures gradual introduction of complex samples through an exponential pacing function. Additionally, an adaptive mini-batch sampling strategy has been developed that dynamically adjusts class ratios depending on training progress and guarantees the representation of the minority class in each mini-batch, addressing the problem where standard random sampling results in most batches containing no minority class examples. A mathematical formalization of all method components is provided, including weight normalization mechanisms ensuring unit mean weight and partial compensation of sampling bias through a tunable balance parameter. The research results can be applied to improve the effectiveness of automated software security analysis systems.

Key words: *software code vulnerabilities, imbalanced data, adaptive weighting, neural networks, curriculum learning, Focal Loss, mini-batch sampling, cybersecurity.*

Постановка проблеми. При побудові систем автоматичного виявлення вразливостей виникає суттєва проблема незбалансованості навчальних даних. У реальних програмних проєктах частка вразливого коду зазвичай не перевищує 5–10 % від загального обсягу, а співвідношення може сягати 1:100 і більше. Такий дисбаланс призводить до домінування класу більшості при навчанні, що суттєво знижує здатність моделі виявляти вразливості – тобто виконувати свою основну задачу [1], [2].

Метод базується на динамічній зміні важливості різних прикладів та коригуванні функції втрат безпосередньо в процесі навчання [3].

Формально, проблема незбалансованості класів визначається через коефіцієнт дисбалансу:

$$IR = \frac{|D_{maj}|}{|D_{\min}|}, \tag{1}$$

де $|D_{maj}|$ – кількість прикладів класу більшості; $|D_{\min}|$ – кількість прикладів класу меншості. Для типових датасетів виявлення вразливостей IR знаходиться в діапазоні від 10 до 100, що класифікується як високий або екстремальний дисбаланс [3], [4], [5].

Вплив незбалансованості на навчання нейронних мереж проявляється через градієнтне домінування класу більшості:

$$\nabla L = \sum_{i \in D\_maj} \nabla L_i + \sum_{j \in D\_min} \nabla L_j . \tag{2}$$

Де∇L_i – градієнт втрат для одного конкретного прикладу i з класу більшості; ∇L_j – градієнт втрат для одного прикладу j з класу меншості.

При IR $\gg$ 1 перша сума домінує над другою, що призводить до зміщення моделі в бік класу більшості.

Очікуване значення градієнта для незбалансованого датасету:

$$E[\nabla L] = (1 - \pi) \cdot E[\nabla L | y = 0] + \pi \cdot E[\nabla L | y = 1], \tag{3}$$

де $\pi = |D_{\min}|/(|D_{maj}| + |D_{\min}|)$ – частка класу меншості.

При $\pi \ll 0.5$ внесок класу меншості у загальний градієнт стає незначним:

$$E[\nabla L] \approx E[\nabla L \mid y = 0]. \tag{4}$$

Аналіз останніх досліджень і публікацій. Існують три основні групи методів для розв'язання проблеми дисбалансу класів: методи, що змінюють дані (збільшення або зменшення вибірки, генерація штучних прикладів), алгоритмічні методи (введення вагових коефіцієнтів, регулювання порогів, зміна функції втрат) та їх комбінації [6], [7]. Порівняння цих підходів наведено в табл. 1.

Таблиця 1

Порівняння підходів до роботи з незбалансованими даними

Підхід	Переваги	Недоліки	Застосовність для коду
Random Oversampling	Простота реалізації	Перенавчання	Низька
SMOTE	Генерація нових прикладів	Некоректна для дискретних даних	Обмежена
Random Undersampling	Зменшення часу навчання	Втрата інформації	Середня
Class Weighting	Не змінює дані	Статичні ваги	Висока
Focal Loss	Адаптивне зважування	Чутливість до гіперпараметрів	Висока
Запропонований метод	Динамічна адаптація	Обчислювальні витрати	Дуже висока

Джерело: розроблено авторами

Стандартний підхід зважування класів використовує статичні ваги, обернено пропорційні до частоти класу:

$$w_c = N/(C \cdot N_c), \tag{5}$$

де N – загальна кількість прикладів; C – кількість класів; N_c – кількість прикладів класу.

Зважена функція крос-ентропії:

$$L_{WCE} = -\sum_{c=1}^{C} w_c \cdot y_c \cdot \log(p_c). \tag{6}$$

Focal Loss модифікує стандартну крос-ентропію шляхом додавання модулюючого фактора:

$$L_{FL} = -\alpha_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t), \tag{7}$$

де p_t = p, якщо $y = 1$, та $p_t = 1 - p$, якщо $y = 0$; α_t – ваговий параметр класу; γ – фокусуючий параметр (зазвичай $\gamma = 2$).

Модулюючий фактор $(1 - p_t)^\gamma$ зменшує внесок легко класифікованих прикладів:

$$(1 - p_t)^\gamma \to 0 \text{ якщо } p_t \to 1; \to 1 \text{ якщо } p_t \to 0. \quad (8)$$

Водночас, Focal Loss має певні недоліки, особливо коли існує дуже велика різниця в кількості прикладів різних класів, а також не бере до уваги особливості програмного коду [8], [9], [10].

Метою дослідження є розробка методу адаптивного зважування навчальних прикладів для підвищення ефективності навчання нейронних мереж на незбалансованих даних у задачі автоматичного виявлення вразливостей програмного коду. Метод має забезпечити динамічне коригування впливу окремих прикладів на процес навчання з урахуванням належності до класу, складності класифікації та прогресу навчання моделі.

Виклад основного матеріалу. Запропонований підхід передбачає зміну важливості навчальних прикладів у процесі навчання. Це здійснюється на основі трьох критеріїв: ступеня складності класифікації прикладу, його інформативності для навчання та поточного рівня навченості моделі.

Динамічна вага обчислюється як композиція трьох компонентів:

$$w_i(t) = w_i^{class} \cdot w_i^{difficulty}(t) \cdot w_i^{curriculum}(t). \quad (9)$$

Компонент зважування класу:

$$w_i^{class} = \left(N / \left(C \cdot N\_\{y\_i\}\right)\right)^\beta, \quad (10)$$

де $\beta \in [0, 1]$ – параметр, що контролює силу зважування класу. При $\beta = 0$ зважування відсутнє, при $\beta = 1$ – повне обернено пропорційне зважування.

Компонент складності прикладу базується на невизначеності передбачення:

$$w_i^{difficulty}(t) = \left(1 - \left|p_i(t) - 0.5\right| \cdot 2\right)^\gamma + \varepsilon, \quad (11)$$

де $p_i(t)$ – передбачена ймовірність на епосі t; γ – параметр фокусування; ε – мала константа для числової стабільності.

Аналогічно визначається вираз:

$$w_i^{difficulty}(t) = \left(H_i(t) / H_{\max}\right)^\gamma, \quad (12)$$

де $H_{\max} = \log(2)$ – максимальна ентропія для бінарної класифікації.

Компонент навчального плану (curriculum) забезпечує поступове введення складних прикладів:

$$w_i^{curriculum}(t) = \min\left(1, \lambda(t) \cdot s_i\right), \quad (13)$$

де s_i – нормалізований показник складності прикладу; $\lambda(t)$ – функція темпу, що зростає з часом.

Функція темпу визначається як:

$$\lambda(t) = \lambda_0 + (\lambda_{\max} - \lambda_0) \cdot (1 - \exp(-t/\tau)), \quad (14)$$

де λ_0 – початковий темп; $\lambda_{\max}$ – максимальний темп; τ – константа часу.

Загальну архітектуру методу адаптивного зважування наведено на рис. 1. Параметри адаптивної функції втрат подано в табл. 2.

Таблиця 2

Параметри адаптивної функції втрат

Параметр	Типове значення	Опис
β (class weight)	0.75	Сила зважування класів
γ (focusing)	2.0	Параметр фокусування
ε (stability)	1e-8	Числова стабільність
λ_0 (initial pace)	0.3	Початковий темп curriculum
$\lambda_{\max}$ (max pace)	1.0	Максимальний темп
τ (time constant)	10	Швидкість зростання темпу
$\alpha_1, \alpha_2, \alpha_3$	0.3, 0.4, 0.3	Ваги компонентів складності

Джерело: розроблено авторами

Ефективний розмір вибірки з урахуванням ваг:

$$N_{eff} = \left(\sum w_i\right)^2 \Big/ \sum w_i^2 . \tag{15}$$

Для забезпечення стабільності навчання ваги нормалізуються:

$$\tilde{w}_i(t) = w_i(t) \cdot N \Big/ \sum_j w_j(t). \tag{16}$$

Це гарантує, що середня вага залишається рівною 1, зберігаючи масштаб градієнтів.
Візуалізацію динаміки ваг протягом навчання наведено на рисунку 1.

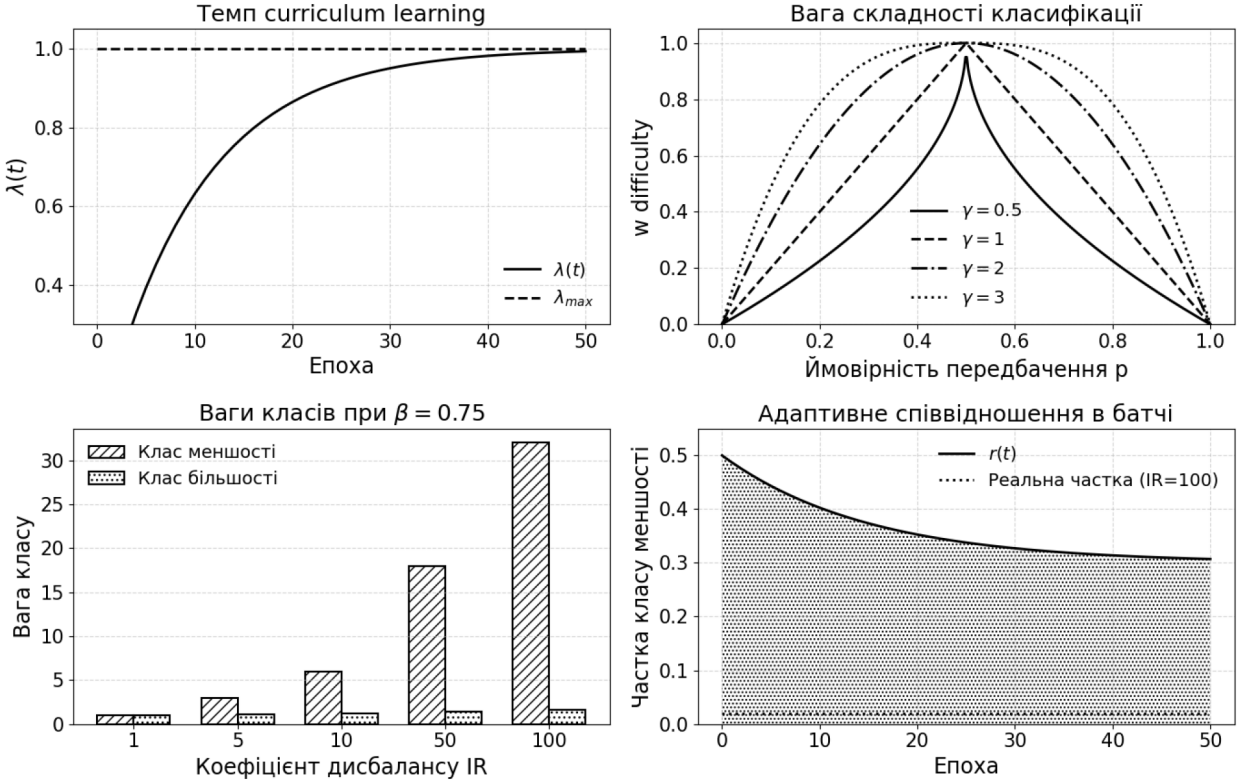

Рис. 1. Візуалізація динаміки ваг протягом навчання

Джерело: розроблено авторами

Окрім адаптивного зважування, запропонований метод включає стратегію збалансованого семплювання для формування мінібатчів, що додатково покращує збіжність навчання.

Стандартне випадкове семплювання призводить до того, що в мінібатчі розмірів B очікувана кількість прикладів класу меншості:

$$E[n_{\min}] = B \cdot \pi \approx B/IR. \tag{17}$$

При $IR = 100$ та $B = 32$: $E[n_{\min}] \approx 0.32$, тобто в більшості мінібатчів клас меншості взагалі не представлений.

Комбінована вага з урахуванням семплювання та адаптивного зважування:

$$w_i^{total}(t) = w_i^{sampling} \cdot w_i(t). \tag{18}$$

Проте, повна компенсація не є оптимальною стратегією для виявлення вразливостей, оскільки основна ціль полягає у покращенні розпізнавання меншого класу об'єктів. З цієї причини використовується часткова компенсація. Формалізацію наведено у виразі:

$$w_i^{total}(t) = \left(w\_i \wedge \{sampling\}\right)^{(1-\delta)} \cdot w_i(t), \tag{19}$$

де $\delta \in [0, 1]$ – параметр балансу ($\delta = 0$ – повна компенсація, $\delta = 1$ – без компенсації). Порівняння стратегій семплювання наведено в табл. 3.

Таблиця 3

Порівняння стратегій семплювання

Стратегія	$E[n_{min}]$ в батчі	Дисперсія	Зміщення оцінки
Випадкове семплювання	B/IR	Висока	Немає
Збалансоване семплювання	B/2	Низька	Є (коригується)
Стратифіковане семплювання	B·π	Середня	Немає
Адаптивне (запропоноване)	Динамічно	Низька	Контрольоване

Джерело: розроблено авторами

Адаптивне семплювання динамічно регулює співвідношення класів у батчі залежно від прогресу навчання:

$$r(t) = r_0 + (r_{target} - r_0) \cdot (1 - \exp(-t/\tau_r)), \quad (20)$$

де $r(t)$ – частка класу меншості в батчі на епосі t; r_0 – початкове співвідношення; r_{target} – цільове співвідношення; τ_r – константа часу.

У таблиці 4 представлені результати порівняння VulnHybrid з іншими моделями на основі датасету Devign. Експеримент повторювався п’ять разів, а в таблиці показано середні значення та їхні стандартні відхилення.

Таблиця 4

Результати на датасеті Devign

Модель	F1	Precision	Recall	AUC-ROC	MCC
SySeVR	0.542 ± 0.018	0.513 ± 0.021	0.578 ± 0.024	0.698 ± 0.015	0.289 ± 0.022
Devign	0.612 ± 0.015	0.583 ± 0.019	0.647 ± 0.022	0.751 ± 0.012	0.358 ± 0.019
IVDetect	0.634 ± 0.014	0.602 ± 0.017	0.673 ± 0.020	0.768 ± 0.011	0.384 ± 0.018
ReVeal	0.658 ± 0.013	0.627 ± 0.016	0.694 ± 0.019	0.782 ± 0.010	0.412 ± 0.017
CodeBERT	0.697 ± 0.011	0.668 ± 0.014	0.731 ± 0.016	0.814 ± 0.009	0.462 ± 0.015
LineVul	0.714 ± 0.010	0.684 ± 0.013	0.748 ± 0.015	0.826 ± 0.008	0.487 ± 0.014
VulnHybrid	0.738 ± 0.009	0.709 ± 0.012	0.771 ± 0.014	0.842 ± 0.007	0.521 ± 0.013

Джерело: розроблено авторами

VulnHybrid показала найкращі результати за всіма критеріями. Зокрема, якщо порівнювати з LineVul, то VulnHybrid покращила F1-score на 2.6 процентних пункти, тобто на 3.6 % відносно. Порівняння результатів для всіх моделей представлено на рисунку 2.

Для підтвердження покращення застосовувався парний t-тест. Згідно з його результатами, усі відмінності VulnHybrid та LineVul мають статистичну значущість на рівні $p < 0.01$.

Аналіз результатів за групами моделей показує наступне. RNN-підхід (SySeVR) значно гірший за інші, що підтверджує обмеженість послідовного представлення під час аналізу коду. GNN-підходи (Devign,

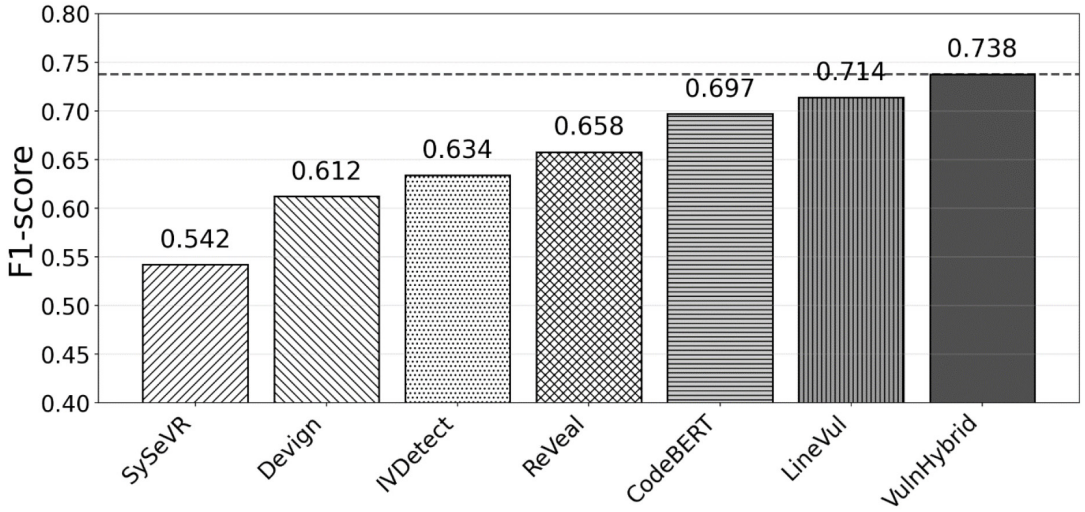

Рис. 2. Порівняння моделей на основі F1-score на датасеті Devign

Джерело: розроблено авторами

IVDetect, ReVeal) показують помірні результати, які поступово покращуються з додаванням механізмів уваги. Трансформерні підходи (CodeBERT, LineVul) є кращими серед базових моделей. Гібридний підхід VulnHybrid перевершує як графові, так і трансформерні методи [11], [12], [13].

Результати оцінки на датасеті Big-Vul із природним дисбалансом класів (лише 5.7 % вразливих функцій) наведено в таблиці 5.

Таблиця 5

Результати на датасеті Big-Vul

Модель	F1	Precision	Recall	F1 (minority)
Devign	0.487 ± 0.024	0.412 ± 0.028	0.598 ± 0.031	0.234 ± 0.029
ReVeal	0.523 ± 0.021	0.451 ± 0.025	0.624 ± 0.028	0.271 ± 0.026
CodeBERT	0.568 ± 0.018	0.498 ± 0.022	0.663 ± 0.025	0.312 ± 0.024
LineVul	0.591 ± 0.016	0.521 ± 0.020	0.687 ± 0.023	0.338 ± 0.022
VulnHybrid (без адапт.)	0.602 ± 0.015	0.534 ± 0.019	0.694 ± 0.022	0.351 ± 0.021
VulnHybrid (з адапт.)	0.647 ± 0.014	0.572 ± 0.017	0.748 ± 0.020	0.418 ± 0.019

Джерело: розроблено авторами

Адаптивне зважування позитивно впливає на результати при роботі з незбалансованими даними. F1-score для класу вразливого коду зросла з 0.351 до 0.418, тобто на 19.1 %. Ефективність адаптивного зважування на датасеті Big-Vul показано на рисунку 3.

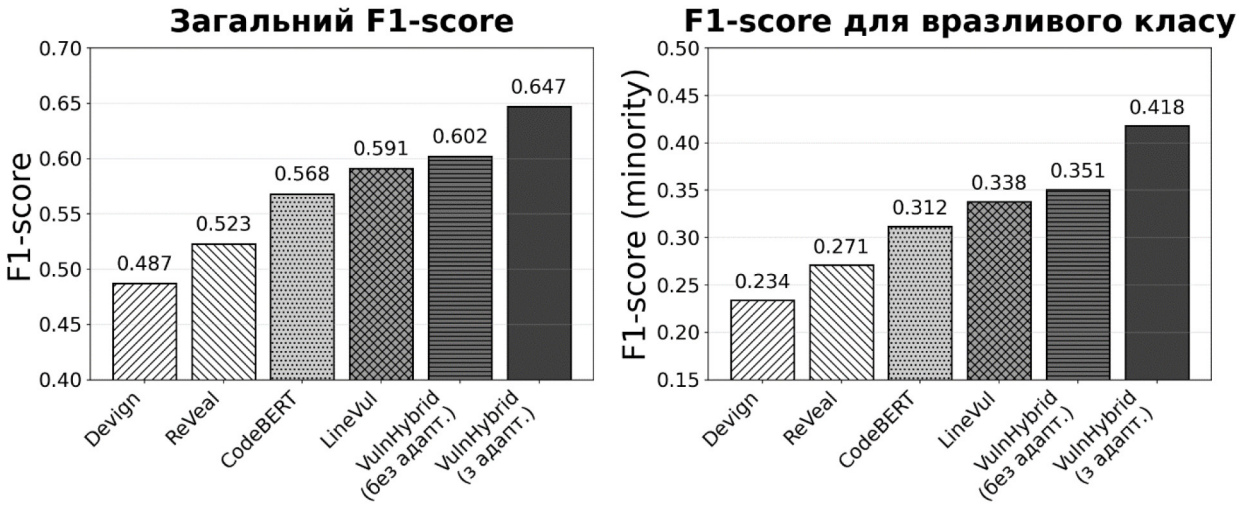

Рис. 3. Ефективність адаптивного зважування на датасеті Big-Vul

Джерело: розроблено авторами

Якщо порівнювати VulnHybrid з адаптивним зважуванням та без нього, то можна помітити, що запропонований метод ефективно працює з незбалансованими даними: загальний F1-score покращився на 7.5 %, а F1-score для класу меншості – на 19.1 % [14], [15], [16], [17].

Результати класифікації за типами CWE на основі датасету Big-Vul наведено в таблиці 6 [18], [19], [20], [21].

Результати багатокласової класифікації за типами CWE представлено на рисунку 4.

Згідно з результатами для різних груп вразливостей, були зроблені такі висновки. Для групи «Пам’ять» (CWE-119, CWE-125, CWE-787) середній F1 = 0.559. Модель краще розпізнає переповнення буфера (CWE-119), але гірше виявляє запис за межі (CWE-787). Для групи «Покажчики» (CWE-476, CWE-416) середній F1 = 0.553. Розіменування нульового покажчика добре виявляється завдяки чіткій схемі. Використання після звільнення виявляється складніше через необхідність відстежувати життєвий цикл пам’яті. Для групи «Арифметика» (CWE-190, CWE-369) середній F1 = 0.571. Ділення на нуль розпізнається краще через просту схему.

Висновки. У статті розроблено метод навчання нейронних мереж на незбалансованих даних для задачі автоматичного виявлення вразливостей програмного коду. Основні результати дослідження полягають у наступному.

Запропоновано метод адаптивного зважування навчальних прикладів, що поєднує три компоненти: зважування класів із параметричним контролем сили корекції, динамічне оцінювання складності прикладів

Таблиця 6

Результати багатокласової класифікації за типами CWE

CWE	Тип	Precision	Recall	F1
CWE-119	Buffer Overflow	0.623 ± 0.019	0.587 ± 0.022	0.604 ± 0.020
CWE-125	Out-of-bounds Read	0.578 ± 0.023	0.542 ± 0.026	0.559 ± 0.024
CWE-787	Out-of-bounds Write	0.534 ± 0.027	0.498 ± 0.030	0.515 ± 0.028
CWE-476	NULL Pointer Deref	0.651 ± 0.018	0.618 ± 0.021	0.634 ± 0.019
CWE-416	Use After Free	0.489 ± 0.031	0.456 ± 0.034	0.472 ± 0.032
CWE-190	Integer Overflow	0.567 ± 0.024	0.534 ± 0.027	0.550 ± 0.025
CWE-369	Divide by Zero	0.612 ± 0.026	0.573 ± 0.029	0.592 ± 0.027
Safe	Безпечний код	0.847 ± 0.011	0.876 ± 0.013	0.861 ± 0.012
Macro-avg		0.613 ± 0.015	0.585 ± 0.017	0.598 ± 0.016

Джерело: розроблено авторами

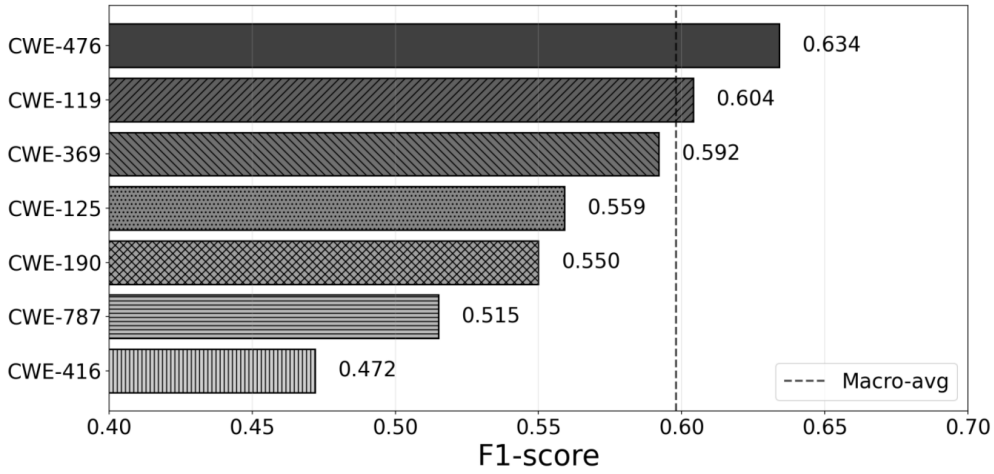

Рис. 4. F1-score за типами вразливостей CWE

Джерело: розроблено авторами

на основі ентропії передбачення та компонент навчального плану (curriculum learning) з експоненціальною функцією темпу. На відміну від існуючих підходів, таких як статичне зважування класів або Focal Loss, запропонований метод враховує зміну складності прикладів у процесі навчання та забезпечує поступове введення складних зразків.

Розроблено стратегію адаптивного семплювання мінібатчів, що динамічно регулює співвідношення класів залежно від прогресу навчання. Показано, що при стандартному випадковому семплюванні з коефіцієнтом дисбалансу $IR = 100$ та розміром батчу $B = 32$ клас меншості в більшості мінібатчів не представлений, тоді як запропонована стратегія гарантує його присутність у кожному мінібатчі.

Наведено повну математичну формалізацію всіх компонентів методу, включаючи механізми нормалізації ваг та часткової компенсації зміщення семплювання, що забезпечує контрольоване зміщення на користь класу меншості.

Експериментальна оцінка на датасетах Devign та Big-Vul підтвердила ефективність запропонованого методу. На датасеті Devign гібридна модель VulnHybrid із адаптивним зважуванням досягла F1-score 0.738, що на 3.6 % перевищує найкращий базовий метод LineVul. На датасеті Big-Vul із природним дисбалансом класів адаптивне зважування забезпечило покращення F1-score для класу вразливого коду на 19.1 %. Багатокласова класифікація за типами CWE продемонструвала macro-avg F1 = 0.598, із найкращими результатами для CWE-476 (NULL Pointer Dereference, F1 = 0.634) та CWE-119 (Buffer Overflow, F1 = 0.604). Перспективи подальших досліджень полягають у розширенні методу на інші мови програмування, оптимізації гіперпараметрів адаптивного зважування та інтеграції з методами пояснюваного штучного інтелекту для інтерпретації результатів виявлення вразливостей.

Список використаних джерел:

1. Ni C., Shen L., Yang X., Zhu Y., Wang S. MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representations. Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR 2024). Lisbon, Portugal, April 15–16, 2024. P. 738–742. DOI: https://doi.org/10.1145/3643991.3644886

2. Ni C., Wang S., Ren J., Chen S., Nguyen T. VULGEN: Realistic Vulnerability Generation Via Pattern Mining and Deep Learning. Proc. 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023). 2023. P. 346–358. DOI: https://doi.org/10.1109/ICSE48619.2023.00211

3. Zheng Y., Pujar S., Lewis B., Buratti L., Epstein E., Yang B., Laredo J., Morari A., Su Z. D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2021). Madrid, Spain, May 25–28, 2021. P. 111–120. DOI: https://doi.org/10.1109/ICSE-SEIP52600.2021.00020

4. Liu S., Wu B., Xie S., Meng G., Liu Y. ContraBERT: Enhancing Code Pre-trained Models via Contrastive Learning. Proc. 45th IEEE/ACM International Conference on Software Engineering (ICSE 2023). 2023. P. 2476–2487. DOI: https://doi.org/10.1109/ICSE48619.2023.00207

5. FIRST.org. Common Vulnerability Scoring System v3.1: Specification Document. Forum of Incident Response and Security Teams (FIRST.org). 2019. URL: https://www.first.org/cvss/v3.1/specification-document

6. Boot H., Reik D., Witte H. National Vulnerability Database. National Institute of Standards and Technology (NIST). 2013. Special Publication 800-51. URL: https://nvd.nist.gov

7. Lin G., Wen S., Han Q.-L., Zhang J., Xiang Y. Software Vulnerability Detection Using Deep Neural Networks: A Survey. Proceedings of the IEEE. 2020. Vol. 108. No. 10. P. 1825–1848. DOI: https://doi.org/10.1109/JPROC.2020.2993293

8. Ghaffarian S.M., Shahriari H.R. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. ACM Computing Surveys. 2017. Vol. 50. No. 4. Article 56. P. 1–36. DOI: https://doi.org/10.1145/3092566

9. Wagner D. A., Foster J. S., Brewer E. A., Aiken A. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. Proc. 2000 Network and Distributed System Security Symposium (NDSS 2000). 2000. P. 3–17.

10. Chawla N. V., Bowyer K. W., Hall L. O., Kegelmeyer W. P. SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research. 2002. Vol. 16. P. 321–357. DOI: https://doi.org/10.1613/jair.953

11. Wen S.-C., Wang S., Gao K., Wang S., Liu Y., Gu C. When Less is Enough: Positive and Unlabeled Learning Model for Vulnerability Detection. Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023). Luxembourg, September 11–15, 2023. P. 345–357. DOI: https://doi.org/10.1109/ASE56229.2023.00144

12. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language Models are Unsupervised Multitask Learners. OpenAI Blog. 2019. Vol. 1. P. 9.

13. Cao S., Sun X., Bo L., Wei Y., Li B. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. Information and Software Technology. 2021. Vol. 136. P. 106576. DOI: https://doi.org/10.1016/j.infsof.2021.106576

14. Cao S., Sun X., Wu X., Lo D., Bo L., Li B., Liu X., Lin X., Liu W. Snopy: Bridging Sample Denoising with Causal Graph Learning for Effective Vulnerability Detection. Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024). Sacramento, CA, USA, October 27–November 1, 2024. P. 606–618. DOI: https://doi.org/10.1145/3691620.3695057

15. Harer J. A., Kim L. Y., Hamilton R. L., Lazovich T., Russell R. L. et al. Automated Software Vulnerability Detection with Machine Learning. arXiv preprint. 2018. arXiv:1803.04497. DOI: https://doi.org/10.48550/arXiv.1803.04497

16. Russell R. L., Kim L. Y., Hamilton L. H., Lazovich T., Harer J. A., Ozdemir O., Ellingwood P. M., McConley M. W. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA 2018). Orlando, FL, USA, December 17–20, 2018. P. 757–762. DOI: https://doi.org/10.1109/ICMLA.2018.00120

17. Mikolov T., Sutskever I., Chen K., Corrado G. S., Dean J. Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems 26 (NIPS 2013). Lake Tahoe, NV, USA, December 5–8, 2013. P. 3111–3119. DOI: https://doi.org/10.48550/arXiv.1310.4546

18. Nguyen G.H., Nguyen V., Nguyen T., Nguyen T.N. MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities. arXiv preprint. 2022. arXiv:2208.13252. DOI: https://doi.org/10.48550/arXiv.2208.13252

19. Zeng P., Lin G., Pan L., Tai Y., Zhang J. Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey. IEEE Access. 2020. Vol. 8. P. 197158–197172. DOI: https://doi.org/10.1109/ACCESS.2020.3034766

20. Hin D., Kan A., Chen H., Babar M. A. LineVD: Statement-Level Vulnerability Detection Using Graph Neural Networks. Proceedings of the 19th IEEE/ACM International Conference on Mining Software Repositories (MSR 2022). Pittsburgh, PA, USA, May 23–24, 2022. P. 596–607. DOI: https://doi.org/10.1145/3524842.3527949

21. Li Y., Wang S., Nguyen T.N. Vulnerability Detection with Fine-Grained Interpretations. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). Athens, Greece, August 23–28, 2021. P. 292–303. DOI: https://doi.org/10.1145/3468264.3468597

References:

1. Ni, C., Shen, L., Yang, X., Zhu, Y., & Wang, S. (2024). MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representations. Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR 2024) (pp. 738–742). https://doi.org/10.1145/3643991.3644886

2. Ni, C., Wang, S., Ren, J., Chen, S., & Nguyen, T. (2023). VULGEN: Realistic Vulnerability Generation Via Pattern Mining and Deep Learning. Proc. ESEC/FSE 2023 (pp. 346–358). https://doi.org/10.1109/ICSE48619.2023.00211

3. Zheng, Y., Pujar, S., Lewis, B., Buratti, L., Epstein, E., Yang, B., Laredo, J., Morari, A., & Su, Z. (2021). D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. Proceedings of ICSE-SEIP 2021 (pp. 111–120). https://doi.org/10.1109/ICSE-SEIP52600.2021.00020

4. Liu, S., Wu, B., Xie, S., Meng, G., & Liu, Y. (2023). ContraBERT: Enhancing Code Pre-trained Models via Contrastive Learning. Proc. ICSE 2023 (pp. 2476–2487). https://doi.org/10.1109/ICSE48619.2023.00207

5. FIRST.org. (2019). Common Vulnerability Scoring System v3.1: Specification Document. Retrieved from https://www.first.org/cvss/v3.1/specification-document

6. Boot, H., Reik, D., & Witte, H. (2013). National Vulnerability Database (Special Publication 800-51). National Institute of Standards and Technology. Retrieved from https://nvd.nist.gov

7. Lin, G., Wen, S., Han, Q.-L., Zhang, J., & Xiang, Y. (2020). Software Vulnerability Detection Using Deep Neural Networks: A Survey. Proceedings of the IEEE, 108(10), 1825–1848. https://doi.org/10.1109/JPROC.2020.2993293

8. Ghaffarian, S. M., & Shahriari, H. R. (2017). Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. ACM Computing Surveys, 50(4), Article 56, 1–36. https://doi.org/10.1145/3092566

9. Wagner, D. A., Foster, J. S., Brewer, E. A., & Aiken, A. (2000). A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. Proc. NDSS 2000 (pp. 3–17).

10. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953

11. Wen, S.-C., Wang, S., Gao, K., Wang, S., Liu, Y., & Gu, C. (2023). When Less is Enough: Positive and Unlabeled Learning Model for Vulnerability Detection. Proceedings of ASE 2023 (pp. 345–357). https://doi.org/10.1109/ASE56229.2023.00144

12. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog, 1, 9.

13. Cao, S., Sun, X., Bo, L., Wei, Y., & Li, B. (2021). BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. Information and Software Technology, 136, 106576. https://doi.org/10.1016/j.infsof.2021.106576

14. Cao, S., Sun, X., Wu, X., Lo, D., Bo, L., Li, B., Liu, X., Lin, X., & Liu, W. (2024). Snopy: Bridging Sample Denoising with Causal Graph Learning for Effective Vulnerability Detection. Proceedings of ASE 2024 (pp. 606–618). https://doi.org/10.1145/3691620.3695057

15. Harer, J. A., Kim, L. Y., Hamilton, R. L., Lazovich, T., Russell, R. L., et al. (2018). Automated Software Vulnerability Detection with Machine Learning. arXiv:1803.04497. https://doi.org/10.48550/arXiv.1803.04497

16. Russell, R. L., Kim, L. Y., Hamilton, L. H., Lazovich, T., Harer, J. A., Ozdemir, O., Ellingwood, P. M., & McConley, M. W. (2018). Automated Vulnerability Detection in Source Code Using Deep Representation Learning. Proceedings of ICMLA 2018 (pp. 757–762). https://doi.org/10.1109/ICMLA.2018.00120

17. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in NIPS 26 (pp. 3111–3119). https://doi.org/10.48550/arXiv.1310.4546

18. Nguyen, G.H., Nguyen, V., Nguyen, T., & Nguyen, T.N. (2022). MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities. arXiv:2208.13252. https://doi.org/10.48550/arXiv.2208.13252

19. Zeng, P., Lin, G., Pan, L., Tai, Y., & Zhang, J. (2020). Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey. IEEE Access, 8, 197158–197172. https://doi.org/10.1109/ACCESS.2020.3034766

20. Hin, D., Kan, A., Chen, H., & Babar, M. A. (2022). LineVD: Statement-Level Vulnerability Detection Using Graph Neural Networks. Proceedings of MSR 2022 (pp. 596–607). https://doi.org/10.1145/3524842.3527949

21. Li, Y., Wang, S., & Nguyen, T. N. (2021). Vulnerability Detection with Fine-Grained Interpretations. Proceedings of ESEC/FSE 2021 (pp. 292–303). https://doi.org/10.1145/3468264.3468597

Дата першого надходження статті до видання: 26.03.2026
Дата прийняття статті до друку після рецензування: 19.04.2026
Дата публікації (оприлюднення) статті: 30.05.2026