УДК 004.85:004.41

Штукований М. О., магістрант
Духновська К. К., к.т.н., доцент
Київський національний університет імені Тараса Шевченка

МАШИННЕ НАВЧАННЯ ДЛЯ ВИЯВЛЕННЯ СЕМАНТИЧНИХ ПОМИЛОК У КОДІ

Розвиток сучасного програмного забезпечення (ПЗ) супроводжується зростанням кількості семантичних і логічних помилок, що становлять серйозну загрозу надійності систем. На відміну від синтаксичних дефектів, які ефективно виявляються традиційними лінтерами, семантичні помилки зазвичай проявляються лише під час виконання програми та вимагають розуміння контексту виконання коду. Зростання складності ПЗ обумовлює потребу в автоматизованих засобах на базі методів машинного навчання (МН), зокрема великих мовних моделей (LLM — *Large Language Models*), Transformer-архітектур та графових нейронних мереж (GNN — *Graph Neural Networks*), здатних аналізувати семантику коду в реальних умовах CI/CD-конвеєрів.

Огляд підходів. S. B. Hossain та ін. (2024) [1] запропонували фреймворк Toggle, у якому LLM виконує двоетапну обробку: локалізацію дефекту на рівні токенів та його подальше виправлення за допомогою оптимізованих промптів. На датасеті Defects4J точність Top-1 підвищилася на 2,3–54,4 % порівняно з базовими методами МН. A. Poniszewska-Marańda (2024) [2] дослідила використання архітектур Transformer та GNN для автоматичного виявлення семантичних дефектів, акцентувавши увагу на складнощах їхньої корекції через контекстну неоднозначність у великомасштабних проєктах.

S. Ullah та ін. (2023) [3] у межах проєкту SecLLMHolmes оцінили 8 моделей LLM на 228 сценаріях помилок, критичних для безпеки. Було виявлено системні обмеження в логічному міркуванні моделей, що обґрунтовує потребу в їх тонкому налаштуванні (*fine-tuning*) та застосуванні гібридних підходів. M. N. Rafi та ін. (2024) [4] розробили GNN із представленням коду у вигляді дерева синтаксичного аналізу (AST), проаналізувавши 6 052 дефекти з 307 репозиторіїв GitHub. K. Qayyum та ін. (2025) [5] досягли 64,7 % точності корекції семантичних помилок у мові Verilog завдяки методології RAG-LLM, що перевищило показники SOTA на 53,4 %.

Результати аналізу. Порівняння архітектур показало, що LLM демонструють перевагу у генеративному міркуванні [1, 5], тоді як GNN ефективніші у структурному аналізі великих кодових баз [4]. Системним обмеженням LLM залишається нестабільність логічного виведення [3], що підтверджує необхідність гібридизації (табл. 1).

Таблиця 1 – Порівняння моделей виявлення семантичних помилок

Модель	Тип помилок	Точність (Top-1)	Датасет	Перевага
Toggle LLM [1]	Семантичні	+2,3–54,4 %	Defects4J	Токен-рівень
AST-GNN [4]	Логічні	SOTA vs покриття	6052 дефекти	AST-контекст
SecLLM Holmes [3]	Безпека	Обмежена	228 сценаріїв	Fine-tuning
RAG-LLM Verilog [5]	Функціональні	64,7 %	HDL-баги	RAG-корекція

Напрями подальших досліджень. Отримані результати свідчать, що найбільш перспективним напрямом є гібридизація LLM і GNN з механізмом RAG: LLM забезпечує генеративне міркування, GNN — структурний аналіз AST, а RAG — контекстуалізацію на доменних наборах дефектів. Пріоритетними напрямами подальших досліджень є: тонке налаштування моделей на пов'язаних з безпекою кодових базах, оцінювання гібридних архітектур на стандартних бенчмарках (Defects4J, CodeXGLUE) та дослідження застосовності підходів у низькорівневих мовах (Verilog, C/C++).

Висновки. LLM і GNN суттєво перевершують традиційні статичні аналізатори у виявленні семантичних помилок, проте кожна архітектура має власні обмеження: LLM — у стабільності логічного виведення, GNN — у генеративних можливостях. Встановлено, що гібридні підходи (RAG-LLM + AST-GNN) забезпечують найвищу ефективність: точність до 64,7 % у спеціалізованих доменах. Також встановлено, що ефективна система виявлення семантичних помилок має поєднувати обидві архітектури з механізмом RAG-індексації доменних дефектів. В перспективах подальших досліджень є оцінювання гібридних архітектур на розширених бенчмарках, тонке налаштування на доменних наборах даних та аналіз застосовності підходів у різних мовах програмування.

Список використаних джерел

1. Hossain S. B. A Deep Dive into Large Language Models for Automated Bug Localization and Repair / S. B. Hossain, N. Jiang, Q. Zhou [et al.] // arXiv. – 2024. – arXiv:2404.11595v3 [cs.SE]. – URL: https://arxiv.org/abs/2404.11595.

2. Poniszewska-Marańda A. Automatic Detection and Correction of Code Errors Applying Machine Learning Algorithms / A. Poniszewska-Marańda // Proc. of the 2024 Conf. on Innovation in Software Engineering (CISE). – ACM, 2024. – DOI: 10.1145/3661167.3661198.

3. Ullah S. LLMs Cannot Reliably Identify and Reason About Security-Related Bugs / S. Ullah [et al.] // arXiv. – 2023. – arXiv:2312.12575v2 [cs.SE]. – URL: https://arxiv.org/abs/2312.12575.

4. Rafi M. N. Towards Better Graph Neural Network-based Fault Localization Through Enhanced Code Representation / M. N. Rafi [et al.] // arXiv. – 2024. – arXiv:2404.04496 [cs.SE]. – URL: https://arxiv.org/abs/2404.04496.

5. Qayyum K. LLM-Assisted Bug Identification and Correction for Verilog Hardware Description Language / K. Qayyum [et al.] // Proc. of the ACM/IEEE Design Automation Conf. – 2025. – DOI: 10.1145/3733237.