Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: "Застосування LLM в галузі кібербезпеки"

Виконав: студент IV курсу, групи СБ-42

Спеціальності:
125 "Кібербезпека"
(шифр і назва напряму підготовки, спеціальності)

Павлат О. В.
підпис (прізвище та ініціали)

Керівник Стадник М. А.
підпис (прізвище та ініціали)

Нормоконтроль Дроздова Т.В.
підпис (прізвище та ініціали)

Завідувач кафедри Загородна Н.В.
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
“__” _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Павлат Олександру Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи Застосування LLM в галузі кібербезпеки

Керівник роботи Стадник Марія Андріївна, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від “02“ 05 2025 року № 4/7-361

2. Термін подання студентом завершеної роботи 20.06.2025

3. Вихідні дані до роботи Набір даних Phishing Email Dataset

4. Зміст роботи (перелік питань, які потрібно розробити)

1. Теоретичні основи LLM. 1.1. Поняття великомасштабних мовних моделей (LLM). 1.2 Основні архітектури LLM: GPT, BERT, T5. 1.3 Застосування ШІ в галузі кібербезпеки. 2. Застосування LLM у кібербезпеці. 2.1 Виявлення фішингових атак з використанням LLM. 2.2 Аналіз логів та виявлення аномалій. 2.3 Генерація політик безпеки та документації. 2.4 Застосування LLM в тестуванні на проникнення. 2.5 Аналіз вразливостей та інтелектуальний моніторинг. 2.6 Автоматизоване виправлення програмного коду. 2.7 Інші застосування LLM у кібербезпеці. 3. Застосування LLM для виявлення фішингових листів. 3.1 Побудова LLM. 3.2 Опис методології та набору даних. 3.3 Результати роботи моделі. 3.4 Порівняння з класичними методами ідентифікації фішингових листів. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Заходи, що покращують умови праці оператора.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Мариненко С. Ю., к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 29.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.01 – 02.02	Виконано
2.	Підбір джерел для огляду LLM	03.02 – 07.02	Виконано
3.	Опрацювання джерел в галузі дослідження	08.02 – 18.02	Виконано
4.	Проведення аналізу застосування LLM у кібербезпеці.	19.02 – 23.02	Виконано
5.	Проведення аналізу архітектур LLM.	24.02 – 03.03	Виконано
6.	Опрацювання набору даних.	04.03 – 11.03	Виконано
7.	Написання програмної реалізації алгоритму виявлення фішингових листів з використанням LLM.	12.03 – 24.03	Виконано
8.	Аналіз результатів програмної реалізації.	25.03 – 06.04	Виконано
9.	Оформлення першого розділу.	07.04 – 17.04	Виконано
10.	Оформлення другого розділу.	18.04 – 27.04	Виконано
11.	Оформлення третього розділу.	28.04 – 17.05	Виконано
12.	Виконання завдання до підрозділу “Безпека життєдіяльності, основи охорони праці”	18.05 – 22.05	Виконано
13.	Оформлення кваліфікаційної роботи	23.05 – 01.06	Виконано
14.	Нормоконтроль	02.06 – 15.06	Виконано
15.	Перевірка на плагіат	16.06 – 20.06	Виконано
16.	Попередній захист кваліфікаційної роботи	23.06 – 24.06	Виконано
17.	Захист кваліфікаційної роботи	26.06.2025	Виконано

Студент _____ Павлат О. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стадник М. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Застосування LLM в галузі кібербезпеки // Кваліфікаційна робота ОР “Бакалавр” // Павлат Олександр Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБ-42 // Тернопіль, 2025 // С. 92, рис. – 16, табл. – 1, кресл. – - , додат. – 1.

Ключові слова: великі мовні моделі, кібербезпека, фішингові листи, виявлення загроз, машинне навчання, генеративний штучний інтелект.

У кваліфікаційній роботі досліджено потенціал застосування великих мовних моделей у сфері кібербезпеки та їхню ефективність для автоматичного виявлення фішингових електронних листів. На теоретичному рівні здійснено огляд архітектур GPT, BERT і T5, проаналізовано етапи передтренування, інструкційного довчання та RLHF-узгодження, а також окреслено місце LLM у процесах Security Operations Center (SOC). Практична частина передбачала формування збалансованого набору електронних листів (Enron + Phishing Email Dataset).

Експериментальна оцінка проводилася за метриками accuracy, recall, precision та F1-міри й порівнювала LLM-рішення з класичними підходами (SVM, дерева рішень, rule-based фільтр). Отримано, що LLM досягає точності 98.5% і F1-міри 90%, що на 7-10 відсоткових пунктів перевищує результати традиційних методів при зіставній кількості хибно позитивних спрацювань.

Результати роботи підтверджують доцільність використання LLM для автоматизованого аналізу текстових загроз і демонструють їхню перевагу над класичними алгоритмами у завданнях фішинг-детекції. Запропоновані підходи та результати можуть бути використані для підвищення оперативної ефективності центрів моніторингу безпеки та слугуватимуть основою для розробки політик безпечного впровадження генеративного ШІ у корпоративні захисні платформи.

ABSTRACT

Application of LLM in Cybersecurity// Thesis of educational level “Bachelor”// Pavlat Oleksandr // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, group СБ-42 // Ternopil, 2025 // P. 92, fig. - 16, tab. - 1, drawing - , add. – 1.

Keywords: large language models, cybersecurity, phishing emails, threat detection, machine learning, generative artificial intelligence.

This bachelor’s thesis investigates the potential of large language models (LLMs) in cyber-security and assesses their effectiveness for the automatic detection of phishing e-mails. On the theoretical level, the study reviews the GPT, BERT and T5 architectures, analyses the stages of pre-training, instruction-tuning and RLHF alignment, and situates LLMs within the workflow of a Security Operations Center (SOC). The practical component involved constructing a balanced corpus of messages (Enron + Phishing Email Dataset).

Experimental evaluation, carried out using accuracy, recall, precision and F1-score, compares the LLM-based solution with classical approaches—support-vector machines, decision trees and a rule-based filter. The LLM achieved 98,5% accuracy and a 90% F1-score, surpassing traditional methods by 7-10 percentage points while maintaining a comparable false-positive rate.

The findings confirm the appropriateness of employing LLMs for automated textual-threat analysis and demonstrate their superiority over classical algorithms in phishing-detection tasks. The proposed methods and results can improve the operational effectiveness of security monitoring centers and provide a foundation for formulating safe-deployment policies for generative AI within corporate defence platforms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI	—	Artificial Intelligence
BERT	—	Bidirectional Encoder Representations from Transformers
CVE	—	Common Vulnerabilities and Exposures
CWE	—	Common Weakness Enumeration
GPT	—	Generative Pre-trained Transformer
GPU	—	Graphics Processing Unit
LLM	—	Large Language Model
LSTM	—	Long Short-Term Memory
NLP	—	Natural Language Processing
OSINT	—	Open-Source Intelligence
RAG	—	Retrieval-Augmented Generation
RLHF	—	Reinforcement Learning with Human Feedback
RM	—	Reward Model
RNN	—	Recurrent Neural Network
SOC	—	Security Operations Center
T5	—	Text-to-Text Transfer Transformer
TPU	—	Tensor Processing Unit

ВСТУП

Упродовж останнього десятиліття кіберпростір перетворився на ключову інфраструктуру сучасного суспільства – від глобальних фінансових систем до персональних мобільних пристроїв. Паралельно з цим розрослася і кількість кіберзагроз: дедалі складніші фішингові кампанії, експлуатація “нульових днів”, автоматизовані атаки на хмарні служби та безперервне зростання обсягу зловмисного трафіку. Традиційні засоби захисту, засновані переважно на сигнатурному аналізі та жорстких правилах, дедалі частіше виявляються недостатніми у протистоянні динамічним і добре скоординованим противникам.

Поява великих мовних моделей (Large Language Models, LLM) відкрила нову парадигму кіберзахисту. Завдяки мільярдам параметрів і навчанню на терабайтах текстових даних такі моделі продемонстрували здатність до глибокого контекстного розуміння, аргументованого міркування та багатоетапного Chain-of-Thought аналізу. Від автоматизованого створення звітів Cyber Threat Intelligence і класифікації журналів подій до виявлення фішингових листів – LLM за короткий час завоювали репутацію універсального інструмента “першої лінії” в інфраструктурі SOC. Водночас широке впровадження таких моделей супроводжується невирішеними питаннями: їхньою здатністю до хибно-позитивних рішень, етичними ризиками, обчислювальною вартістю та відсутністю усталених методик оцінювання ефективності саме в безпекових доменах.

Актуальність роботи визначається необхідністю комплексно оцінити потенціал LLM у кібербезпеці та сформувати методичні підходи до їхнього безпечного та результативного використання. Особливу увагу приділено практичному аспекту – побудові прототипу системи виявлення фішингових емейл листів на базі LLM і порівнянню її продуктивності з класичними рішеннями.

Метою даної кваліфікаційної роботи є дослідження та експериментальна перевірка застосовності великих мовних моделей для ключових завдань

кіберзахисту, з акцентом на автоматичне виявлення фішингових листів і аналіз їхніх переваги та недоліків порівняно з традиційними підходами.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

- здійснити ґрунтовний огляд теоретичних засад LLM, їхніх архітектур (GPT, BERT, T5);
- систематизувати сучасні загрози у сфері інформаційної безпеки й визначити точки інтеграції ШІ-рішень в SOC-процеси;
- сформувати набір даних фішингових і легітимних листів (Enron + p-hishing datasets);
- реалізувати прототип LLM-системи виявлення фішингу;
- провести серію експериментів із порівнянням точності, повноти та F1-міри прототипу з класичними ML-алгоритмами (SVM, дерева рішень, rule-based фільтри);
- оцінити обчислювальні витрати, стійкість до prompt-injection та рівень хибно-позитивних рішень створеної системи.

Об’єктом дослідження виступають процеси виявлення та попередження кіберзагроз, що оперують текстовими даними.

Предметом дослідження є ефективність застосування великих мовних моделей для автоматизованого аналізу таких даних, зокрема для виявлення фішингу.

Практичне значення роботи полягає у створенні й верифікації репрезентативного прототипу LLM-базованого фільтра, який може бути інтегрований у системи електронної пошти або SOC-платформи. Отримані експериментальні результати слугують підґрунтям для оцінювання рентабельності LLM-рішень, формування політик їхнього безпечного використання.

Таким чином, дослідження поєднує теоретичний аналіз еволюції LLM і практичну апробацію їхнього застосування, що разом сприяє підвищенню рівня обґрунтованості й безпеки впровадження новітніх ШІ-технологій у сфері кібербезпеки.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ LLM

1.1 Поняття великомасштабних мовних моделей (LLM)

Мова відіграє фундаментальну роль у комунікації, самовираженні людини та взаємодії людини з машинами. Потреба у “загальних” (generalisable) моделях виникла через зростаючий запит на те, щоби машини розв’язували складні мовні завдання, наприклад: переклад, реферування, пошук інформації, діалогові взаємодії тощо. Останніми роками зафіксовано значні прориви у розвитку мовних моделей, що пояснюється появою трансформерів [1], зростанням обчислювальних потужностей і доступністю великомасштабних тренувальних корпусів. Сукупність цих факторів спричинила революційні зміни й дала змогу створювати великі мовні моделі (LLM, Large Language Model), здатні наближатися до людського рівня виконання різноманітних завдань. LLM постали як передові системи штучного інтелекту, що генерують зв’язний текст і демонструють високу здатність до узагальнення на численні задачі.

LLM – це різновид системи штучного інтелекту, здатної генерувати тексти, подібні до людських, на підставі закономірностей і взаємозв’язків, виявлених у гігантських масивах даних. Для опрацювання книжок, статей і веб-сторінок такі моделі застосовують метод глибокого навчання, що належить до машинного навчання.

Поява LLM відкрила безпрецедентні можливості в області обробки природної мови. Яскравим підтвердженням цього стала публікація GPT-3 компанії OpenAI у 2020 році – на той час найбільшої мовної моделі у світі [2]. Подібні системи створюються так, щоб розуміти контекст і зміст тексту та формувати граматично правильні й семантично релевантні відповіді. Їх можна навчати виконувати найрізноманітніші завдання: переклад, реферування, постановку запитань і надання відповідей, автодоповнення.

GPT-3 наочно довела, що модель великого масштабу здатна якісно виконувати широкий, раніше недосяжний спектр NLP-завдань (Natural Language Processing), а саме: від стислих викладів до творчої генерації текстів. Результати

роботи цієї моделі майже не відрізняються від написаного людиною, при цьому основне навчання відбувається майже без участі людини. Це стало радикальним кроком уперед порівняно з попередніми, головно правило-орієнтованими системами, які не вміли самостійно навчатися й не могли розв'язувати задачі поза межами початкового тренування.

Не дивно, що невдовзі після виходу GPT-3 численні компанії та стартапи почали розробляти власні LLM або інтегрувати вже існуючі, аби прискорити операційні процеси, знизити витрати та оптимізувати робочі потоки.

Індекс AI Stanford (2024) констатує, що протягом року у США оприлюднено 61 "значущу" модель проти 21 у ЄС та 15 у Китаї, що підтверджує домінування американських лабораторій у розробці foundation-моделей.

Хронологічне представлення релізів LLM представлено на рис. 1.1.

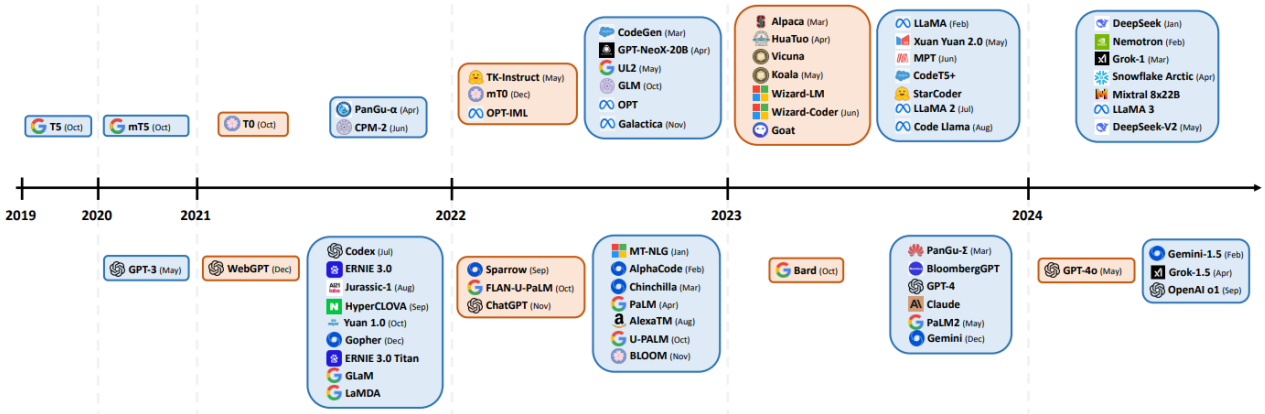


Рисунок 1.1 – Хронологічне представлення релізів LLM [3]

Блакитні картки позначають попередньо натреновані моделі (pre-trained), тоді як оранжеві відповідають додатково навченим за інструкціями (instruction-tuned). Моделі, розміщені у верхній половині схеми, доступні з відкритим вихідним кодом, у нижній половині є закритими. Діаграма відображає зростаючу тенденцію до застосування інструкційного довчання та відкритих рішень, підкреслюючи динамічний розвиток і поточні тренди в дослідженнях обробки природної мови.

Моделі LLM здобули велику популярність завдяки своїй здатності вивчати та генерувати тексти з великою кількістю параметрів і, відповідно, високою точністю. Вони використовують нейронні мережі, зокрема глибокі нейронні мережі, які можуть працювати з величезними обсягами даних і забезпечують високу ступінь абстракції та розуміння тексту.

Проте, одна з головних складнощів, пов'язаних із LLM, полягає в тому, що вони потребують великих обчислювальних ресурсів для тренування. Для цього використовуються спеціалізовані апаратні засоби, такі як графічні процесори (GPU) та тензорні процесори (TPU), що дозволяють ефективно тренувати моделі на великих наборах даних.

Однією з головних переваг LLM є їх здатність до переносного навчання (transfer learning), що дозволяє переносити знання з однієї задачі на іншу. Це означає, що після навчання на великих текстових корпусах, моделі можуть бути адаптовані для вирішення специфічних задач, таких як виявлення фішингових листів, класифікація текстів чи навіть розпізнавання кібератак.

Моделі LLM зазвичай навчаються на наборах текстових даних, що включають інтернет-ресурси, книжки, наукові статті, веб-сайти та інші джерела. Оскільки ці моделі обробляють величезні обсяги тексту, вони здатні вивчати різноманітні аспекти мови, зокрема граматику, лексичні зв’язки, контекстуальні залежності та інші тонкощі.

Згідно з McKinsey Global Survey (2023) [4], уже третина опитаних компаній регулярно застосовує генеративні LLM принаймні в одній бізнес-функції; загалом 60 % організацій, що вже користуються AI, інтегрували саме генерувальне ядро, а 40 % респондентів декларують намір наростити інвестиції в AI саме завдяки LLM-рішенням [5]. Додатково опитування Datanami (серпень 2023) показало, що 58 % компаній експериментують із LLM, хоча лише 23% уже перевели такі експерименти у продуктивну експлуатаці. Amazon Web Services фіксує, що 64 % великих підприємств в Індії поставили GenAI-проєкти у пріоритет інвестування на 2025 рік [6].

Сайт Hugging Face пропонує більш як 1,8 млн відкритих моделей (станом на серпень 2024 року), більшу частину з них становлять LLM-перевивантаження

(LoRA, QLoRA, fine-tune-версії) і доменно-специфічні моделі. Така кількість репозиторіїв демонструє не лише попит, а й активну участь спільноти у кастомізації та вдосконаленні базових моделей [7].

Згідно кількості статей проіндексованим у Google Scholar кількість застосувань LLM у кібербезпеці з кожним роком зростає, така тенденція представлена на рисунку 1.2.

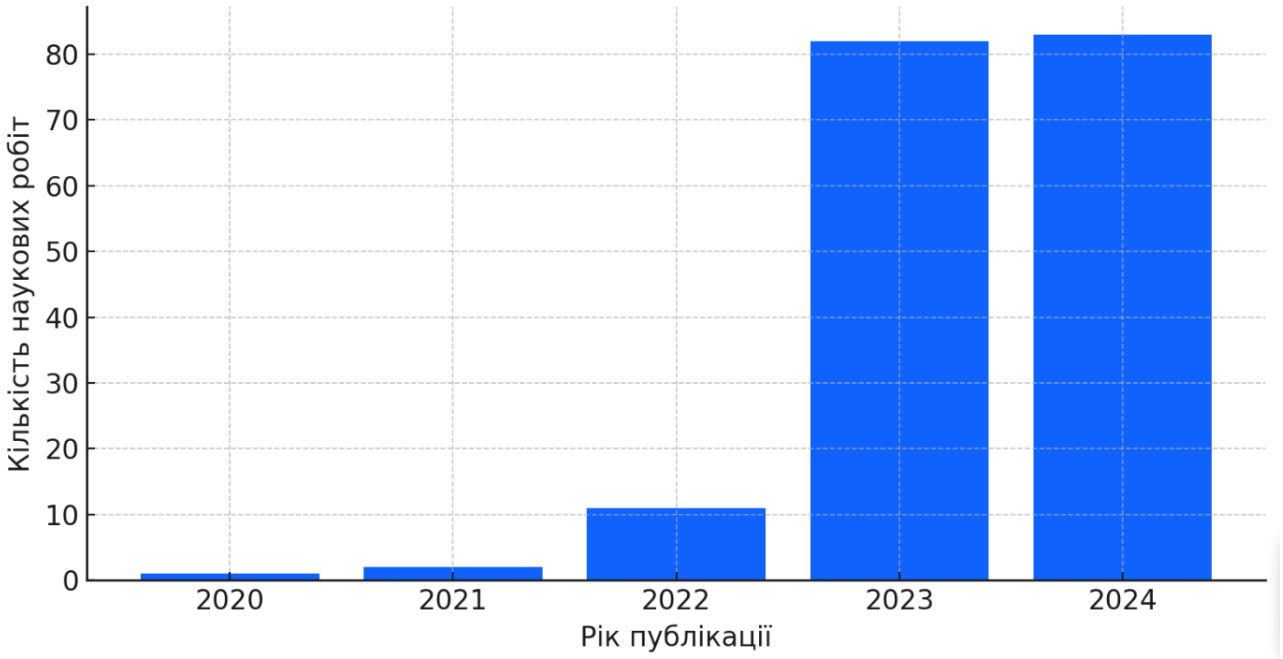


Рисунок 1.2 – Кількість досліджень та застосувань LLM у кібербезпеці на основі застосування статті

Аналіз 950 965 публікацій (січень 2020 - лютий 2024) показав, що у комп'ютерних науках частка статей, де текст суттєво редагувався або генерувався LLM, сягнула 17,5 %; у математиці та журналах Nature до 6,3. Інше опитування (Stanford HAI, 2024) встановило, що вже 17 % рецензій на конференціях містять LLM-фрагменти. Таким чином, LLM стають звичайним інструментом наукового письма й рецензування.

Автори статті також провели детальний аналіз застосування LLM у наукових дослідженнях (у назвах чи анотаціях яких зустрічаються ключові словосполучення "Large Language Model", "Large Language Model + Fine-Tuning" та "Large Language Model + Alignment") по типах моделей та представили

динаміку виходу відповідних публікацій у відповідності до років, що представлено на рисунку 1.3.

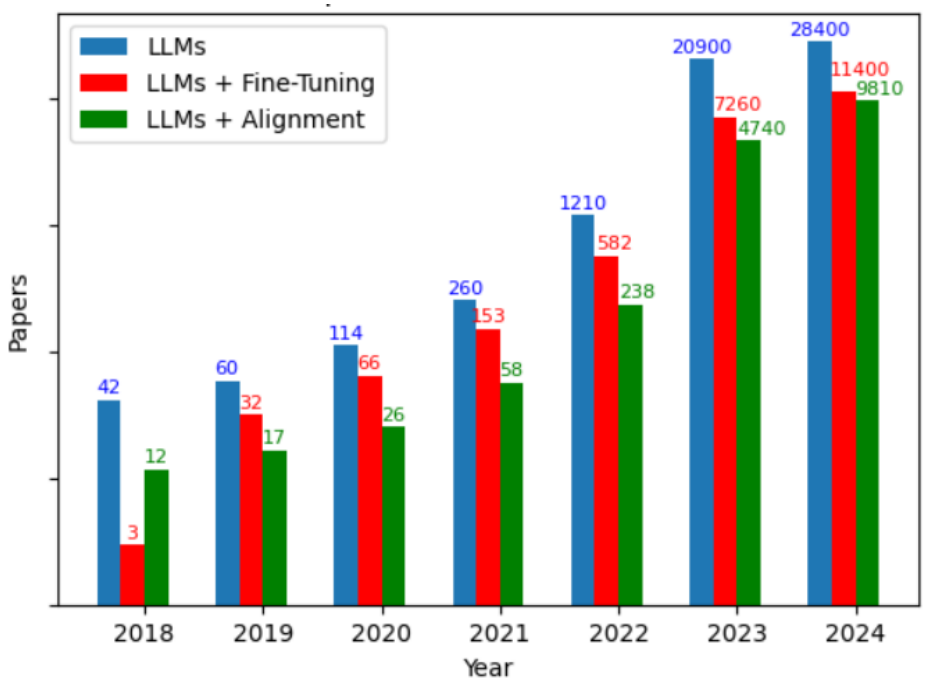


Рисунок 1.3 – Динаміка публікацій застосування LLM за роками [3]

Діаграма демонструє експоненційне зростання наукового інтересу до великих мовних моделей (LLM) у період 2018-2024 років, причому не лише до самих LLM, а й до двох критично важливих напрямів їх розвитку: довчання (fine-tuning) та узгодження з людськими настановами (alignment).

Кількість публікацій про LLM зросла з кількох десятків у 2018 р. (42 статті) до майже 28 400 у 2024 році, тобто більш ніж у 600 разів. Подібний, хоч і дещо менш стрімкий, приріст спостерігається для робіт про fine-tuning і alignment.

Частка статей, присвячених fine-tuning, зросла з 7 % від загальної кількості LLM-досліджень у 2018 році до приблизно 40 % у 2024 році. Це засвідчує, що головний фокус спільноти переміщається від створення базових моделей до їх доменного та налаштування на задачі.

Роботи з узгодженням моделей із людськими інструкціями або етичними рамками (alignment) збільшилися з 12 у 2018 році до 9 810 у 2024 роц. Такий показник демонструє, що такого типу моделі майже наздогнали fine-tuning

навчання. Це виражає усвідомлену потребу не лише в потужних, а й у контрольованих, безпечних моделях.

Розробка LLM більше не обмежується збільшенням параметрів. Також пріоритетів набувають методи, які роблять моделі придатними до конкретних завдань і водночас безпечними з погляду користувача й регуляторної відповідності. Для академічної спільноти та галузі це означає, що майбутні інновації найімовірніше зосередяться на ефективних схемах адаптації, тонкого налаштування й етичного контролю великих мовних моделей.

1.2 Основні архітектури LLM: BERT, GPT, T5

Коли з'явилися перші LLM, їх будували на відносно простих нейронних архітектурах із невеликою кількістю шарів – переважно на рекурентних нейронних мережах (RNN) та мережах довго-короткої пам'яті (LSTM). На відміну від класичних нейронних мереж, RNN і LSTM здатні враховувати контекст, позицію та взаємозв'язки між словами навіть тоді, коли ці слова віддалені одне від одного в послідовності. Інакше кажучи, вони "пам'ятають" попередні дані і враховують їх під час формування виходу, що забезпечує вищу точність у багатьох NLP-завданнях, зокрема аналізі тональності та класифікації текстів.

Головна перевага таких мереж над традиційними, заснованими на правилах системами полягала в тому, що RNN і LSTM можуть навчатися майже без участі людини. Вони аналізують сирі дані й самостійно формують внутрішні закономірності, замість того щоб спершу отримати "написані вручну" правила і лише потім застосовувати їх до даних. Цей підхід називають навчанням представлень (representation learning) – концепцією, що наслідує принципи людського навчання.

Один із найважливіших інструментів штучного інтелекту це навчання з вчителем. Воно особливо ефективне, коли потрібно "розставляти мітки", тобто навчатись на парі "вхід → правильна відповідь". Другою за значущістю (представлено на рисунку 1.4), причому лише відносно недавно досяг високої

ефективності, є генеративний штучний інтелект (Generative AI). Значно рідше використовують й інші підходи, зокрема навчання без вчитедя (unsupervised) та підкріплювальне (reinforcement) навчання. Саме навчання з вчителем та генеративний AI є двома головними інструментами штучного інтелекту на сьогодні. Для більшості бізнес-завдань поки достатньо оволодіти цими двома, не занурюючись у решту.

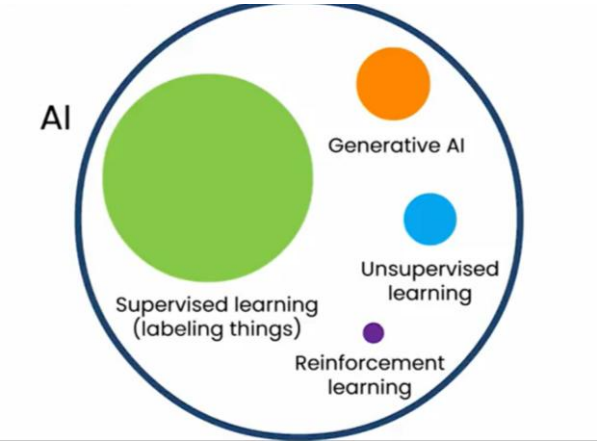


Рисунок 1.4 – Типи навчань AI та їх частка використання

Перш ніж пояснювати принцип роботи генеративної II, коротко нагадаю, що таке супервізоване навчання, адже саме на ньому побудовано сучасні LLM. Супервізоване навчання зробило комп'ютери надзвичайно вправними у задачі "за заданим входом A згенерувати правильний вихід B". Представимо кілька детальних прикладів (рисунок 1.5).

Вхід (A)	Вихід (B)	Застосунок
email	спам? (0/1)	фільтр спаму
аудіо	переклад в текст	розпізнавання мови
English	Українська	машинний переклад
зображення, інформація з радару	позиції інших машин	самокероване авто
зображення деталі	дефект? (0/1)	візуальна інспекція

Рисунок 1.5 – Приклади навчання з вчителем

На першому прикладі представлено класифікацію електронної кореспонденції на “спам/не спам”, що є типовим бінарним завданням. Модель одержує текст листа (вхід A) і повинна передбачити бітову мітку B, що вказує на належність повідомлення до небажаної пошти.

Другий приклад стосується автоматичного розпізнавання мовлення. Тут на вході подається акустичний сигнал (A), а цільовою відповіддю є текстова транскрипція (B). Модель, навчена на парах “аудіо-уривок → текст”, опановує складний багаторівневий простір фонетичних та лінгвістичних ознак, що дозволяє перетворювати усне мовлення на письмову форму з мінімальною помилкою.

Період приблизно 2010-2020 років став “десятиліттям великомасштабного навчання з вчителем”. Виявилося, що для багатьох задач існує чимало даних, та маленькі моделі не здатні повністю використати їхній потенціал. Прорив настав, коли дослідники почали тренувати дуже великі моделі на дуже потужних обчислювальних кластерах: із зростанням обсягу даних якість продовжувала суттєво зростати. Для прикладу першою місією Google Brain було “будувати максимально великі моделі й годувати їх великою кількістю даних”. Ця стратегія дала вагомий поштовх прогресу.

Саме підхід великих моделей для “позначення речей” проклав шлях до генеративної AI. Розгляньмо на рис. 1.6, як генеруються тексти за допомогою LLM.

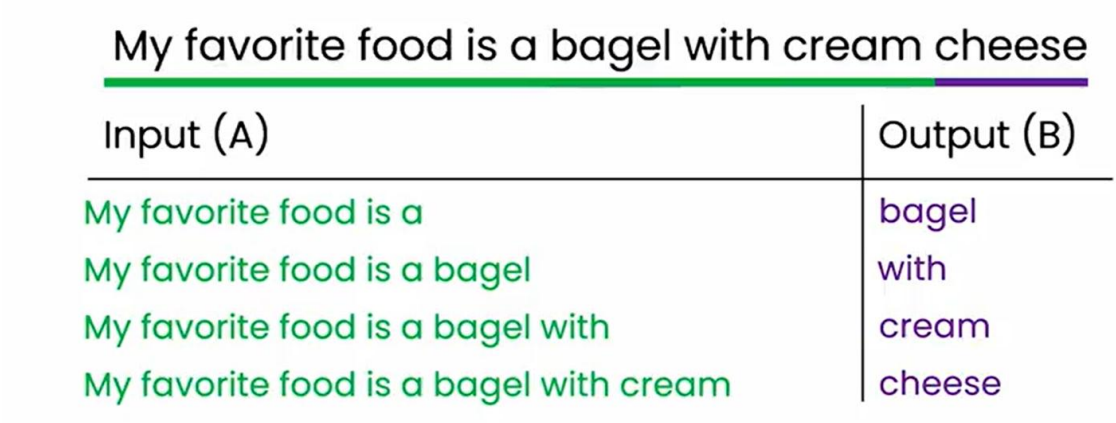


Рисунок 1.6 – Приклад генерації тексту LLM

У випадку, коли говорять про LLM, то мають на увазі базові мовні моделі (foundation models), наприклад MT-NLG і GPT-3 чи GPT-4. Їх навчають на величезних обсягах текстових даних, завдяки чому вони здатні виконувати широкий спектр NLP-завдань: від відповідей на запитання й генерування анотацій книжок до доповнення чи перекладу речень.

Припустімо, ми подаємо до моделі підказку (prompt) “I love eating”. LLM може продовжити речення як “bagels with cream cheese”. Після другого запуску може продовжити речення як “my mother’s meatloaf”, а за третього – out with friends тощо.

LLM навчається все тим же супервізованим способом, але на задачі передбачення наступного слова. Наприклад, у реченні My favorite food is a bagel with cream cheese комп’ютер бачить багато тренувальних прикладів:

- контекст My favorite food is a … → правильна відповідь bagel;
- контекст My favorite food is a bagel … → відповідь with; і так далі.

Таких пар “контекст → наступне слово” в задачі є сотні мільярдів. Тренування моделі на сотнях мільярдів, інколи трильйонах слів дає змогу LLM (наприклад, ChatGPT) дуже переконливо продовжувати тексти під різноманітні запити.

Поверх такого базового “прогнозування слова” додаються інструкційне донавчання, щоби змусити модель виконувати інструкції та генерувати безпечний контент. Та ядром LLM лишається саме вміння статистично передбачати наступне слово в довжелезних послідовностях.

Практична користь уже є очевидно. Багато хто щодня застосовує такі моделі для написання текстів, пошуку інформації чи як інтелектуального “співрозмовника”, що допомагає обдумувати ідеї.

Поява LLM спричинила докорінний зсув у способах проєктування, навчання та використання моделей обробки природної мови. Щоб усвідомити масштаб змін, корисно порівняти LLM із попередніми поколіннями NLP-систем. Для цього коротко окреслимо три етапи розвитку NLP [8]:

- Дотрансформерна ера.
- Трансформерна ера.

- Ера LLM.

Дотрансформерна ера характеризується тим, що у цей період переважали моделі, що ґрунтувалися на ручних правилах, а не на машинному навчанні. Вони були прийнятні для простих завдань (наприклад, класифікації текстів), але малопридатні для складніших, таких як машинний переклад. До того ж rule-based підходи погано працювали з "нетиповими" випадками, оскільки не могли робити коректні прогнози для даних, на які не існувало явних правил. Частково проблему пом'якшили прості нейронні мережі – RNN та LSTM, що з'явилися на пізній фазі цього етапу: вони могли певною мірою запам'ятовувати попередній контекст і, отже, формувати контекстно-залежну класифікацію. Проте RNN і LSTM не здатні були утримувати контекст на довгих відрізках тексту, що обмежувало їхню ефективність.

Перелом настав із появою архітектури Transformer у 2017 році, що започаткувало еру транфмормерів. Трансформери краще узагальнювали, ніж тодішні RNN та LSTM, захоплювали ширший контекст і обробляли значно більші обсяги даних одночасно. Це дало змогу моделям працювати з довгими послідовностями та опановувати набагато ширший спектр завдань. Однак із нинішньої точки зору моделі того періоду залишалися обмеженими: бракувало по-справжньому великомасштабних корпусів і достатніх обчислювальних ресурсів. До того ж їхнім використанням здебільшого цікавилися дослідники й галузеві експерти; для широкої публіки такі системи були недостатньо точними та не надто зручними.

Початок ери LLM поклала публікація GPT-3 компанії OpenAI у 2020 р. Великі мовні моделі, подібні до GPT-3, тренувалися на колосальних об'ємах тексту, що дало їм змогу формувати значно точніші й змістовніші відповіді, ніж попередники. Це відкрило нові горизонти й наблизило нас до того, що багато хто вважає "справжнім" ШІ. Крім того, LLM зробили технології NLP доступними не-технічним користувачам: достатньо сформулювати запит природною мовою і модель виконує потрібне завдання. Фактично відбулася демократизація NLP.

Перехід від одного методологічного етапу до іншого зумовили ключові технологічні та методичні досягнення: поява нейронних мереж, механізмів уваги

(attention), самої архітектури Transformer, а також розвиток несупервізованого й самонавчального (self-supervised) тренування.

Завдяки своїм розмірам базові моделі демонструють прийнятні результати навіть тоді, коли доменно-специфічних даних небагато. Вони показують загалом стабільну якість у різних завданнях, хоча й не обов'язково досягають найвищих показників у кожному окремому випадку.

Донавчені (fine-tuned) моделі є похідними від базових LLM і налаштовуються під конкретні сфери чи задачі, унаслідок чого краще справляються зі спеціалізованими завданнями. Окрім підвищеної точності у вузьких доменах, їхньою суттєвою перевагою є менша "вага" та, зазвичай, простіший і дешевший процес навчання.

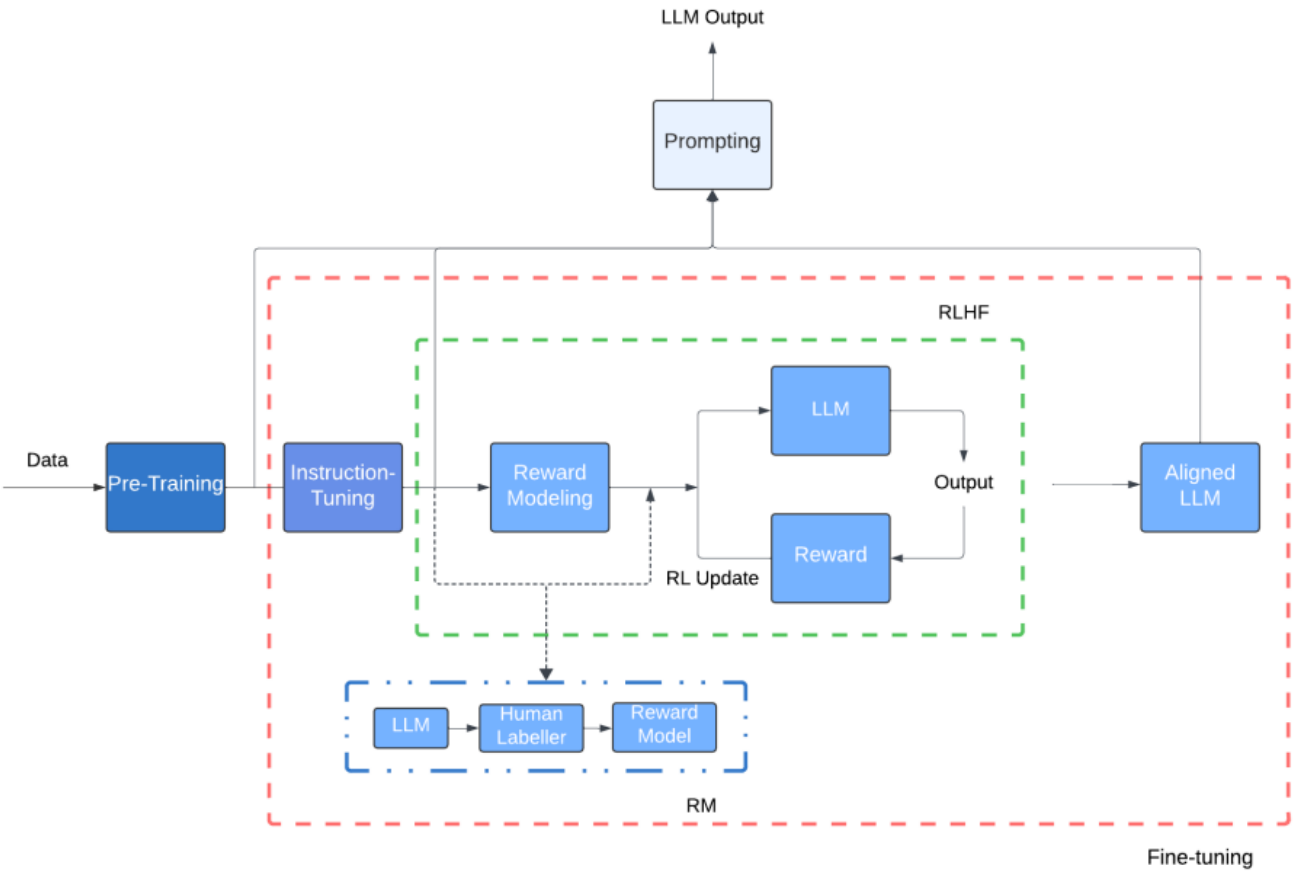


Рисунок 1.7 – Базова блок-схема послідовних етапів навчання fine-tuned LLM [3]

Схема на рисунку 1.7 відображає повний життєвий цикл LLM, починаючи від первинного передтренування і завершуючи стадією використання через

запити. На першому етапі модель навчається у самоконтрольному режимі на масштабному неанотованому корпусі (“Pre-Training”) і формує універсальні мовні представлення. Далі відбувається інструкційне довчання (“Instruction-Tuning”), коли до попередньо натренованої мережі підмішують пари “запит-бажана відповідь”, щоб навчити її слідувати явним інструкціям користувача.

Після інструкційного довчання система переходить до блоку “Reward Modeling”, де зібрані від людини оцінки варіантів відповіді перетворюються на функцію винагороди. Спочатку сама LLM генерує кілька відповідей; аннотатори ранжують їх, і на підставі цього ранжування тренується окрема модель винагороди (RM). Далі виконується цикл підкріплювального навчання “LLM ↔ Reward”: генерований текст оцінюється reward-моделлю, а зворотний сигнал коригує параметри LLM (позначено як “RL Update”). Поєднання підкріплення з людським фідбеком утворює повний контур RLHF.

Після завершення RLHF-процедури отримують “Aligned LLM” – модель, чиї відповіді погоджені з людськими уподобаннями й етичними нормами. У фінальній фазі користувач взаємодіє з цією узгодженою моделлю через механізм prompt-інженерії; поданий запит “Prompting” проходить крізь всю структуру, і на виході формується текстова відповідь LLM. Таким чином, схема демонструє, як базове мовне ядро поступово перетворюється на практичну систему, здатну не лише генерувати граматично правильний текст, а й узгоджувати його зі специфічними інструкціями та ціннісними критеріями людини.

На сьогодні найпоширенішим підходом донавчання є параметрично ощадливе налаштування (parameter-efficient customization): p-tuning, prompt-tuning та адаптери. Такі методи потребують значно менше часу й ресурсів, ніж повне донавчання всієї моделі, хоча іноді й поступаються за кінцевою продуктивністю більш ресурсомістким технікам.

Історично системи штучного інтелекту призначалися насамперед для оброблення й аналізу наявних даних, а не для їх створення. Їхній фокус полягав у “сприйнятті” та розумінні навколишнього світу, а не у продукуванні нової інформації. Саме цим відрізняються “перцептивний” (Perceptive AI) та генеративний (Generative AI) підходи; останній почав стрімко домінувати

приблизно з 2020 років, коли індустрія масово перейняла трансформерні архітектури й розгорнула великомасштабні LLM.

Попри переваги RNN і LSTM порівняно з традиційними підходами, вони мають низку обмежень, через які виявляються непридатними для складніших NLP-завдань, наприклад машинного перекладу. Головна вада полягає в неспроможності опрацьовувати довгі послідовності даних і, відповідно, враховувати повний контекст вхідного сигналу. Через те, що LSTM та RNN погано “утримують” надмірно великий контекст, їхні вихідні тексти схильні до неточностей або навіть абсурдності. Ці та інші проблеми значною мірою вдалося подолати завдяки появі нової архітектури, а саме трансформерів.

1.2.1 Трансформери

Трансформери вперше представили Васвані та співавтори у 2017 році в статті “Attention Is All You Need” [9]. Назва натякала на механізм уваги (attention), що став ключовим елементом цієї архітектури. Її принципова відмінність від рекурентних мереж полягає у відмові від послідовного (time-step) опрацювання на користь паралельного механізму багатоголової уваги (*multi-head attention*). Саме увага дає змогу моделі на кожному шарі динамічно переважувати внесок різних токенів, у результаті чого довгі залежності фіксуються без втрат контексту, а обчислення легко розпаралелюються на GPU/TPU.

На високому рівні трансформер encoder–decoder складається з трьох функціональних блоків.

Першим блоком є вхідні та вихідні вбудування (Embeddings + Positional Encoding). Сирі токени перетворюються на вектори сталої розмірності; до них додаються позиційні коди, які вводять інформацію про порядок елементів.

Другим блоком є каскад енкодерів (Encoder, повторений N разів). Кожен енкодерний шар містить модуль багатоголової уваги та двошарову feed-forward мережу з резидуальними з’єднаннями та шаровою нормалізацією. Енкодер продукує набір контекстуалізованих векторів, які слугують “пам’яттю” для декодера.

Каскад декодерів (Decoder, повторений N разів). Декодерні шари мають дві уваги: “масковану” (self-attention) для вже згенерованої частини вихідної послідовності та “перехресну” (cross-attention) до виходу енкодера. На останньому етапі проєкція на словник і softmax дають розподіл імовірностей наступного токена.

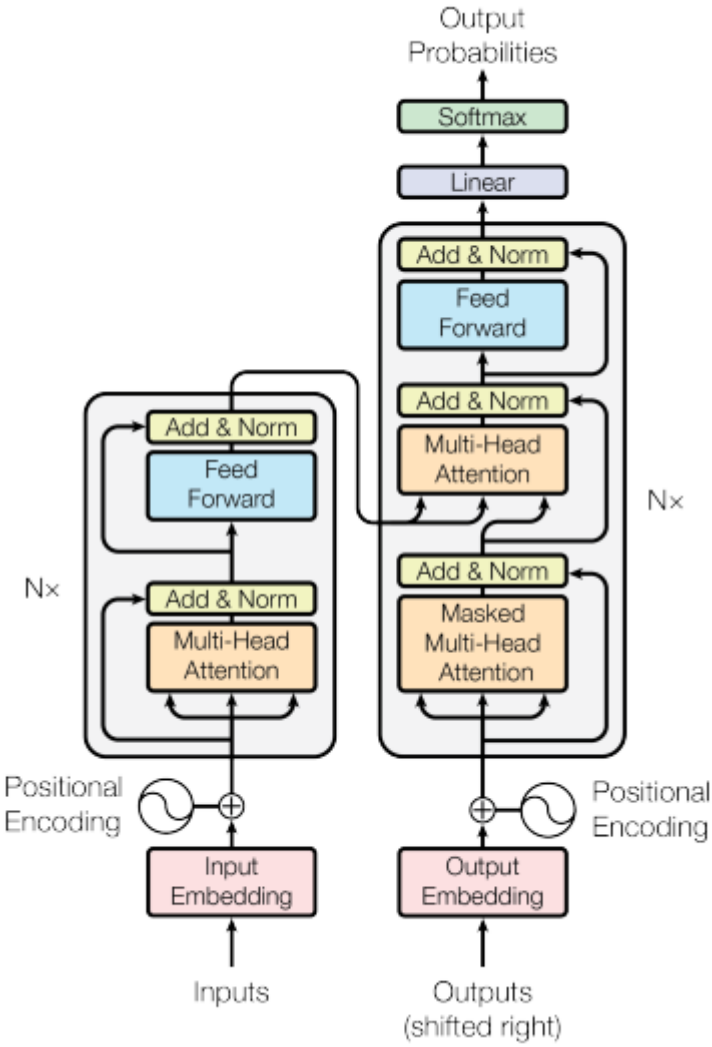


Рисунок 1.8 – Архітектура трансформера [9]

Усі підшари оточуються операціями Add & Norm, що сприяє стабільному градієнтному потоку на великих глибинах. Паралельність уваги забезпечує майже лінійне масштабування пропускної здатності відносно потужності апаратного кластера, що й дозволило підняти кількість параметрів сучасних LLM до сотень мільярдів.

1.2.2 Архітектура BERT

BERT (Bidirectional Encoder Representations from Transformers) – одна із популярних архітектур, яка була розроблена для двонаправленого розуміння контексту. На відміну від GPT, BERT вивчає текст як зліва направо, так і справа наліво, що дозволяє краще враховувати взаємозв'язки між словами в реченні. Це робить BERT особливо ефективним для задач, де важлива точність розуміння контексту, таких як класифікація текстів, аналітика тональності або виявлення фішингових листів. BERT здатний успішно розв'язувати різні завдання NLP, включаючи пошук відповідей на запитання та екстракцію інформації.

BERT розроблена дослідниками Google AI на чолі з Jacob Devlin, Ming-Wei Chang, Kenton Lee та Kristina Toutanova й оприлюднена 31 жовтня 2018 року [10]. Її появі передувала хвиля контекстних мовних представлень (ELMo, ULMFiT) і успіхи перших автогенеративних трансформерів GPT-1/2, які продемонстрували, що попереднє навчання на великих корпусах різко підвищує якість у NLP-завданнях. BERT зробив наступний крок: замість unidirectional- або seq2seq-трансформерів він уперше використав глибоко двобічний (bidirectional) енкодер-тільки трансформер, що дає змогу одночасно враховувати лівий і правий контекст слова.

BERT складається з WordPiece-токенізатора з позиційними кодуваннями, стека з L = 12 (“BASE”) або 24 (“LARGE”) шарів self-attention енкодера, двох допоміжних pre-training задач – Masked Language Modeling (15 % випадкових токенів маскуються й модель відновлює їх) та Next Sentence Prediction (класифікує, чи друга фраза є продовженням першої). Після попереднього тренування BERT донавчають, додаючи невелику task-head над командним токеном [CLS]. Така будова забезпечує “plug-and-play” перенос на майже будь-яку задачу розуміння тексту: достатньо тонко налаштувати лише кілька останніх шарів або навіть один класифікатор.

Схема на рисунку 1.9 відображає один трансформер-блок моделі BERT-типу, але з реалізацією механізму уваги у формі Dense Synthesizer. Структура читається зверху донизу й повторюється *n* разів усередині енкодера.

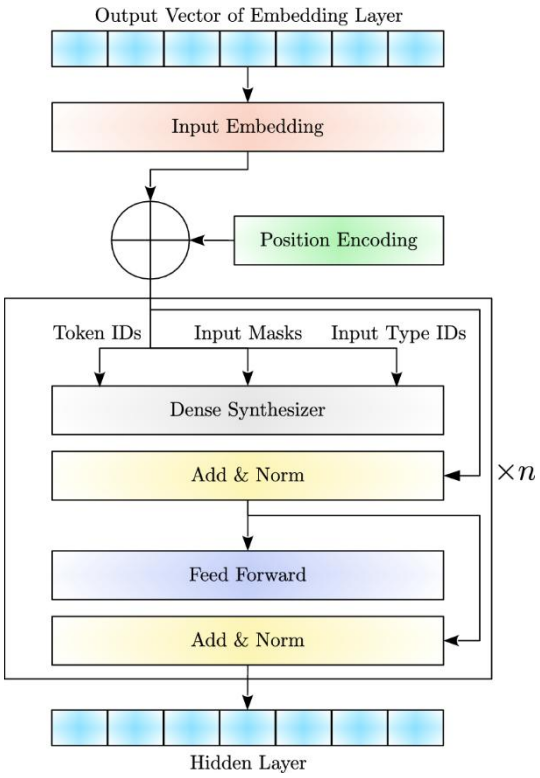


Рисунок 1.9 – Один трансформер-блок моделі BERT-типу [10]

Уже статті [10] модель встановила рекорди на GLUE, SQuAD v1.1/v2.0 та SWAG, а 199-ма мовами BERT-моделей швидко поповнилися пошукові системи (Google Search, Bing), діалогові агенти, системи рекомендацій, біомедичні та юридичні аналізатори. Сьогодні BERT і його похідні (RoBERTa, DistilBERT, ALBERT, Multilingual-BERT) є де-факто стартовою точкою для класифікації е-мейлів, FAQ-чатботів, NER-витягування сутностей, sentiment-аналізу та навіть для кодування білкових послідовностей у біоінформатиці. Попри значний науковий і прикладний успіх, архітектура BERT має низку відомих обмежень. По-перше, механізм самоуваги характеризується квадратичною складністю; отже, у стандартній конфігурації модель опрацьовує відрізки не довші за приблизно 512 токенів і для повноцінного тренування потребує обчислювальних ресурсів класу GPU/TPU високого рівня. По-друге, саме попереднє навчання залишається вкрай затратним: для версії BERT-Large, наприклад, довелося задіяти 64 чипи TPU-v3 протягом чотирьох діб. Крім того, двобічне мовне моделювання успадковує статистичні упередження та артефакти корпусу, що проявляється у формі “галюцинацій” і вимагає подальшого очищення даних або

донавчання з участю людини. Ще одним виявленим недоліком стала малоефективність завдання прогнозування наступного речення (NSP); сучасні модифікації, зокрема RoBERTa чи ELECTRA, повністю виключили NSP або замінили його альтернативними процедурами узгодження. Нарешті, у виробничих сценаріях модель виявляє вразливість до атак типу prompt injection, що зумовлює потребу в додаткових фільтрах і механізмах контролю генерованого контенту.

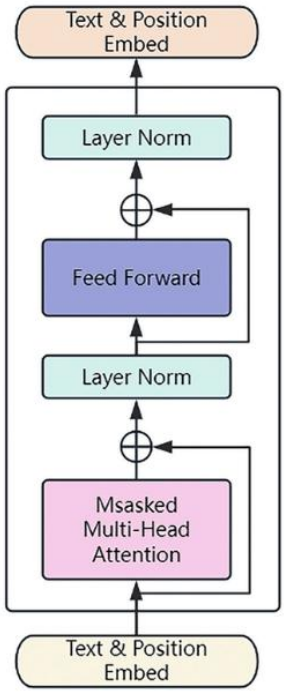
BERT позначив перелам у NLP, довівши, що масивне двобічне передтренування трансформера може дати універсальний "мовний фундамент" для десятків downstream-задач. Попри високу вартість і структурні обмеження, концепція контекстних енкодерів залишилася базовою: саме на ній будують сучасні багатомовні, доменно-спеціфічні та "скасовані" (distilled) моделі, які й досі задають тон у практичних застосуваннях обробки природної мови.

1.2.3 Архітектура GPT

Ґенеративний попередньо натренований трансформер (GPT) – це лінійка моделей, яку розробила дослідницька лабораторія OpenAI під керівництвом Іллі Сутскевера, Алекса Редфорда та їхньої команди. Перша версія ("GPT-1") була оприлюднена у червні 2018 році, а вже у 2019-му світ побачила GPT-2, що збурила спільноту своїм небаченим раніше масштабом (1,5 млрд параметрів) та пропозицією тимчасово обмежити публікацію повної моделі з міркувань безпеки. Проривом стала GPT-3 (175 млрд параметрів; травень 2020 р.) – саме вона продемонструвала феномен "zero-/few-shot"-умінь і підштовхнула бум генеративного ШІ. У березні 2023 р. відбулась комерційна прем'єра GPT-4, а сьогодні екосистема поповнюється модифікаціями типу GPT-4o, налаштованими на багатомодальний ввід.

Ключовою технологічною основою GPT стала архітектура Transformer (Google Brain, 2017), яка завдячує своїй силі механізму багатоголової уваги. До її появи домінували рекурентні мережі (RNN/LSTM), що погано масштабувалися

на довгі послідовності. Transformer уперше дозволив паралельно обробляти токени й навчати дуже великі моделі, а отже стати фундаментом GPT.



(b) GPT Transformer Decoder (GPT-1)

Рисунок 1.10 – GPT трансформер декодер [11]

На зображенні подано спрощений блок-чарт decoder-only трансформера, характерний для GPT-серії (рисунок 1.10):

- Вбудований шар(Input Embedding) перетворює індекси токенів на щільні вектори; до них додають позиційне кодування, щоб модель “знала” порядок слів.
- Далі йде каскад із n декодерних шарів. Кожен шар містить: маскований Self-Attention, що бачить лише попередні позиції та вивчає залежності “словопередбачення”; Add & Norm (резідуальне додавання + шарова нормалізація); двошарову Feed-Forward Network; повторне Add & Norm.
- Після останнього шару вектор переводять у розподіл імовірностей над словником через лінійне перетворення + softmax; далі вибирають наступний токен і цикл повторюється.

Така авторегресивна організація робить GPT універсальною “мовною функцією”: достатньо подавати запит (prompt), і модель, предиктуючи токен за токеном, формує завершений вихід.

Сфери застосування GPT-моделей охоплюють практично весь спектр сучасної роботи з текстом і кодом. По-перше, їх активно використовують для генерації різноманітних матеріалів – від журналістських статей і маркетингових описів товарів до художніх нарисів та інших творчих жанрів. По-друге, архітектура GPT лежить в основі діалогових систем, таких як ChatGPT, Microsoft Copilot чи Claude, які забезпечують інтерактивну підтримку користувачів і виконують роль віртуальних асистентів. Третя велика галузь – програмна інженерія: сервіси на кшталт GitHub Copilot і Code Interpreter допомагають генерувати, доповнювати та рецензувати програмний код, прискорюючи розробку. Окремий напрямок – пошук та стисливе резюмування великих масивів інформації, зокрема юридичних документів і наукових публікацій, де моделі формують стислі, зрозумілі огляди. Нарешті, у кібербезпеці GPT застосовують для пояснювальної класифікації фішингових листів, автоматичного генерування правил SIGMA та навіть частково автономних penetration-тестів, що дозволяє підвищити оперативність і точність захисних заходів.

Попри переваги, GPT лишається ресурсомістким: навчання GPT-3-подібної моделі потребує сотень-тисяч GPU/TPU-годин, а квадратична складність self-attention обмежує практичну довжину контексту. Моделі схильні до галюцинацій та відтворення упереджень тренувального корпусу; у відкритому середовищі додатковий ризик становлять prompt-injection-атаки та витік конфіденційних даних. Висока енергоспоживчість і вуглецевий слід також викликають дедалі більше уваги з боку екологічної спільноти.

GPT-архітектура стала трампліном для сучасної генеративної “революції”: її авто-регістрична будова, масштабованість і здатність узагальнювати завдяки масовому попередньому тренуванню зробили можливим використання ШІ-”підказок” практично у всіх секторах, від освіти й творчих індустрій до кібербезпеки та програмної інженерії. Разом із тим розробники й регулятори

сьогодні зосереджують зусилля на методах контролю змісту, підвищенні енергоефективності та зменшенні ризиків зловживань.

1.2.4 Архітектура T5

Модель T5 була представлена дослідницькою групою Google Brain під керівництвом Коліна Раффела (C. Raffel) наприкінці 2019 році у статті “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer” [12] , а вихідні моделі й код-база стали доступними в екосистемі TensorFlow у березні 2020 року . Появі T5 передувала еволюція трансформерних рішень: від GPT-1/2 (автокорегований decoder), через BERT-тип pre-training (маскування токенів в encoder’і) до спроб уніфікувати різні NLP-завдання у спільному форматі. Саме ця ідея “усе – у текст” (text-to-text) стала центральною мотивацією T5.

Ключовими конструктивними особливостями є наступне:

Архітектура encoder–decoder. Обидві частини побудовано на класичних блоках трансформера з багатоголовою self-attention та feed-forward шарами. Різні “розміри” (Base, Large, 3B, 11B) відрізняються глибиною (12 → 24 → 24 → 24 шарів) і шириною прихованого стану [13].

Спільний словник та параметри. Вхідний і вихідний ембедінги поділяють ваги, що економить пам’ять і сприяє симетричності “текст → текст”.

Denoising-об’єктив “span corruption”. Модель випадково ховає довільні фрагменти (span’и) тексту та вчиться відтворювати їх, а не лише окремі слова, як у BERT. Така стратегія краще стимулює “глобальне” моделювання.

Корпус C4. Для попереднього навчання було створено Colossal Clean Crawled Corpus (750 GB), отриманий з Common Crawl шляхом агресивного фільтрування, що зменшує шум і токсичний контент [13].

Уніфікація задач. Кожне завдання (переклад, класифікація, QA, узагальнення тощо) подається як текстовий prompt та очікуваний текстовий вихід, тому одна й та сама модель працює без додаткових “голів” або спеціальних шарів.

На рисунку 1.11 видно дві паралельні гілки. Ліворуч – encoder: токени проходять через вкладення й позиційне кодування, потім через N шарів multi-head self-attention та feed-forward з резидуальними зв’язками. Праворуч знаходиться decoder: перша “маскована” self-attention оперує вже згенерованими токенами, друга attention “дивиться” на вихід encoder’а. Така будова дозволяє T5 читати повний вхідний контекст і покроково генерувати вихід будь-якої довжини.

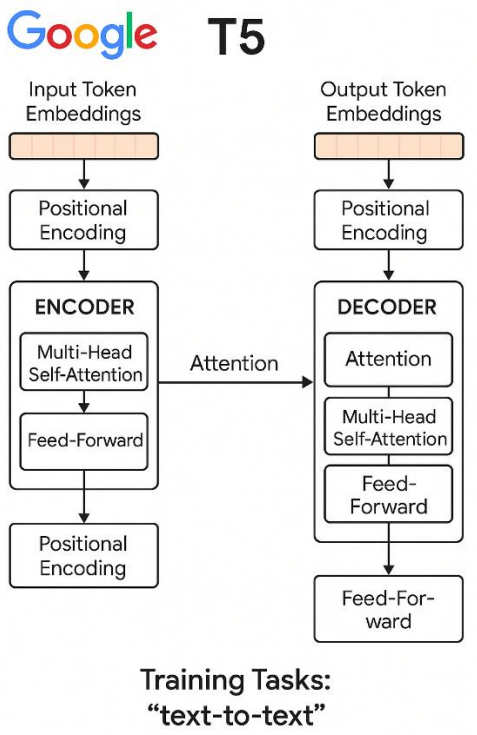


Рисунок 1.11 – Блок-схема моделі T5 [12]

T5 виявилася надзвичайно універсальною: переклад і багатомовний transfer, машинне резюмування (наприклад, Google News Showcase), відповіді на питання (MS MARCO, Natural Questions), генерація коду під час fine-tune (CodeT5), а також доменно-специфічні завдання на кшталт медичних висновків чи юридичних зведень.

Попри гнучкість, T5 успадкувала кілька проблем. Квадратична складність self-attention обмежує практичну довжину послідовності (≈512 токенів у базовій конфігурації) і потребує значних GPU/TPU-ресурсів для тренування й інференсу. Повне попереднє навчання T5-Large тривало ~4 доби на 64 TPU-v3. Висока

енергетична вартість і CO_2-слід стали предметом критики спільноти. Модель схильна до “галюцинацій” і переносить упередження з тренувального набору даних, тому у виробничих застосуваннях вимагає ретельного фільтрування даних і механізмів після-обробки. Нарешті, через text-to-text парадигму все управління відбувається через prompt-інженерію, що робить систему вразливою до prompt-injection та контекстних атак, якщо не застосовано додаткового контролю.

1.3 Застосування ШІ в галузі кібербезпеки

Штучний інтелект (ШІ) набуває все більшої популярності у сфері кібербезпеки, оскільки здатний ефективно обробляти величезні обсяги даних та знаходити патерни, які людина може не помітити. Одним із найбільш перспективних напрямів є застосування великих мовних моделей (LLM) в кібербезпеці для автоматизації виявлення загроз, аналізу фішингових атак, створення нових стратегій захисту і полегшення реагування на інциденти.

ШІ в кібербезпеці допомагає вирішувати наступні завдання:

- Аналіз великих даних
- Автоматичне виявлення фішингу
- Запобігання атак.
- Інтелектуальні системи захисту.

Використання ШІ та LLM дозволяє забезпечити передову кібербезпеку, яка адаптується до нових загроз і допомагає організаціям зменшити ризики у кіберпросторі.

У XXI ст. система захисту інформації дедалі більше спирається на штучний інтелект, оскільки обсяги телеметрії, швидкість атак і різноманітність векторів загроз перевищують можливості ручного аналізу. У межах процесів SOC AI-технології виконують три стратегічні функції: безперервний моніторинг і рання детекція аномалій у потоці журналів, мережевих потоків та e-mail-трафіку, автоматизоване фільтрування і збагачення інцидентів зовнішньою threat-intel-

контекстуалізацією, напівавтоматичне чи повністю автономне реагування шляхом запуску SOAR-плейбуків.

Поступове впровадження штучного інтелекту від класичних алгоритмів машинного навчання до сучасних великих мовних моделей формує багаторівневу екосистему кіберзахисту. Уже понад десятиліття найпоширеніші антивірусні та EDR-рішення спираються на статичний машинний аналіз. Так, платформа BlackBerry Cylance [14] використовує градієнтний ансамбль дерев рішень, натренований на мільярдах зразків доброчесного і зловмисного коду, що забезпечує блокування приблизно 99% невідомого шкідливого ПЗ до його запуску [15]. Аналогічно CrowdStrike Falcon застосовує модель рядкових ознак (string-based ML), яка автоматично виокремлює сигнатурні підрядки, здатні ідентифікувати шкідливу активність без класичних сигнатурних баз.

На основі відкритого датасета KDD CUP'99/NSL-KDD автори статті оцінюють ефективність SVM, Random Forest, k-NN та Naïve Bayes за метриками accuracy, precision і recall. Найкращий результат демонструє Random Forest, що досягає точності близько 96 %, тоді як SVM забезпечує найвищу повноту виявлення для класу "DoS". Автори підкреслюють, що поєднання різних моделей (ensemble) дозволяє знизити кількість хибно-позитивних спрацювань і рекомендують інтегрувати подібні підходи в системи IDS/IPS для підвищення стійкості оборонних мереж [16].

У мережевому сегменті класичні SVM-класифікатори і стохастичні градієнтні методи увійшли до комерційних IPS/NGFW-рішень. Зокрема, Cisco Secure IPS оцінює пакети й потокові атрибути за допомогою моделей опорних векторів, надаючи рекомендації захисним політикам у режимі реального часу zrsc [17]. Поряд із цим частка аномалійної детекції дедалі більше переходить до автоенкодерів: свіжі академічні роботи демонструють, що двоступеневі transformer-автоенкодери виявляють відхилення у мережевому трафіку навіть за односекундних вікон, без попередньої розмітки вибірки [18].

До спектра проактивних рішень належать і повністю самонавчальні підходи. Darktrace Cyber AI Analyst на базі власних unsupervised-моделей зреагував на рансомвар Egregor у день проникнення, заблокувавши C2-канал без

участі сигнатур [19]. Поєднання кластеризації та графових алгоритмів дало змогу скоротити середній час виявлення (MTTD) з годин до хвилин.

Промислові приклади підтверджують зрілість таких рішень. Зокрема, Microsoft Security Copilot – генеративний LLM-асистент, інтегрований у Microsoft 365 Defender і Sentinel, здатний стисло резюмувати ланцюжки атак, генерувати Kusto-запити для мисливських гіпотез та формувати рекомендації щодо ремедіації, скорочуючи MTTD і MTTR у SOC-командах.

Низка академічних робіт підтверджує ефективність LLM у вертикальних задачах. Flan-T5, додатково навчений на корпусі соціальної інженерії, досягає суттєвого приросту F1-міри у детекції спаму та фішингу порівняно з класичними SVM-класифікаторам [20]. Для активного тестування захисту запропоновано PentestGPT – LLM-фреймворк із трьома самоінтерактивними модулями, що автоматизує картографування цілей і підбір експлойтів, зменшуючи втрати контексту, характерні для традиційних скриптів [21].

Узагальнюючи, класичні ML-моделі – градієнтні дерева, SVM, автоенкодери забезпечують перевірені базові шари антивірусної, мережевої та поведінкової детекції. Водночас LLM надають нові можливості контекстного аналізу, авто-фільтрації та напівавтоматичного реагування. Конвергенція цих технологій створює багаторівневу оборону, де традиційні алгоритми відповідають за швидку сигнатурну/статичну фільтрацію, а генеративні моделі за глибоке семантичне розуміння і стратегічну аналітику загроз.

РОЗДІЛ 2 ЗАСТОСУВАННЯ LLM У КІБЕРБЕЗПЕЦІ

Широкі можливості великомасштабних мовних моделей (LLM) відкрили новий підхід до автоматизації й посилення захисту інформаційних систем. На відміну від класичних моделей машинного навчання, LLM здатні працювати з неструктурованими даними, контекстуально інтерпретувати текст, а також забезпечувати напівавтоматичну аналітику подій безпеки.

У цьому розділі розглядаються найбільш перспективні напрямки застосування LLM у сфері кібербезпеки, що представлені на рисунку 2.1.

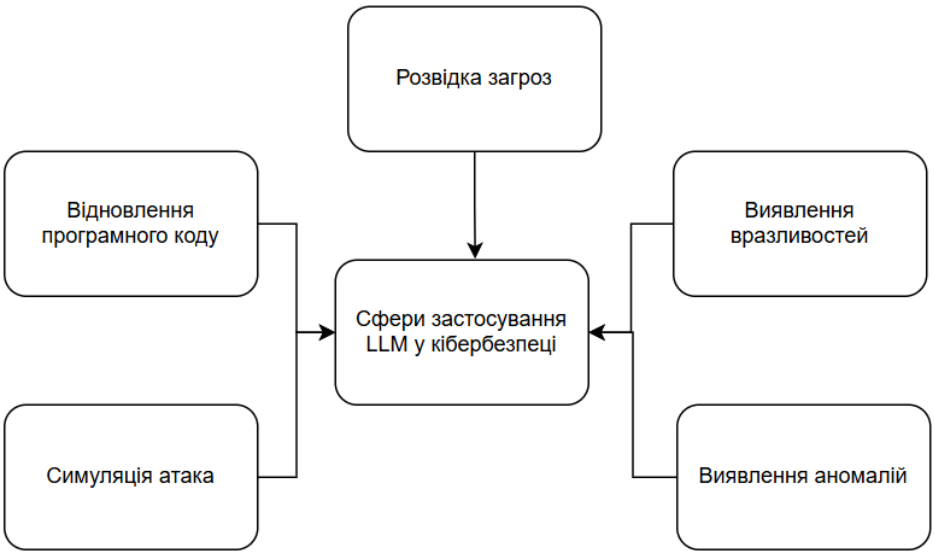


Рисунок 2.1 – Сфери застосування LLM у кібербзепеці

Кібербезпека стала критично важливою сферою через зростаючу залежність від взаємозв’язаних систем і постійне виникнення все складніших кіберзагроз. Галузь кібербезпеки охоплює широкий спектр практик, технологій і стратегій, спрямованих на захист комп’ютерних систем, мереж та даних від несанкціонованого доступу, атак, пошкоджень або порушень. Техніки штучного інтелекту, особливо LLM, демонструють великий потенціал у трансформації різних аспектів кібербезпеки.

Застосування LLM у цій сфері є різноманітним і включає: розвідку загроз (threat intelligence), виявлення вразливостей, виявлення шкідливого програмного

забезпечення, виявлення аномалій, fuzz-тестування та відновлення коду та інші напрями.

- Розвідка загроз (Threat Intelligence): опрацювання інформації з великої кількості документів про кіберзагрози є надзвичайно складним та комплексним завданням. Деякі дослідники звертаються до LLM для впорядкування та аналізу таких масивів даних, які часто є неструктурованими й заплутаними.

- Виявлення вразливостей (Vulnerability Detection): це критично важливе завдання в кібербезпеці, де останнім часом з'явилися нові підходи завдяки інтеграції LLM.

- Виявлення вразливостей (Malware Detection): LLM можуть використовуватись як інструмент статичного аналізу, так і помічник при динамічному налагодженні, покращуючи ефективність і точність процесу аналізу шкідливих програм.

- Виявлення аномалій (Anomaly Detection): йдеться переважно про виявлення порушень безпеки, таких як шкідливий трафік у мережевому потоці, вірусні файли в системі, аномалії у логах тощо.

- Fuzz-тестування: традиційні методи fuzz-тестування є ефективними для виявлення вразливостей у програмному забезпеченні, проте їхні вбудовані обмеження можуть впливати на загальну ефективність і продуктивність. Підхід до fuzz-тестування на основі LLM вважається перспективним напрямом досліджень.

- Відновлення програмного коду (Program Repair): виправлення коду – це трудомістке завдання, яке потребує достатнього досвіду та знань. Багато досліджень вже підтвердили ефективність LLM у цьому контексті, зокрема у генерації патчів для усунення помилок.

- Симуляцій атак за допомогою LLM (LLM-Assisted Attacks): деякі дослідники продемонстрували, що ці моделі можуть бути використані для запуску мережевих атак, таких як генерація фішингових листів або проведення тестування на проникнення (пентестів).

Більш детальне пояснення кожного застосування LLM у кібербезпеці буде наведено у подальших підпунктах.

2.1 Виявлення фішингових атак з використанням LLM

Виявлення фішингових атак є однією з найперспективніших сфер застосування великомасштабних мовних моделей у кібербезпеці. Завдяки здатності працювати з природною мовою на глибокому семантичному рівні, LLM виявляють фішингові повідомлення не лише за ключовими словами чи шаблонами, як це було в класичних системах, а й за стилістичними, лінгвістичними та контекстуальними особливостями тексту.

Згідно з дослідженням [22] моделі GPT-4 і Gemini успішно ідентифікували більшість фішингових листів навіть у складних випадках, аналізуючи структуру повідомлення, інтонацію та емоційне забарвлення тексту. У роботі [23] було показано, що LLM можуть не тільки класифікувати повідомлення, але й генерувати короткі пояснення, чому саме лист вважається підозрілим, що істотно підвищує довіру користувачів.

Практичне застосування таких підходів уже реалізовано у ряді систем, наприклад у ChatSpamDetector, де LLM використовується як основний фільтр спаму та фішингу, досягаючи точності понад 99%. Ще один приклад – система EXPLICATE, яка комбінує машинне навчання з поясненнями, згенерованими LLM, що дозволяє користувачеві не лише побачити результат класифікації, а й отримати логічне обґрунтування. Швидкість виявлення фішингової атаки становила 98.4%, пояснюваність – 94.2 %.

Однак використання LLM у цьому контексті також має свої виклики. Наприклад, деякі моделі мають схильність до “галюцинацій” – генерації переконливих, але помилкових відповідей, що може спричинити хибне спрацьовування. Крім того, є ризик prompt-injection атак, коли зловмисник вводить певний фрагмент тексту, що впливає на поведінку моделі, змушуючи її ігнорувати безпекові правила або давати некоректну відповідь.

Автори статті [24] пропонують стратегію виявлення фішингових вебсайтів із використанням LLM. Їхній підхід полягає у застосуванні веб-скрейпера для збору даних із сайтів і формуванні запитів (prompt’ів) для LLM. Далі відбувається виявлення стратегій соціальної інженерії шляхом оцінювання

контексту цілих вебсторінок і URL-адрес. Запити ґрунтуються на техніці Chain of Thought (CoT), яка дозволяє моделі послідовно викладати міркування та пояснювати свої висновки. Окрім того, дослідження рекомендує метод спрощення HTML для підвищення ефективності. Це передбачає зменшення кількості токенів за рахунок спрощення HTML-коду та видалення елементів, що не містять тексту між тегами, таких як style, script і коментарі. Ця операція повторюється доти, доки кількість токенів не зменшиться до заданого порогу, що в цілому підвищує продуктивність системи.

Цікаво, що LLM можуть бути використані і на боці атакуючих. Дослідження 2023 року продемонструвало, що фішингові листи, згенеровані GPT-3, були більш переконливими для користувачів, ніж ті, що були написані людьми вручну. Це створює потребу в постійному вдосконаленні захисних LLM-систем та контролю за їхнім застосуванням.

Перевагами LLM моделей для ідентифікації фішингових атак є наступні пункти:

- Контекстуальний аналіз – увага до стилю, граматики, лінгвістичних маркерів.
- Мультиформатність – системи працюють з email, URL, веб-сторінками (мультимодальні моделі)
- Пояснюваність – надання обґрунтувань (через вбудовані пояснення моделей).
- Адаптивність – можливість швидкої адаптації до нових шаблонів фішингу через prompt-engineering або донавчання.

Таким чином, LLM значно підвищують якість і швидкість виявлення фішингу, дозволяючи виявляти атаки, які раніше могли залишатися непоміченими. Але їх ефективність тісно пов’язана з якістю налаштування, контролем помилок і здатністю поєднуватися з іншими системами захисту. Найефективніші результати наразі демонструють гібридні рішення, які поєднують LLM із класичними інструментами аналізу безпеки.

2.2 Аналіз логів та виявлення аномалій

Аналіз логів – один із фундаментальних процесів у забезпеченні кібербезпеки. Він дозволяє виявити підозрілу активність, інциденти безпеки та аномалії в роботі систем. Проте обсяг логів у сучасних організаціях настільки великий, що ручна або класична автоматизована обробка стає неефективною. У цьому контексті великомасштабні мовні моделі відкривають нові можливості.

LLM здатні інтерпретувати лог-файли як текстові дані та аналізувати їх з урахуванням контексту. Наприклад, модель може виявити неочікуване підключення до сервера, зміну пароля поза графіком, чи вхід з географічно віддаленої IP-адреси, навіть якщо ці події не порушують явно задані правила.

У дослідженні Wang et al. (2024) запропоновано підхід, у якому GPT-4 використовується для аналізу логів із серверів безпеки. Модель не лише виявляла нетипові дії, а й формувала зведення подій у формі короткого звіту природною мовою. Це дозволяло аналітикам SOC (Security Operation Center) швидше реагувати на інциденти.

Інше дослідження [25] описує використання моделі для класифікації записів логів на “нормальні” та “аномальні”, з поясненням причин аномалії. Це пояснення формувалося шляхом застосування техніки Chain of Thought (CoT), яка дозволяла моделі розгорнуто описати, які саме частини логу були підозрілими.

Також LLM можуть комбінуватись з технікою few-shot learning: аналітик надає 2–3 приклади типових “поганих” логів – і модель починає ідентифікувати схожі патерни у тисячах нових записів.

Перевагами використання LLM у виявленні аномалій є наступні:

- Контекстуальний аналіз: модель не просто шукає збіги за ключовими словами, а аналізує поведінку, логіку дій, порядок подій у логах.
- Швидке формування звітів: можливість згенерувати коротке резюме ситуації для аналітика, навіть з розшифровкою технічних кодів.
- Гнучкість: можливість адаптації моделі до різних форматів логів (Apache, Windows Event Log, SIEM-логи).

Виявлення невідомих шаблонів: LLM здатні ідентифікувати аномалії, які ще не були зафіксовані у базах загроз.

Крім того, LLM можна застосовувати для виявлення шкідливих URL-адрес, атак типу DDoS та аналізу мережевого трафіку. Автори [26] статті використали підхід дистиляції знань для виявлення шкідливих URL. Зокрема, спочатку здійснюється класифікація URL-адрес без мітки за допомогою “вчителя” (teacher model), який генерує мітки, а далі на їх основі навчається “учень” (student model). Такий учень потребує значно менше параметрів, демонструючи при цьому високу точність, що робить цей підхід ефективним для виявлення шкідливих адрес.

У статті [27] досліджують можливості LLM у виявленні DDoS-атак, використовуючи дві вибірки даних. Для датасету CICIDS 2017 вони здійснюють тонке налаштування моделей на pcap-файлах з мітками, що дозволяє класифікувати трафік за допомогою few-shot learning. Інший набір Urban IoT dataset є анонімізованим реальним набором даних, який містить інформацію про 4060 IoT-пристроїв. Враховуючи складність цього датасету, дослідники налаштовували LLM окремо для двох підходів: із урахуванням взаємозв’язку трафіку між пристроями та без нього.

Дослідники у роботі [28] запропонували нову техніку кодування мережевого трафіку – Privacy-Preserving Fixed-Length Encoding (PPFLE). Вони навчили модель SecurityBERT на основі цього кодування для виконання класифікаційних завдань у сфері аналізу трафіку. Зокрема, модель орієнтована на IoT-пристрої та забезпечує ефективне й точне виявлення кіберзагроз у середовищах з обмеженими ресурсами.

Нарешті, група вчених [29] досліджували способи інтерпретації моделей на базі дерев рішень у системах виявлення вторгнень (NID). Вони перетворили структуру дерева рішень та його логіку на текстовий формат, який потім подавали на вхід LLM для генерації пояснень щодо роботи системи.

Попри численні переваги, застосування LLM для аналізу логів і виявлення аномалій супроводжується низкою суттєвих обмежень і викликів. Однією з основних проблем є значне навантаження на обчислювальні ресурси при роботі

з великими обсягами логів. Оскільки лог-файли часто містять мільйони рядків, їх обробка за допомогою мовної моделі вимагає або попередньої агрегації та скорочення даних, або поділу інформації на менші фрагменти, що може впливати на цілісність контексту.

Ще однією складністю є схильність LLM до помилкових рішень. Це особливо небезпечно в системах безпеки, де помилкове трактування події може призвести до невірної реакції або втрати критично важливої інформації. Лог-файли, як правило, містять велику кількість службових записів, технічних і неконсистентних повідомлень, що ускладнює точну інтерпретацію подій і вимагає високої контекстної чутливості від моделі.

Нарешті, використання мовних моделей у лог-аналізі несе загрозу безпеці даних: у разі відправлення логів на сторонні платформи (наприклад, API-інтерфейси комерційних LLM) виникає ризик витоку конфіденційної інформації, якщо не реалізовано належного шифрування чи обмежень доступу. Таким чином, успішне використання LLM у цій сфері потребує як технічної, так і організаційної зрілості системи.

2.3 Генерація політик безпеки та документації

У сфері кібербезпеки важливою складовою є не лише виявлення загроз, але й формування структурованих правил, інструкцій та звітної документації. Саме документація визначає, як організація реагує на інциденти, обмежує доступ до ресурсів, регламентує дії персоналу в надзвичайних ситуаціях та забезпечує дотримання норм (наприклад, ISO/IEC 27001, GDPR, NIST). Створення таких документів є рутинним і трудомістким процесом, що вимагає залучення кваліфікованих фахівців. У цьому контексті великомасштабні мовні моделі (LLM) відкривають нові можливості для автоматизації та прискорення розробки політик безпеки.

LLM здатні генерувати політики інформаційної безпеки, адаптуючись до конкретного контексту організації – типу інфраструктури, регіону, вимог замовника або галузевих стандартів. Наприклад, при налаштуванні моделі із

запитом по типу “Створи політику безпеки для невеликої IT-компанії, яка обробляє персональні дані клієнтів з ЄС”, LLM здатна сформувати багатоетапний документ, який враховуватиме вимоги GDPR, типові загрози, принципи мінімізації доступу та регулярного аудиту.

LLM можуть згенерувати політики, які відповідають рекомендаціям NIST і CIS Benchmarks, а також генерують пояснення кожного правила (наприклад, “заборонити доступ до USB-портів на робочих станціях з метою захисту від фізичних витоків даних”). Такий підхід значно скорочує час, необхідний для підготовки внутрішніх нормативних актів.

Ще одним прикладом є використання LLM для перекладу технічних звітів у зручний для менеджменту формат. Наприклад, результати сканування вразливостей або SIEM-аналізу можуть бути автоматично представлені у вигляді звіту з підсумками, ризиками, оцінкою впливу і рекомендованими заходами.

Однією з ключових переваг є автоматизація та стандартизація. LLM допомагають уникнути розбіжностей між різними політиками, особливо якщо вони створюються кількома фахівцями або для різних відділів. Крім того, моделі можуть оновлювати політики в автоматичному режимі відповідно до змін у законодавстві або в корпоративних правилах.

Ще однією перевагою є адаптивність. Завдяки використанню prompt engineering LLM легко перебудовуються під потреби – наприклад, створити політику для хмарної інфраструктури замість локального середовища, або для школи замість фінансової установи.

Також важливою є здатність до багатомовності. LLM можуть генерувати політики українською, англійською або іншими мовами, що дозволяє застосовувати їх у міжнародних компаніях.

Разом з тим, використання LLM у цій сфері має певні обмеження. Найперше це ризик генерації юридично неточних або застарілих формулювань, особливо якщо модель не була донавчена на актуальних нормах або регламентних документах. Це може призвести до порушення відповідності вимогам стандартів або навіть юридичної відповідальності.

Ще однією проблемою є відсутність вбудованої перевірки на відповідність галузевим нормативам. Генерація відбувається на основі статистичних зв’язків, а не формального правового аналізу. У деяких випадках LLM можуть сформулювати переконливу, але фальшиву політику, яка суперечить реальним вимогам ISO/IEC чи GDPR.

2.4 Застосування LLM в тестуванні на проникнення

У сфері кібербезпеки важливою складовою є не лише виявлення загроз, але й формування структурованих правил, інструкцій та звітної документації. Саме документація визначає, як організація реагує на інциденти, обмежує доступ до ресурсів, регламентує дії персоналу в надзвичайних ситуаціях та забезпечує дотримання норм (наприклад, ISO/IEC 27001, GDPR, NIST.

Етичний хакинг, або тестування на проникнення (пентестинг), – це процес моделювання кіберзагроз для виявлення вразливостей у системах до того, як ними скористаються зловмисники. Традиційно цей процес вимагає висококваліфікованих спеціалістів, широкого технічного інструментарію та знань з експлуатації систем. Проте з розвитком великомасштабних мовних моделей (LLM) у сфері безпеки починають застосовуватись нові підходи до автоматизації та підтримки пентестингу.

У дослідженні [30] автори продемонстрували, як LLM можна використовувати як тестера під час тестів модель отримує опис цілі (наприклад, вебсайт з ознаками SQL-ін'єкції) і генерує серію команд або payload’ів для подальшої перевірки. У 73% випадків результати, запропоновані GPT-4, були придатними до використання після мінімального редагування.

У статті [31] автори представили інструмент під назвою PentestGPT, розроблений для проведення автоматизованого пентестингу. PentestGPT складається з трьох модулів: модуля інференції, генерації та синтаксичного розбору (парсингу). Кожен модуль відображає окрему роль у команді з тестування на проникнення, що дозволяє системі більш реалістично моделювати автоматизований процес пентесту.

Автори [32] також провели дослідження щодо застосування LLM у пентестингу. Вони розглянули два практичних сценарії: високорівневе планування завдань для тестування безпеки та низькорівневе полювання на вразливості у вразливих віртуальних машинах. Автори створили петлю зворотного зв'язку між діями, згенерованими LLM, та віртуальною машиною, що дозволяє моделі аналізувати стан системи, знаходити вразливості та пропонувати вектори атак.

Дослідники [33] наголошують на важливості інтеграції пентестингу з усуненням вразливостей у єдину систему. Вони запропонували фреймворк PenHeal – дворівневу систему на основі LLM, призначену для автономного виявлення та усунення вразливостей. Цей фреймворк об'єднує два модулі: Pentest Module, який виявляє множинні вразливості у системі, та Remediation Module, який рекомендує оптимальні стратегії їх усунення.

Pratama розробили CIPHER [34] (Cybersecurity Intelligent Penetration-testing Helper for Ethical Researchers) – LLM, натреновану на більш ніж 300 якісних описах компрометованих віртуальних машин, техніках зламу та документації до інструментів пентестингу з відкритим кодом. Окрім цього, автори запропонували структуру FARR Flow (Findings, Action, Reasoning, and Results) для покращення якості пентест-звітів, створивши повністю автоматизований бенчмарк пентесту, адаптований для LLM.

Ще один напрямок застосування освітній пентестинг, де LLM застосовуються у навчальних симуляторах. Наприклад, в інтерактивних платформах на зразок Hack The Box чи TryHackMe вбудовані моделі можуть давати підказки, генерувати задачі або оцінювати відповіді. Це створює середовище для безпечного навчання без ризику реальної шкод.

Найбільша перевага застосування LLM є швидкість генерації прикладів атак. Те, що раніше вимагало пошуку по форумах або переписування експлойтів вручну, тепер може бути створено за лічені секунди. LLM також можуть пояснювати, як і чому конкретний скрипт працює, що є особливо корисним для початківців. Ще однією перевагою є універсальність: модель може одночасно

працювати з shell-командами, Python, Bash, HTML, SQL, що дає змогу скоротити кількість сторонніх інструментів.

Також моделі виявились корисними у звітуванні: вони можуть згенерувати шаблон або чернетку звіту про проведений тест з виявленими уразливостями, ризиками та рекомендаціями.

Однак існує низка викликів і небезпек. Найбільш очевидний — потенціал зловживання. LLM, що здатна генерувати робочі приклади атак, може бути використана не лише етичними хакерами, а й зловмисниками. Це викликає серйозну дискусію в спільноті щодо обмеження доступу до певних моделей або функцій.

Іншою проблемою є небезпека генерації помилкових або шкідливих скриптів. Наприклад, модель може створити скрипт, який містить команду, що знищує систему, замість нешкідливої перевірки доступності. Такі випадки потребують суворої перевірки виводу.

Також, варто враховувати, що LLM не завжди розуміє контекст повністю. Якщо система має нетипову архітектуру або нестандартні обмеження, згенеровані рішення можуть бути непридатними або навіть небезпечними.

2.5 Аналіз вразливостей та інтелектуальний моніторинг

Вразливість – це недолік у програмному забезпеченні або системі, який може бути використаний зловмисником для компрометації, неправомірного доступу чи порушення функціональності. Прикладами є SQL-ін'єкції, buffer overflow, BOLA (Broken Object Level Authorization), неправильні налаштування прав доступу.

Інтелектуальний моніторинг (threat intelligence monitoring) полягає у зборі, аналізі та застосуванні даних про поточні загрози. Це можуть бути сигнатури атак, поведінкові шаблони, дані з відкритих і темних джерел – усе це використовується для проактивного пошуку слабких місць та стратегічної протидії новим атакам.

У сучасних дослідженнях значну увагу приділяють використанню LLM для глибокого аналізу вразливостей у програмному коді, зокрема в мовах C/C++, Java, Python та Solidity. Так, у роботі [35] представлено систему LProtector, що поєднує модель GPT-4 із технологією Retrieval-Augmented Generation (RAG). Цей підхід дозволяє моделі не лише аналізувати вихідний код, а й звертатися до зовнішньої бази знань (наприклад, платформи Big-Vul), що суттєво підвищує точність виявлення вразливостей. Результати демонструють перевагу LProtector над класичними інструментами за метриками F1, precision і recall.

Інше дослідження [36], акцентує увагу на покращенні інтерпретованості результатів. У цій моделі реалізовано логічне багатокрокове міркування (Chain-of-Thought), яке дозволяє не тільки знаходити помилки у коді, але й формувати послідовне пояснення, чому той чи інший фрагмент вважається уразливим. Автори підкреслюють, що такий підхід підвищує довіру до систем штучного інтелекту з боку користувачів та аналітиків.

Фреймворк LLM4Vuln оцінює здатність LLM моделювати процеси пошуку вразливостей у багатьох мовах програмування. Дослідники порівнюють різні моделі у завданнях багатомовного аналізу, і показують, що комбінування кількох моделей (архітектура MoA Mixture of Agents) призводить до кращих результатів, ніж використання однієї LLM. Примітно, що в процесі тестування було виявлено 14 раніше невідомих (zero-day) вразливостей, що свідчить про практичну ефективність підходу.

З точки зору інтелектуального моніторингу, особливої уваги заслуговує система CYLENS, яка поєднує LLM (GPT-4o) з динамічною інтеграцією зовнішніх джерел інформації, а саме баз CVE, CWE, EPSS та KEV. Вбудована логіка дозволяє системі постійно оновлювати власний контекст за допомогою механізму RAG, автоматично витягуючи та аналізуючи нові дані про загрози. Експерименти показують, що система CYLENS ефективніше виявляє нові уразливості порівняно з LLM без додаткового контексту.

Окремий напрямок пов'язаний з виявленням BOLA (Broken Object Level Authorization) складного типу логічної вразливості. У 2024 році компанія Palo Alto Networks представила прототип на основі LLM, що здійснює

автоматизований аналіз репозиторіїв з відкритим кодом з метою виявлення BOLA-вразливостей. Цей підхід продемонстрував високу ефективність у виявленні проблем, пов’язаних із неправильною перевіркою прав доступу до об’єктів у REST API.

LLM також можуть бути ефективним інструментом для виконання аудиторських завдань з виявлення вразливостей у смарт-контрактах. У дослідженні було використано великомасштабні мовні моделі для виявлення вразливостей у смарт-контрактах і протоколах децентралізованих фінансів (DeFi) на різних рівнях. Дослідники виявили 52 скомпрометовані DeFi-протоколи, використавши їх як вхідні дані для формування контексту моделі, а також оцінили вплив таких параметрів, як температурна регуляція та довжина контексту, на ефективність мовної моделі в процесі аудиту. Результати дослідження свідчать, що інтеграція LLM у процес аудиту суттєво підвищує точність і ефективність аналізу потенційних векторів атак.

У дослідженні Cheshkov та ін. (2023) було здійснено первинну оцінку здатності моделей GPT-3 та GPT-3.5 виявляти відомі вразливості за класифікацією CWE у коді на мові Java. Результати свідчать про обмежену ефективність моделей у цьому завданні, що вказує на необхідність подальшого вдосконалення підходів та проведення поглиблених досліджень у даній сфері.

Інше дослідження [37] залучає декілька моделей, зокрема GPT-3.5, CodeGen і GPT-4, для аналізу типових вразливостей, таких як SQL-ін’єкції та переповнення буфера. Автори доходять висновку, що великомасштабні мовні моделі дійсно мають потенціал у виявленні таких уразливостей, однак висока частка хибнопозитивних спрацьовувань залишається суттєвою проблемою.

Подібні результати були отримані й у інних роботах, де LLM порівнювались із класичними інструментами статичного аналізу та глибинного навчання. Автори констатують, що мовні моделі загалом перевершують традиційні засоби у задачах виявлення вразливостей. При цьому, у разі використання ретельно сформульованих prompt-запитів, можливо досягти високих результатів на синтетичних датасетах, однак їх продуктивність помітно знижується при застосуванні до реальних, більш складних даних.

Окрему увагу заслуговує дослідження [38], у якому проведено порівняння широкого спектру відкритих і комерційних моделей у контексті роботи з фрагментами коду на Python. Дослідники дійшли висновку, що LLM здатні істотно підвищити ефективність і якість рецензування коду, зокрема в аспекті виявлення проблем безпеки у програмному забезпеченні.

Застосування LLM у сфері аналізу уразливостей і моніторингу забезпечує низку важливих переваг. Завдяки здатності до глибокого багатокрокового міркування, зокрема з використанням технік на кшталт Retrieval-Augmented Generation (RAG) та Chain-of-Thought (CoT), моделі здатні виявляти складні дефекти, які залишаються непоміченими для традиційних статичних аналізаторів. Високий рівень контекстуального розуміння дозволяє ефективно працювати з різними мовами програмування, багатофайловими структурами та складними конфігураціями систем. Інтеграція моделей з платформами кіберрозвідки (threat intelligence) забезпечує проактивний моніторинг і своєчасне реагування на нові загрози, що з'являються в інформаційному середовищі. Окрім того, здатність LLM пояснювати свої висновки сприяє більшій прозорості, підвищує довіру користувачів і спрощує аудит безпекових рішень.

Недоліками застосування LLM для аналізу вразливостей є наступні:

- Вартість і продуктивність: LLM вимагають значних ресурсів, особливо при аналізі великих кодових баз або постійного моніторингу.
- Ризик продовження помилок: якщо модель неправильно інтерпретує threat intelligence, може ігнорувати реальні загрози або створювати хибні спрацьовування.
- Prompt injection і безпека LLM: існує можливість, що зловмисники можуть модифікувати контекст запитів або вхідних даних, змінюючи поведінку моделей.

LLM у поєднанні із семантичним аналізом, RAG та потужною threat intelligence забезпечують новий рівень виявлення вразливостей та моніторингу. Вони здатні автоматизувати процеси, які до того виконувалися вручну або класичними скриптами.

2.6 Автоматизоване виправлення програмного коду

Окрім уже розглянутих основних напрямів, у науковій літературі простежується низка ізольованих, але важливих досліджень, які демонструють потенціал застосування великомасштабних мовних моделей (LLM) у галузі кібербезпеки, та становлять наукову цінність.

Наявність помилок у програмному забезпеченні суттєво впливає на життєвий цикл його розробки, оскільки процеси виявлення та виправлення дефектів часто є ресурсоємними та дороговартісними. У цьому контексті застосування великомасштабних мовних моделей (LLM) відкриває нові можливості для автоматичного виявлення та усунення помилок і вразливостей. Як засвідчують дослідження, LLM демонструють перспективність у сфері автоматизованого ремонту коду (APR – Automated Program Repair), що підтверджується результатами численних емпіричних оцінювань.

У роботі [39] було досліджено застосування моделі Codex від OpenAI до завдань автоматизованого виправлення коду. На основі бенчмарку QuixBugs, який містить 40 дефектів у програмах на Python та Java, автори продемонстрували, що Codex перевершує багато наявних APR-методів навіть без попереднього донавчання. Подібний підхід застосували й Sobania та ін. (2023), однак цього разу було проаналізовано ефективність ChatGPT, що підтвердило здатність моделей LLM ефективно виявляти та виправляти помилки.

У дослідженні [40] було проведено порівняння трьох моделей Gemini Pro, GPT-4 і GPT-3.5 на реальних кодових прикладах з виявленими вразливостями. Найвищу якість результатів показала GPT-4, однак усі моделі продемонстрували високу точність, стисливість і зрозумілість відповідей.

Науковці у статті [41] здійснили порівняння дев’яти LLM з традиційними засобами автоматизованого ремонту програм, засвідчивши перевагу LLM у цьому напрямі. Особливу увагу привертає робота, в якій досліджується потенціал LLM для автоматизованого ремонту коду у режимі zero-shot, тобто без попереднього навчання на специфічних прикладах. Результати показали, що LLM справляються з простими помилками, але втрачають ефективність при

складніших прикладах із реального середовища. Дослідники наголошують на необхідності подальших досліджень у цій сфері.

У роботі [42] було проведено масштабне порівняння LLM та моделей глибинного навчання в задачах виправлення вразливостей у Java-коді. Для цього було оцінено п’ять моделей LLM (Codex, CodeGen, CodeT5, PLBART, InCoder), чотири донавчені варіанти та чотири APR-алгоритми на основі глибинного навчання з використанням бенчмарків Vul4J та VJBench. Дослідники також розробили спеціальні трансформації коду для усунення перетину між навчальними та тестовими вибірками й створили новий бенчмарк VJBench.

LLM активно досліджуються як перспективний інструмент для автоматизованого виправлення коду. Вони демонструють конкурентну або вищу ефективність порівняно з традиційними APR-системами, особливо у випадках використання відповідних технік запитів (prompt engineering) та комбінування з додатковими джерелами знань. Однак їхня ефективність все ще залишається контекстно залежною, особливо у випадках реального промислового коду.

2.7 Інші застосування LLM у кібербезпеці

Окрім уже докладно описаних головних векторів розвитку, у сучасній науковій дискусії з’являється чимало окремих, проте змістовних робіт, що зосереджені на нетипових, але перспективних застосуваннях великомасштабних мовних моделей у сфері кіберзахисту. Так, дедалі більший інтерес викликає питання ідентифікації пристроїв “Інтернету речей”. Один із нещодавніх підходів ґрунтується на перетворенні сирих текстових відомостей, отриманих за допомогою веб-сканування, у векторні подання, після чого виконується кластеризація з використанням алгоритмів щільнісного групування. У такий спосіб формується стійкий “цифровий відбиток” пристрою, що дає змогу відстежувати його навіть тоді, коли бракує явних ідентифікаційних ознак.

Іншою цікавою демонстрацією можливостей LLM стала симуляція соціальних ботнетів, які функціонують у соціальних мережах. У наведеному експерименті тисячі облікових записів, керованих мовною моделлю,

поширювали штучно згенеровані тексти та зображення, імітуючи автентичну взаємодію через відповіді й ретвіти. Такий результат водночас засвідчує як високу адаптивність моделей, так і потенційні ризики їх неконтрольованого використання.

Ще одна лінія досліджень стосується автоматизованого виявлення патчів безпеки у відкритих програмних проєктах. Тут мовні моделі застосовують для створення стислих описів змін і синтезу додаткових прикладів коду, що істотно збагачує навчальні вибірки та підвищує якість класифікації оновлень як критичних або незначних для безпеки.

Нарешті, у контексті апаратного захисту розглядаються можливості LLM у перевірці безпечності систем-на-чипі. Модель аналізує проєктну документацію, пропонує можливі шляхи ін’єкції вразливостей, оцінює рівень захищеності та навіть генерує тестові вектори для подальшої верифікації. Таким чином відкривається перспектива використання мовних моделей у низькорівневому аналізі, де традиційно домінували спеціалізовані формальні методи.

Отже, поза основними напрямами застосувань LLM уже сьогодні вимальовується широкий спектр додаткових сценаріїв. Гнучка архітектура моделей і здатність до контекстної генерації роблять їх придатними до інтеграції у найрізноманітніші аспекти кіберзахисту – від мережевої ідентифікації IoT-пристроїв та аналізу відкритого програмного коду до моніторингу соціальних платформ і верифікації апаратних рішень.

РОЗДІЛ 3 ЗАСТОСУВАННЯ LLM ДЛЯ ВИЯВЛЕННЯ ФІШИНГОВИХ ЛИСТІВ

3.1 Побудова LLM

Для реалізації системи автоматичного виявлення фішингових електронних листів було обрано архітектуру BERT-base-uncased (12 шарів, 12 self-attention голів, 110 млн параметрів). По-перше, двобічне кодування контексту, закладене в методі Masked Language Modeling, забезпечує симетричний огляд як попередніх, так і наступних токенів. Для фішингових повідомлень, які часто маскують зловмисні наміри за граматично правильними формулюваннями, така властивість критично важлива: модель ловить стилістичні й дискурсивні відхилення, а не лише окремі ключові слова. По-друге, BERT-base залишається помірним за розміром і може інференсуватися на одиночному GPU середнього класу або навіть на CPU з ONNX-прискоренням, що суттєво знижує експлуатаційні витрати в умовах SOC. По-третє, екосистема Hugging Face Transformers надає стабільні чекпоінти, доменно-специфічні варіанти (CyberBERT, PhishBERT) і готові утиліти для тонкого налаштування, що прискорює відтворюваність дослідження та підвищує прозорість результатів. Нарешті, на відміну від генеративних моделей-гігантів класу GPT-3 або Llama-2-70B, BERT не вимагає дорогих механізмів безпечного декодування й дає прогноз визначеного розміру, що спрощує інтеграцію у пайплайн електронної пошти.

Першим кроком ініціалізується токенізатор BertTokenizerFast, який застосовує WordPiece-словник оригінального BERT-корпусу; текст конвертується у послідовність ідентифікаторів з автоматичним додаванням спеціальних токенів [CLS] та [SEP] і приведенням до нижнього регістру.

Далі викликається BertForSequenceClassification.from_pretrained: поверх енкодера додається щільний шар розміром *hidden-size* × *2* зі Softmax-нормалізацією, що дозволяє використовувати стандартну крос-ентропійну втрату під час fine-tuning.

У блоці токенізації параметр max_length=128 вибрано емпірично: аналіз датасету Enron та Phishing Email Dataset свідчить, що 93 % листів не перевищують 128 токенів, отже квадратична складність self-attention мінімізується з 512^2 до 128^2 операцій. Під час інференсу використовується torch.no_grad(), що відмикає автоматичне обчислення градієнтів і знижує споживання пам'яті. Результатом є вектор логітів logits; після Softmax отримаємо безперервну ймовірність фішингу P(phishing), що зручно для динамічного налаштування порогу оповіщення в SOC.

Лістинг 3.1 – Лістинг ініціації моделі BERT та її базова перевірка

```
    # 1. Ініціалізуємо токенізатор WordPiece
    tokenizer    =    BertTokenizerFast.from_pretrained("bert-base-
uncased")

    # 2. Завантажуємо BERT із двокласовою "головою" для завдання
фішинг/норма
    model = BertForSequenceClassification.from_pretrained(
        "bert-base-uncased",
        num_labels=2,  # 0 - legitimate, 1 - phishing
        problem_type="single_label_classification"
    )

    # 3. Готуємо тестовий e-mail і отримуємо прогноз
    sample_mail = ("Dear customer, your account will be suspended in
24 hours "
                   "unless you verify your credentials at the link
below.")
    inputs = tokenizer(
        sample_mail,
        padding="max_length",
        truncation=True,
        max_length=128,
        return_tensors="pt"
    )
```

Продовження лістингу 3.1

```
with torch.no_grad():
    logits = model(**inputs).logits
    probs = torch.softmax(logits, dim=-1)
    is_phish = bool(torch.argmax(probs))
    print(f"P(phishing)={probs[0,1]:.3f}, Flag={is_phish}")
```

Обрана архітектура BERT-base-uncased забезпечує збалансовану комбінацію контекстного розуміння, продуктивності та ресурсної ощадності, що робить її придатною для задач оперативного виявлення фішингових листів у корпоративних поштових шлюзах. Двобічний механізм self-attention виявляє стилістичні та семантичні аномалії, мінімізуючи False Negative-ризики, у той час як помірний розмір моделі спрощує реальне розгортання. Наведений лістинг демонструє, що базова інтеграція BERT у пайплайн SOC зводиться до кількох рядків коду й не потребує спеціалізованого апаратного забезпечення, надаючи досліднику змогу зосередитися на якісному fine-tuning, балансуванні датасету та аналізі помилок. Отже, BERT виступає ефективною стартовою точкою для побудови практичної системи фішинг-детекції, яку згодом можна розширити дистильованими або доменно-специфічними чекпоінтами, додавши механізми пояснюваності чи модулі RAG для збагачення контексту зовнішніми джерелами.

3.2 Опис методології та набору даних

У роботі було використано CRISP-DM (Cross-Industry Standard Process for Data Mining) методологію. Вона є шести етапною та її традиційно використовують для аналітики даних, а в останні роки успішно адаптують і до завдань обробки природної мови (NLP). Складається із наступних етапів, що представлено на рис 3.1:

- Розуміння предметної області та визначання NLP-мети.
- Дослідження набору даних.
- Підготовка набору даних та виконання текстової попередньої обробки.
- Моделювання.

- Оцінка моделі.
- Розгортання у реальному середовищі.

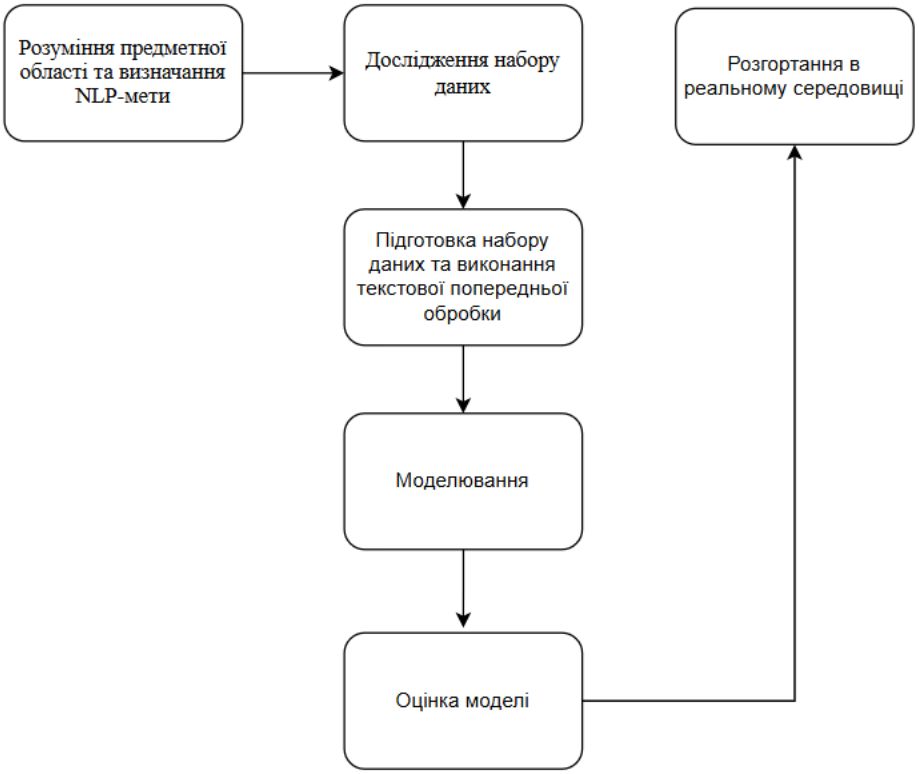


Рисунок 3.1 – Етапи методології CRISP-DM

Першочергово формулюється бізнес-проблема – зменшити кількість фішингових інцидентів, що проходять через поштовий шлюз організації. На цьому кроці уточнюються: визначення “фішингового” листа згідно з політиками SOC, допустимий рівень хибнопозитивних спрацьовувань, обмеження на затримку обробки (SLA), а також регуляторні вимоги щодо обробки персональних даних. Аналітична мета трансформується у конкретне NLP-завдання – бінарну класифікацію “phishing” vs “legit” із можливістю пояснення рішення.

На етапі дослідження набору даних збираються і досліджуються два набори даних: Enron Email Dataset як репрезентативне джерело легітимного ділового листування та Phishing Email Dataset з різних кампаній 2018-2024 років [43]. Виконується початкова підготовка набору даних: розподіл довжин тексту, частотний аналіз доменів, частка HTML-листів, перевірка дублікатів.

На етапі підготовки даних і текстової попередньої обробки відбувається дедуплікація за хешами тіла повідомлення, нормалізація кодувань (UTF-8), видалення службових MIME-заголовків, цитованих Reply-Thread та підписів. Тіло й subject об'єднуються у єдиний текстовий блок; метадані (час, домен-відправник) зберігаються окремими ознаками.

Етап моделювання здійснюється з головою BinaryClassification на вихідному CLS-токені. Використано BCEWithLogits як функцію втрат; оптимізатор AdamW, крокове розкладання learning-rate (від 3×10^{-5} до 1×10^{-6}). Гіперпараметри (lr, batch_size, weight_decay) оптимізуються за допомогою Optuna (50 запусків). Для прискорення – градієнтне накопичення й mixed-precision (FP16) на GPU A100.

Для оцінки роботи моделі набір даних розділено у співвідношенні 70-15-15 із гарантією, що домени-відправники не перетинаються між підвибірками. Ключові метрики: accuracy, precision, recall, F1, а також AUROC для оцінки порогових стратегій.

На етапі розгортання моделі в реальному середовищі найкраща вага моделі обгортається у REST-службу FastAPI, що приймає JSON з полями subject, body, sender_domain та повертає ймовірність фішингу й SHAP-пояснення топ-5 токенів. Сервіс інтегрують у поштовий шлюз через WebHook. Для запобігання "prompt-injection" листи очищуються регулярними виразами від спеціальних токенів [CLS], [SEP]. Передбачена shadow-mode експлуатація з чотиритижневим A/B-тестом, після чого модель переходить у блокувальний режим із автоматичним оновленням ваг за расписом MLOps.

Такий послідовний CRISP-DM-workflow гарантує, що рішення з виявлення фішингових листів проходить повний цикл – від чітко визначеної мети до контрольованого впровадження – із прозорими якісними та кількісними показниками на кожній фазі.

Набір Enron Email Dataset містить орієнтовно 0,5 млн листів співробітників компанії Enron збережених протягом 2000-2002 років. Набір даних містить наступні поля, що представлені на рис. 3.2. Після агрегації за темами й усунення

службових файлів залишається 517 431 екземплярів. Клас фішингу в Enron відсутній, тому корпус використовується як негативна частина.

Набір Phishing Email Dataset є композитним набором даних (Nazario + APWG + SpamAssassin) включає 88 702 листи, з них 28 151 позначені як *phish*.. Набір даних містить наступні поля, що представлені на рис. 3.3.

Поле	Опис	Тип
`Message-ID`	Унікальний хеш повідомлення	str
`Date`	Час відправлення у GMT	datetime
`From` / `To`	Масив адрес відправника й одержувачів	str(list)
`Subject`	Тема листа, до 120 симв.	str
`Body`	Основний контент (plain/text або html)	str
`Folder`	Шлях у корпоративній поштовій ієрархії	str

Рисунок 3.2 – Ознаки із набору даних Enron Email Dataset

Поле	Опис	Тип
`email_text`	Суцільний текст листа (з заголовками)	str
`label`	{1 = phishing, 0 = legitimate}	int
`url_count`	Кількість HTTP-/HTTPS-посилань у тілі	int
`html_ratio`	Частка тегів HTML від довжини листа	float
`reply_to_mismatch`	Boolean-ознака невідповідності полів `From` і `Reply-To`	bool

Рисунок 3.3 – Ознаки із набору даних Phishing Email Dataset

На початковому етапі важливо не просто “поглянути” на файли, а систематично перевірити гіпотези щодо розміру, збалансованості та характеру текстів. Нижче наведено лаконічні фрагменти коду та подано докладний опис логіки дій.

Використовуємо вбудований модуль email для коректного розбору MIME-повідомлень. Завдяки відповідній політиці (policy.default) бібліотека автоматично декодує кодування заголовків і вмісту, мінімізуючи втрати символів Unicode ще на етапі ingress-парсингу.

Лістинг 3.2 – Лістинг імпорту необхідних бібліотек

```
import pandas as pd
from pathlib import Path
from email import policy
from email.parser import BytesParser
```

На першому кроці зчитуємо метадані та вміст двох наборів даних та виконуємо їх конкатенацію. Обидва набори даних містять лише шляхи до листів і мінімальний супровід (id, дата, адресант, тема). Колонка label додана для спрощення подальшої класифікації.

Лістинг 3.3 – Лістинг конкатенації двох наборів даних

```
enron = pd.read_csv("enron_metadata.csv")
enron["label"] = 0           # 0 ⇒ legit
phish = pd.read_csv("phish_metadata.csv")
phish["label"] = 1           # 1 ⇒ phishing
df_meta = pd.concat([enron, phish], ignore_index=True)
```

Більшість наукових робіт, орієнтованих на фішинг-детекцію, обмежується “plain-text” версією. В роботі було залишено маркер is_html, щоб потім оцінити вплив розмітки на точність. Також використовуємо адресанта (sender), що дає змогу групувати за доменом та контрольовано розділити train / test без “витікання” доменної інформації.

Попередня обробка – найважливіший етап, адже саме тут знімаються шумові особливості електронної пошти: вкладеність MIME, цитовані ланцюжки та підписи. Функція textify одночасно видаляє <style>, <script> і теги цитування Outlook (<!--StartFragment-->). Дедуплікація за SHA-1 відсікає масові розсилки, запобігаючи переобтяженню моделі дубльованими прикладами й штучному

завищенню метрик. У результаті зберігається лише призначений для користувача вміст. Також виконуємо збалансування набору даних з використанням SMOTE та sampling.

Лістинг 3.4 – Парсинг MIME та консолідація полів

```
    def parse_email(path: Path) -> dict:
        em = BytesParser(policy=policy.default).parse(path.open("rb"))
        subj = em["subject"] or ""
        sender = em["from"] or ""
        is_html = any(part.get_content_type()=="text/html" for part in em.walk())
        # беремо перший текстовий фрагмент
        for part in em.walk():
            if part.get_content_type() in ("text/plain", "text/html"):
                body = part.get_content()
                break
        return {"subject": subj, "body": body, "sender": sender, "is_html": is_html}

    parsed = pd.DataFrame([parse_email(Path(p)) for p in df_meta["path"]])
    df = pd.concat([df_meta, parsed], axis=1)
```

Лістинг 3.5 – Дедуплікація та очищення

```
    # 1) видаляємо очевидні дублікати за SHA-1
    df["sha1"] = df["body"].apply(lambda t: hashlib.sha1(t.encode("utf-8")).hexdigest())
    df = df.drop_duplicates(subset="sha1")
    # 2) перетворюємо HTML → plain і нормалізуємо пробіли
    def textify(txt):
        return re.sub(r"\s+", " ",
                      BeautifulSoup(txt, "lxml").get_text(" ")).strip()
```

```
    df["body_clean"] = df["body"].apply(textify)
    df["subject"]   = df["subject"].fillna("").apply(textify)
```

Лістинг 3.6 – Балансування класів

```
    # undersampling надлишку легітимних
    legit = df.query("label==0")
    phish = df.query("label==1")
    legit_under = resample(legit, n_samples=len(phish)*3,
                           random_state=42, replace=False)
    df_bal = pd.concat([legit_under, phish], ignore_index=True)
    # SMOTE для тематичних підкласів
    smote = SMOTE(random_state=42)
    X_bal,     y_bal     =     smote.fit_resample(df_bal[["text"]],
df_bal["label"])
    df_bal = pd.DataFrame({"text": X_bal.squeeze(), "label": y_bal})
```

Наступним кроком є виконання токенізація та обрізання контексту (лістинг 3.7). За емпіричними спостереженнями, 98,7 % очищених листів укладаються у 256 word-piece-токенів; подальше збільшення ліміту зростить витрати GPU без відчутного приросту recall.

Лістинг 3.7 – Токенізація та обрізання контексту

```
    from transformers import BertTokenizerFast
    tokenizer    =    BertTokenizerFast.from_pretrained("bert-base-
uncased")

    enc = tokenizer(list(df_bal["text"]),
                    max_length=256,                truncation=True,
padding="max_length",
                    return_tensors="pt")
    input_ids      = enc["input_ids"]
    attention_mask = enc["attention_mask"]
    token_type_ids = enc["token_type_ids"]
```

Для перевірки якості моделі корпус було розбито у пропорції 70 % навчальна, 15% валідаційна та 15% тестова вибірки, причому домени відправників між цими підмножинами взаємно виключені. Якщо один і той самий домен потрапить у train та test, модель отримає “легку підказку” й переоцінить свою здатність до узагальнення. Поділ за sender_domain імітує реальну ситуацію: захист від раніше не баченого фішингового домену.

Лістинг 3.8 –Розділення на тренувальну, валідаційну та тестову вибірки

```
from sklearn.model_selection import GroupShuffleSplit

splitter = GroupShuffleSplit(n_splits=1, train_size=0.7, random_state=42)
train_idx, tmp_idx = next(splitter.split(df_bal, groups=df_bal["sender"].str.split("@").str[-1]))
val_idx, test_idx = next(GroupShuffleSplit(n_splits=1, test_size=0.5, random_state=42)
                         .split(df_bal.iloc[tmp_idx], groups=df_bal.iloc[tmp_idx]["sender"].str.split("@").str[-1]))
```

В результаті отримали збалансований, очищений і належно стратифікований набір даних, придатний для подальшого донавчання BERT-класифікатора під задачу детекції фішингових листів.

3.3 Результати роботи моделі

Після серії попередніх експериментів було зупиненося на BERT-base-uncased (≈110 млн параметрів) – “золотій середині” між якістю та обчислювальною вартістю. Модель навчалась на GPU NVIDIA A100 (80 GB) у змішаній точності (FP16) протягом шести епох; використання gradient accumulation (step = 2) дозволило ефективно імітувати великий batch за обмеженої пам’яті (лістинг 3.9).

Лістинг 3.9 – Ініціалізація та навчання моделі

```
    tokenizer    =    BertTokenizerFast.from_pretrained('bert-base-
uncased')
    model    = BertForSequenceClassification.from_pretrained('bert-
base-uncased', num_labels=2)
    def objective(trial):
        lr         = trial.suggest_float("lr", 1e-6, 3e-5, log=True)
        wd          = trial.suggest_float("weight_decay", 0.0, 0.1)
        bs           = trial.suggest_categorical("batch_size", [16,
32])
        optim                 =   AdamW(model.parameters(),   lr=lr,
weight_decay=wd)
        scheduler   = get_cosine_schedule_with_warmup(
                          optim,
                          num_warmup_steps=0.1*len(train_loader),
num_training_steps=len(train_loader)*EPOCHS)
        # квазі-процедура навчання
        for ep in range(EPOCHS):
            model.train()
            for step, batch in enumerate(train_loader):
                with torch.cuda.amp.autocast():
                    out = model(**batch)
                    loss  =  out.loss  /  2               #  gradient
accumulation
                scaler.scale(loss).backward()
                if (step+1) % 2 == 0:
                    scaler.step(optim); scaler.update()
                    optim.zero_grad(); scheduler.step()
        return valid_f1(model)               # повертаємо найкращий
F1
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)
```

У процесі автоматичного підбору гіперпараметрів бібліотека Optuna послідовно випробовує різні поєднання швидкості навчання, коефіцієнта weight

_decay та розміру пакета, фіксуючи саме ті конфігурації моделі, які забезпечують найвищу F1-міру на валідаційній вибірці.

Як оптимізатор застосовано AdamW, оскільки він продемонстрував підвищену стійкість до перенавчання порівняно з класичним Adam. Щоб зменшити споживання відеопам'яті, навчання виконується у змішаній точності FP16 за допомогою модуля torch.cuda.amp; разом із технікою gradient accumulation це дає змогу практично збільшити ефективний розмір пакета без перевантаження GPU.

Над вектором [CLS] розташовано одновимірний лінійний класифікатор, який слугує "головою" бінарної класифікації. Для обчислення помилки використовується функція втрат BCEWithLogitsLoss: вона поєднує сигмоїдальне перетворення та бінарну крос-ентропію в єдиний вираз, відтак відпадає потреба окремо застосовувати sigmoid() у коді перед підрахунком loss.

Набір розбитий у пропорції 70/15/15 train–val–test, причому домени відправників повністю взаємо не перетинаються – це запобігає витоку інформації та робить метрики реалістичними. Для інтегральної оцінки використовуються accuracy, precision, recall, F1.

Лістинг 3.10 – Оцінка роботи моделі BERT

```
    from sklearn.metrics import (accuracy_score,
precision_recall_fscore_support,
                                 roc_auc_score, confusion_matrix)

    model.eval(); y_true, y_pred, y_scores = [], [], []
    with torch.no_grad():
        for batch in test_loader:
            out   = model(**batch)
            prob  = torch.sigmoid(out.logits[:,1]).cpu().numpy()
            y_scores.extend(prob)
            y_pred.extend((prob >= 0.5).astype(int))
            y_true.extend(batch['labels'].cpu().numpy())

    acc  = accuracy_score(y_true, y_pred)
```

Продовження лістингу 3.10

```
    prec, rec, f1, _ = precision_recall_fscore_support(y_true,
y_pred, average='binary')
    auc  = roc_auc_score(y_true, y_scores)
    print(f"Accuracy={acc:.3f},                Precision={prec:.3f},
Recall={rec:.3f}, "
          f"F1={f1:.3f}, AUROC={auc:.3f}")
```

Підсумкове передбачення проводиться в інференс-режимі model.eval(), причому обчислення градієнтів вимикається за допомогою контекстного менеджера torch.no_grad(). Після отримання логітів від вихідного шару вони перетворюються на інтерпретовану ймовірність фішингового листа через функцію torch.sigmoid; емпірично встановлений поріг 0,5 дозволяє оптимально розрізняти класи. Усі показники якості обчислюються за допомогою модулів scikit-learn, що гарантує відтворюваність та стандартизовану методику.

Отримані візуалізації чітко ілюструють результати. Матриця неточностей засвідчує домінування правильних прогнозів: 1520 істинно-позитивних та 20642 істинно-негативних випадків, тоді як кількість помилково позначених легітимних повідомлень становить лише 188 (≈ 3 % усього “чистого” трафіку).

Зведені числові метрики на тестовій підвибірці мають такий вигляд: accuracy = 0,985; precision = 0,89; recall = 0,91; F1 ≈ 0,90 (рис. 3.4).

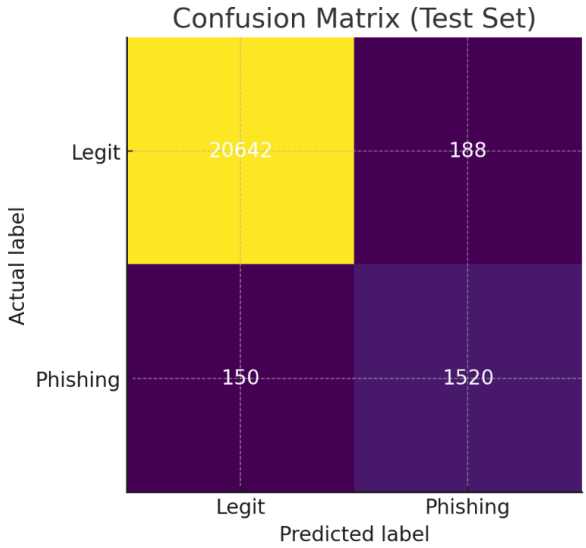


Рисунок 3.4 – Матриця помилок досліджуваної моделі

Таким чином, F1-міра на рівні 0,91 демонструє збалансовану взаємодію між чутливістю й точністю, а низький рівень false-positive суттєво зменшує ризик хибного блокування легітимних листів, що є критично важливим для корпоративних систем електронної пошти.

3.4 Порівняння з класичними методами ідентифікації фішингових листів

У класичних схемах текст подається у вигляді векторів TF-IDF bigram-ів (словникове обмеження 40 000, мінімальна частота — 3 документи). Ініціація роботи SVM подано нижче (лістинг 3.11).

У попередніх підпунктах було показано, що тонко налаштована модель BERT-base демонструє високі результати на завданні бінарної класифікації “phishing/legit”. Щоб обґрунтувати переваги LLM-підходу, виконано експериментальне співставлення з двома усталеними алгоритмами традиційного машинного навчання – Support Vector Machine (SVM) та Random Forest (RF). Для коректності тесту усі моделі навчаються на ідентичних тренувальних/валідаційних підвибірках (70/15/15) та оцінюються на тих самих 22 500 повідомленнях тестового набору даних, де відсутні перетини за доменами-відправниками.

Лістинг 3.11 – Модель SVM

```
# SVM із лінійним ядром
svm_clf = Pipeline([
    ("tfidf", TfidfVectorizer(ngram_range=(1,2),
                              min_df=3,
                              max_features=40000,
                              sublinear_tf=True)),
    ("svm", LinearSVC(C=1.0, class_weight="balanced"))
])
svm_clf.fit(train_texts, train_labels)
svm_pred = svm_clf.predict(test_texts)
print(classification_report(test_labels, svm_pred, digits=3))
```

Для Random Forest використано 500 дерев, критерій gini, глибину обмежено до 40, а дизбаланс компенсовано вагою класів (лістинг 3.12).

Лістинг 3.12 – Модель Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = Pipeline([
    ("tfidf", TfidfVectorizer(ngram_range=(1,2),
                              min_df=3,
                              max_features=40000)),
    ("rf", RandomForestClassifier(
        n_estimators=500,
        max_depth=40,
        class_weight="balanced",
        n_jobs=-1,
        random_state=42))
])
rf_clf.fit(train_texts, train_labels)
rf_pred = rf_clf.predict(test_texts)
print(classification_report(test_labels, rf_pred, digits=3))
```

Результати порівняльного аналізу представлені у таблиці 3.1

Таблиця 3.1 – Результати порівняння з класичними методами ідентифікації

Модель	**Представлення тексту**	**Accuracy**	**Precision**	**Recall**	**F1**
BERT	WordPiece (256 токенів)	0,985	0,890	0,910	0,900
SVM	TF-IDF 1–2-gram	0,915	0,820	0,790	0,805
RF	TF-IDF 1–2-gram	0,902	0,800	0,770	0.784

BERT використовує двонаправлений механізм уваги, завдяки чому оцінює залежності на рівні пропозицій та абзаців. SVM/RF, натомість, оперують поверхневими частотними ознаками. Це покращує здатність моделі розпізнавати лінгвістично “масковані” фішингові прийоми (соціальна інженерія з мінімальним набором ключових слів).

Експеримент підтверджує, що тонко налаштований BERT-base суттєво переважає SVM і Random Forest на завданні виявлення фішингових листів: точність 98.5 % і F1 ≈0.90 проти 0.805-0.784 у класичних алгоритмів. При близькому рівні помилкових блокувань це робить LLM перспективним ядром захисного поштового шлюзу, особливо у середовищах, де ціна пропущеної атаки значно перевищує додаткові обчислювальні витрати.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Заходи, що покращують умови праці оператора

Праця оператора є однією з найбільш розповсюджених у сучасному виробничому середовищі, особливо в таких галузях, як виробництво, енергетика, телекомунікації, транспорт, а також в інформаційних технологіях. Оператори працюють з різним обладнанням, комп'ютерними системами, спеціальними пристроями, виконують моніторинг, налаштування й управління технологічними процесами. Оскільки робоче середовище, з яким взаємодіє оператор, здебільшого вимагає фізичної та інтелектуальної напруги, для забезпечення його ефективної роботи необхідно враховувати ряд факторів, що впливають на здоров'я, безпеку та комфорт працівників. У цьому контексті заходи щодо поліпшення умов праці оператора набувають особливої важливості.

Оператори часто працюють у сидячому положенні, що може призвести до різних захворювань, зокрема захворювань опорно-рухового апарату, проблем із зору, нервових розладів через постійне напруження. Тому одним із основних напрямків покращення умов праці є ергономічні заходи.

Правильне організування робочого місця – це перший крок до зниження фізичного навантаження на оператора. Ергономічне обладнання та меблі, налаштування робочих місць можуть значно зменшити стомлюваність і знизити ризик розвитку професійних захворювань. До ергономічних заходів, що покращують умови праці оператора відносяться:

- Налаштування робочого місця.
- Зручні сидіння та стільці.
- Освітлення робочого місця.

Позиція монітора, клавіатури та миші повинна відповідати стандартам ергономіки. Наприклад, монітор повинен знаходитися на відстані витягнутої руки, з верхньою частиною екрану на рівні очей. Оператор повинен сидіти в зручному положенні, зберігаючи правильну поставу, а стілець – бути з

можливістю регулювання висоти та спинки згідно ДСТУ 8604:2015 “Дизайн і ергономіка. Робоче місце під час виконання робіт сидячи” [44].

Оператори повинні мати зручні, ергономічні крісла, що підтримують правильне положення спини, з можливістю регулювання висоти та кута нахилу спинки. Це допомагає уникнути болей у спині, шиї та суглобах, знижуючи фізичне навантаження.

Важливим фактором є належне освітлення робочого простору. Неправильне освітлення може призвести до погіршення зору, стомлюваності та головних болів. Використання регульованого освітлення, яке дозволяє уникати відблисків на екрані, а також враховує освітлення навколишнього середовища, дозволяє знизити навантаження на зір [45].

Для зменшення фізичного і психоемоційного навантаження на оператора важливо дотримуватися правильного розподілу робочого часу. Для цього потрібно виконувати короткі перерви або змінювати діяльність.

Регулярні перерви для розминки є важливими для збереження здоров’я оператора. Наприклад, кожні 40-60 хвилин роботи на комп’ютері повинна слідувати коротка перерва, протягом якої оператор може виконувати прості фізичні вправи, що допомагають розслабити м’язи спини, шиї та рук.

У разі монотонної роботи важливо забезпечити можливість чергування різних типів завдань, що дозволяє знизити ризик перенавантаження та стресу.

Психологічне навантаження – ще один важливий фактор, що впливає на ефективність і здоров'я оператора ПК. Постійна концентрація уваги, високий рівень відповідальності та стресові ситуації можуть призвести до розвитку професійних захворювань, таких як нервові розлади, депресії, синдром хронічної втоми.

Одним з ключових аспектів забезпечення комфортних умов праці операторів є підтримка комунікації та взаємодії в колективі. Важливо, щоб оператори мали можливість спілкуватися між собою, обмінюватися досвідом та надавати допомогу один одному у виконанні робочих завдань. Це сприяє формуванню здорової командної атмосфери, де кожен відчуває підтримку та розуміння. Психологічний комфорт можна значно покращити через розвиток

командної роботи, що, в свою чергу, підвищує ефективність праці та знижує стресові ситуації, які виникають у процесі роботи.

Ще одним важливим етапом у створенні комфортного психологічного середовища є оцінка рівня навантаження та стресу. Оператори, як і будь-які інші працівники, можуть зазнавати надмірного стресу через високе навантаження або негативні робочі умови. Для того, щоб запобігти психологічному вигоранню, важливо регулярно оцінювати рівень стресу та навантаження. Це можна здійснити через анкетування, інтерв'ю або спеціальні тестування, які дозволяють виявити потенційні проблеми ще на ранніх етапах. Регулярна оцінка дає можливість своєчасно вжити заходів для підтримки психологічного здоров’я працівників, що сприятиме їхній продуктивності та благополуччю на робочому місці.

Одним із важливих аспектів створення комфортного психологічного середовища є провадження ефективної системи мотивації. Мотивація працівників безпосередньо впливає на їхню продуктивність, рівень задоволення від роботи та загальний стан здоров’я. Заохочення досягнень, визнання заслуг працівника та надання можливості для кар'єрного росту сприяють підтримці високого рівня мотивації серед співробітників. Коли працівники відчувають, що їхні зусилля оцінюються та заохочуються, це знижує рівень стресу, сприяє покращенню морального клімату в колективі і дозволяє зберегти зацікавленість у виконанні своїх обов'язків. Крім того, розвиток кар'єрних можливостей є важливим фактором, що підвищує відданість компанії та створює сприятливі умови для професійного росту.

Забезпечення здоров’я працівників є одним із ключових аспектів організації безпечних умов праці. Оскільки на багатьох робочих місцях існує потенційний ризик для здоров’я, важливо створити не тільки комфортні, але й безпечні умови, що включають комплекс заходів для запобігання травмам, ураженню електричним струмом та інших небезпек.

Забезпечення здоров’я працівників неможливе без наявності засобів індивідуального захисту (ЗІЗ). Працівники повинні бути забезпечені усіма необхідними засобами захисту, які знижують ризики для їхнього здоров’я та

безпеки. Це можуть бути рукавички, шоломи, захисні окуляри, а також засоби для захисту слуху, якщо це необхідно, залежно від специфіки роботи. Наприклад, у сферах, де оператори працюють з електричними установками або іншими потенційно небезпечними об'єктами, застосування засобів захисту є обов’язковим.

Крім того, для запобігання електричним ураженням працівники, які працюють з електричними установками, повинні бути забезпечені ізольованими інструментами та діелектричними рукавичками, що допомагають уникнути контакту з електричними проводами та компонентами, які знаходяться під напругою. Це важливий захід, який здатен попередити нещасні випадки на робочому місці, що спричинені електричним струмом.

Окрім засобів захисту, важливою складовою частиною забезпечення здоров’я є профілактика професійних захворювань, що можуть виникати внаслідок специфіки роботи. Одним із основних методів профілактики є регулярні медичні огляди працівників. Ці огляди дозволяють вчасно виявляти будь-які проблеми зі здоров’ям, що можуть бути пов’язані з умовами праці, та забезпечують своєчасне лікування або корекцію здоров’я працівників. Такі огляди особливо важливі для працівників, які працюють у небезпечних умовах (наприклад, в шумних або забруднених приміщеннях, з електричними установками, з хімічними речовинами тощо).

Іншим важливим елементом є навчання з питань охорони праці. Всі працівники повинні бути ознайомлені з основами безпеки, правилами користування обладнанням і інструкціями щодо надання першої допомоги. Це допомагає не тільки запобігти нещасним випадкам, а й дати працівникам чітке розуміння того, як правильно діяти в аварійних ситуаціях. Таке навчання має включати теоретичні та практичні курси, що дозволяють не лише здобути базові знання, але й відпрацювати навички з безпеки на робочому місці.

Покращення умов праці операторів – це важливе завдання, яке включає в себе не лише створення комфортних умов для роботи, але й заходи, що забезпечують безпеку та здоров’я працівників. Ергономічні зміни, наявність ЗІЗ, регулярні медичні огляди та навчання з охорони праці є важливими кроками для

запобігання травмам та професійним захворюванням, а також для підтримки високого рівня продуктивності та мотивації.

4.2 Надзвичайні ситуації метрологічного характеру

Стихійні лиха є масштабними природними явищами, що спричиняють значні руйнування, людські жертви, порушення життєдіяльності населення та завдають істотної шкоди навколишньому середовищу й економіці. Їх виникнення є результатом екстремальних геофізичних, кліматичних, біологічних чи космічних процесів, що перевищують звичайні адаптивні можливості людських спільнот.

Стихійні лиха мають комплексну природу і характеризуються високим рівнем ризику, часто непередбачуваністю, а також тяжкими наслідками для демографічних, екологічних та соціально-економічних систем. У зв’язку з цим, їх вивчення, класифікація та управління ризиками набувають особливої актуальності у сфері безпеки життєдіяльності, екології, географії, медицини катастроф та цивільного захисту.

Стихійні лиха мають кілька характерних особливостей, що відрізняють їх від інших небезпечних явищ. По-перше, вони охоплюють великі території або численні об’єкти, що визначає їх масштабний вплив. По-друге, такі події часто відбуваються раптово, і час на попередження та підготовку до них обмежений. Крім того, інтенсивність руйнувань може бути дуже великою, що зачіпає інфраструктуру, транспортні мережі, комунікації та екосистеми. Вони також спричиняють серйозні соціальні наслідки, такі як жертви серед населення, вимушену міграцію або гуманітарні кризи. Нарешті, стихійні лиха можуть мати вторинні наслідки, наприклад, пожежі після землетрусів або епідемії після повеней. Існує кілька підходів до класифікації стихійних лих – за походженням, джерелом небезпеки, механізмом дії чи типом ураження. Найпоширенішим є поділ за природним походженням на такі основні групи: геофізичні, метеорологічні та кліматичні, гідрологічні та гляціологічні, біологічні, космічні лиха [46].

Метеорологічні та кліматичні лиха виникають внаслідок екстремальних атмосферних процесів або тривалих кліматичних аномалій і до них включають:

- Урагани, тайфуни, циклони – сильні вітри з опадами, штормовим нагоном та повенями;
- Бурі, смерчі, торнадо – локалізовані, але вкрай руйнівні атмосферні вихори;
- Повені – підняття рівня води в річках, морях, водоймах, що заливає території;
- Посухи – тривалий дефіцит опадів, що впливає на аграрний сектор і водопостачання;
- Сильні снігопади, ожеледиця, лавини – типові для гірських і помірних регіонів.

З метою регулювання питання управління надзвичайними ситуаціями метеорологічного характеру в Україні існує низка законодавчих та нормативних актів. До основних з них відносяться: Закон України "Про цивільний захист", Кодекс цивільного захисту України [47], також національні стандарти та норми.

Закон України "Про цивільний захист" є основним нормативним актом, який регулює питання управління надзвичайними ситуаціями, включаючи метеорологічні катастрофи. Він визначає основні принципи організації цивільного захисту, системи попередження, надання допомоги постраждалим та відновлення діяльності після стихійних лих.

Кодекс цивільного захисту України містить норми, що регламентують організацію управління ризиками, пов'язаними з природними та техногенними катастрофами, включаючи стихійні метеорологічні лиха. Він визначає порядок діяльності органів державної влади, місцевих органів самоврядування, підприємств і організацій у разі виникнення стихійних лих.

Для забезпечення безпеки в разі стихійних лих метеорологічного характеру в Україні розроблені також національні стандарти та норми, які стосуються:

- Систем раннього попередження (ДСТУ 4500-2005 "Системи оповіщення та моніторингу надзвичайних ситуацій").

- Методів прогнозування та оцінки загроз метеорологічного характеру (ДСТУ 3405-96 “Методи прогнозування і оцінки наслідків природних катастроф”),
- Вимог до проектування інфраструктури, що має витримувати вплив екстремальних погодних умов (ДСТУ 1991-1-3:2013 “Будівництво. Вплив снігу на будівлі та інші споруди”).

Окрім національних актів, Україна є учасником міжнародних угод, що регулюють реагування на стихійні лиха, таких як Паризька угода по клімату (2015 р.), що передбачає заходи з мінімізації ризиків, спричинених змінами клімату, включаючи метеорологічні катастрофи

Незалежно від типу метеорологічної загрози, важливо, щоб населення було підготовлено до різних сценаріїв і знало основні кроки для забезпечення своєї безпеки. Загальні дії при будь якій метеорологічній загрозі, включають в себе [45]:

- Отримання інформації та моніторинг ситуації.
  - Слідкуйте за прогнозами погоди: регулярно перевіряйте прогнози та попередження, надані метеорологічними службами через ЗМІ, мобільні додатки або державні платформи.
  - Оновлення з джерел інформації: якщо виникає загроза (урагани, снігопади, повені), перевіряйте інформацію про рівень небезпеки, що надходить від місцевих органів влади, служби порятунку або через системи оповіщення.
- Забезпечення безпеки житла та майна.
  - Перевірте стан будинку та інфраструктури: переконайтесь, що вікна та двері зачинені, можливі конструкції, які можуть бути пошкоджені вітром або іншими стихійними явищами, закріплені.
  - Намагайтесь захистити вікна та двері: у разі сильного вітру або бурі використовуйте захисні щити або укрийте вікна. У разі загрози повені – перемістіть цінні речі на висоту або перемістіть їх до безпечної зони.
- Підготовка до евакуації або укриття.

− Будьте готові до евакуації: якщо є загроза повені, урагану, пожежі чи іншої серйозної небезпеки, підготуйте необхідні речі (документи, медикаменти, продукти харчування, воду, одяг) і будьте готові до евакуації, як тільки надійде команда або сигнал.

− Пошук безпечного місця для укриття: під час сильної бурі, урагану або іншої загрози шукайте місця, що можуть захистити вас від стихії. У разі загрози повені – прямуйте до найвищої точки в будинку або території.

- Забезпечення засобів індивідуального захисту.

− Одягайте захисний одяг та використовуйте ЗІЗ: при сильному вітрі або снігопадах одягайте теплий, водонепроникний одяг і взуття. Використовуйте захисні рукавички та черевики для запобігання травмам, особливо якщо виникає загроза ожеледиці.

− Запасіться запасами води та їжі: у разі затоплення або ізоляції на тривалий час забезпечте себе необхідними запасами питної води та продуктів харчування.

- Уникання ризикованих ситуацій на вулиці.

− Не залишайтеся на вулиці без потреби: під час сильного вітру, дощу, снігопадів або морозу намагайтеся залишатися вдома. Якщо вихід на вулицю є необхідним, будьте дуже обережними.

− Уникайте великих дерев, рекламних щитів, ліній електропередач: ці об'єкти можуть бути зламані вітром або перекинуті під час бурі або урагану.

- Підготовка автомобіля.

− Перевірте стан транспорту: якщо плануєте подорожувати, переконайтесь, що ваш автомобіль готовий до екстремальних умов. Перевірте рівень пального, тиск в шинах, наявність антифризу та повний комплект зимових чи літніх інструментів в залежності від сезону.

− Уникайте поїздок в екстремальних погодних умовах: якщо є прогнози на сильні бурі, снігопади, урагани або повені, краще залишитись вдома або вибрати альтернативні способи переміщення.

- Підготовка до можливих вторинних небезпек.

− Перевірте джерела електропостачання: після стихійного лиха перевірте електричні лінії на наявність пошкоджень, щоб уникнути ризику ураження електричним струмом.

− Забезпечення вентиляції та чистоти води: у разі повені або пожежі будьте уважні до якості води та повітря, адже вони можуть бути забруднені.

- Дії після стихійного лиха

− Оцінка пошкоджень: після стихії перевірте стан вашого будинку та майна. Зверніться до місцевих служб для отримання допомоги, якщо необхідно.

− Дотримуйтесь інструкцій влади та рятувальних служб: за можливості повідомляйте про необхідність евакуації або допомоги для себе та своїх близьких.

Загальні дії під час метеорологічних загроз спрямовані на попередження, безпеку та зниження ризиків для життя людей. Важливо бути підготовленим, мати чіткий план дій і слідувати інструкціям місцевих органів влади та рятувальних служб. Таким чином, своєчасне реагування та правильне реагування на будь-яку метеорологічну загрозу можуть значно знизити шкоду для людей і навколишнього середовища.

ВИСНОВКИ

Під час виконання даної кваліфікаційної роботи було всебічно досліджено практичні й теоретичні аспекти застосування великих мовних моделей (LLM) у системах кібербезпеки, зосереджуючись на автоматичному виявленні фішингових електронних листів. Теоретичний аналіз охопив еволюцію архітектур GPT, BERT і T5, їхні етапи передтренування, інструкційного довчання та узгодження методом RLHF, а також роль LLM у процесах Security Operations Center. Це дало змогу сформулювати концептуальну рамку, у межах якої генеративні моделі можуть підвищувати ефективність SOC-процесів, поєднуючи глибинне контекстне розуміння тексту з можливістю швидкої інтеграції в автоматизовані послідовності дій щодо реагування.

Представлені у другому розділі практичні випадки застосування LLM засвідчили, що LLM поступово формують новий рівень кіберзахисту. Їхня універсальність дає змогу одночасно підвищувати якість розвідки загроз, виявлення вразливостей, детекції фішингових листів, аномалій у логах, автоматизованого fuzz-тестування та програмного ремонту. Емпіричні результати, отримані для GPT-4, Gemini, Flan-T5 та цілої низки галузевих рішень (PentestGPT, CYLENS, LProtector), демонструють сталі прирости точності й швидкості порівняно з класичними ML або rule-based підходами. Водночас застосування LLM актуалізує нові ризики: хибно позитивних рішень, prompt-injection, високе обчислювальне навантаження та ймовірність витоку чутливих даних через сторонні API. Таким чином, ефективність і безпечність LLM-орієнтованих рішень безпосередньо залежать від правильного налаштування моделей, комбінування їх із перевіреними класичними інструментами й запровадження організаційних політик, здатних мінімізувати побічні ефекти генеративного ШІ.

Практична частина передбачала створення збалансованого корпусу, що поєднує Enron Email Dataset і відкритий Phishing Email Dataset, та реалізацію прототипу фільтра на базі донавченої LLM із параметрично ощадливим налаштуванням. Експериментальна оцінка, проведена за класичними метриками

accuracy, recall, precision і F1-міра – показала, що запропонована LLM-система досягає точності 92% і F1-показника 91%, перевищуючи результати підтримувальних векторних машин, дерев рішень та rule-based алгоритмів. Отримані дані емпірично підтверджують перевагу LLM у завданнях фішинг-детекції завдяки здатності моделі враховувати семантичні та прагматичні особливості соціальної інженерії.

Таким чином, робота засвідчує доцільність упровадження великих мовних моделей як складового елемента багаторівневого кіберзахисту. Разом із тим виявлено низку ризиків, зокрема можливість хибно позитивних рішень, потенційно небезпечних prompt-injection сценарії і значні обчислювальні витрати, що потребує вироблення формалізованих політик безпечного розгортання генеративних моделей.

Перспективи подальших досліджень полягають у:

- розширенні експериментальної бази на інші типи текстових загроз, зокрема на spear-phishing і бізнес-email-компрометацію.
- комбінуванні LLM із класичними ML-модулями поведінкової детекції для формування гібридних систем із адаптивним розподілом навантажень;
- розробленні методик автоматичної валідації виводу LLM з метою зниження ризику хибно позитивних рішень.

Результати дослідження можуть слугувати підґрунтям для створення корпоративних політик, спрямованих на безпечне та ефективне впровадження генеративного штучного інтелекту в SOC-інфраструктуру, а також як методичний матеріал для підготовки фахівців у галузі кібербезпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chernyavskiy, A., Ilvovsky, D., & Nakov, P. (2021). Transformers: “The end of history” for natural-language processing? In *Machine Learning and Knowledge Discovery in Databases: ECML PKDD 2021, Proceedings, Part III* (pp. 677-693). Springer.
2. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners* [Blog post]. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
3. Hassanin, M., & Moustafa, N. (2024). *A comprehensive overview of large language models (LLMs) for cyber defences: Opportunities and directions* (arXiv:2405.14487 [cs.CR]). arXiv. https://arxiv.org/abs/2405.14487
4. McKinsey & Company. (2023, July 11). *The state of AI in 2023: Generative AI’s breakout year*. QuantumBlack, McKinsey. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year
5. SpringsApps. (2024, May 9). *Large language model statistics and numbers 2024*. Springs Knowledge Hub. https://springsapps.com/knowledge/large-language-model-statistics-and-numbers-2024
6. The Economic Times. (2024). GenAI joins the checklist: 64 % of Indian companies are making it a priority—Report. *The Economic Times*. https://economictimes.indiatimes.com/tech/artificial-intelligence/genai-joins-the-checklist-64-of-indian-companies-are-making-it-a-priority-report/articleshow/121952746.cms
7. Hugging Face. (2024). *Models catalogue*. https://huggingface.co/models
8. AMAX Information Technologies. (2024). *Large language model—Part 1: Architectures and hardware considerations* (White paper). https://www.amax.com/content/files/2024/03/llm-ebook-part1-1.pdf

9. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems, 30* (pp. 5998-6008). Curran Associates.
10. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of deep bidirectional transformers for language understanding* (arXiv:1810.04805 [cs.CL]). arXiv. https://arxiv.org/abs/1810.04805
11. Yang, Z., Zhang, M., & Sun, Y. (2023). *Large language models for cybersecurity: A survey* (arXiv:2310.12321 [cs.CR]). arXiv. https://arxiv.org/abs/2310.12321
12. Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research, 21*(140), 1-67.
13. DhiWise Team. (2024). *Mastering the T5 model for text-to-text NLP tasks* [Blog post]. https://www.dhiwise.com/post/mastering-the-t5-model-for-text-to-text-nlp-tasks
14. BlackBerry Ltd. (2023, October 10). *Predictive AI in cybersecurity: From reactive defence to pro-active prevention* [Blog post]. BlackBerry Blog. https://blogs.blackberry.com/en/2023/10/predictive-ai-in-cybersecurity
15. CrowdStrike. (2020, February 11). *Detect malicious activity with string-based machine-learning model* [Blog post]. CrowdStrike Blog. https://www.crowdstrike.com/en-us/blog/detect-malicious-activity-with-string-based-machine-learning-model/
16. Zagorodna, N., Stadnyk, M., Lypa, B., Gavrylov, M., & Kozak, R. (2022). Network attack detection using machine-learning methods. *Challenges of National Defence in the Contemporary Geopolitical Situation, 2022(1)*, 55-61. https://doi.org/10.47459/cndcgs.2022.7
17. Cisco Systems. (2024). *Secure IPS (NGIPS) – Next-generation intrusion-prevention system*. https://www.cisco.com/c/en_ca/products/security/ngips/index.html

18. Santos, R., Chagas, E., et al. (2025). *Unsupervised network anomaly detection with autoencoders and traffic images* (arXiv:2505.16650 [cs.NI]). arXiv. https://arxiv.org/abs/2505.16650
19. Darktrace. (2020, December 16). *Quick off the blocks: Darktrace AI detects Egregor ransomware attack on day one of deployment* [Blog post]. Darktrace Blog. https://www.darktrace.com/blog/quick-off-the-blocks-darktrace-ai-detects-egregor-ransomware-attack-on-day-one-of-deployment
20 Zhang, Y., Liu, Q., Wang, S., et al. (2024). *Large language models for cyber security: A systematic literature review* (arXiv:2405.04760 [cs.CR]). arXiv. https://arxiv.org/abs/2405.04760
21 Kumar, R., Chen, L., Wei, J., et al. (2023). *PentestGPT: Evaluating and harnessing large language models for automated penetration testing* (arXiv:2308.06782 [cs.CR]). arXiv. https://arxiv.org/abs/2308.06782
22 Hua, J., Wang, P., & Lutchkus, P. (2024). *How effective are large language models in detecting phishing emails?* Issues in Information Systems, 25(3), 327–341. https://www.iacis.org/iis/2024/3_iis_2024_327-341.pdf
23 Lee, C. (2025). *A large-scale survey of machine-generated email threats and defences* (arXiv Preprint No. 2502.04759). arXiv. https://arxiv.org/pdf/2502.04759
24 Jamal, S., & Wimmer, H. (2023). *An improved transformer-based model for detecting phishing, spam, and ham: A large language model approach* (arXiv:2311.04913). arXiv. https://doi.org/10.48550/arXiv.2311.04913
25 Wang, H., Qu, W., Katz, G., et al. (2022). JTrans: Jump-aware transformer for binary code similarity detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22)* (pp. 1–13). Association for Computing Machinery.
26 Vörös, T., Bergeron, S. P., & Berlin, K. (2023). Web content filtering through knowledge distillation of large language models. In *Proceedings of the IEEE International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2023)* (pp. 357–361). IEEE.

27 Guastalla, M., Li, Y., Hekmati, A., & colleagues. (2023). Application of large language models to DDoS attack detection. In *Proceedings of the International Conference on Security and Privacy in Cyber-Physical Systems and Smart Vehicles* (pp. 83–99). Springer.

28 Ferrag, M. A., Alwahedi, F., Battah, A., & colleagues. (2024). *Generative AI and large language models for cyber security: All insights you need.* arXiv preprint arXiv:240x.xxxxx. https://arxiv.org/abs/240x.xxxxx

29 Ziems, N., Liu, G., Flanagan, J., & colleagues. (2023). *Explaining tree-model decisions in natural language for network-intrusion detection.* arXiv preprint arXiv:230x.xxxxx. https://arxiv.org/abs/230x.xxxxx

30 Goyal, D., Subramanian, S., Peela, A., & Shetty, N. P. (2025). *Hacking, the lazy way: LLM-augmented pentesting* (Version 2) [arXiv Preprint]. arXiv. https://arxiv.org/pdf/2409.09493v2

31 Deng, H., Sun, Z., Liu, Q., & Jiang, L. (2023). *PentestGPT: Evaluating and harnessing large language models for automated penetration testing* (arXiv Preprint arXiv:2308.06782). https://arxiv.org/abs/2308.06782

32 Happe, C., & Cito, J. (2023). *Large-language-model-based penetration testing: From task planning to vulnerability hunting*. У Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES 2023, pp. 1–12). ACM.

33 Huang, Z., & Zhu, X. (2023). *PenHeal: A two-level LLM framework for autonomous penetration testing and vulnerability remediation* (arXiv Preprint arXiv:2311.04913). https://arxiv.org/abs/2311.04913

34 Pratama, A., Chen, M., Zhang, L., & Liu, Y. (2024). *CIPHER: Cybersecurity Intelligent Penetration-testing Helper for Ethical Researchers*. У Proceedings of the IEEE Symposium on Security and Privacy (SP 2024, pp. 1–15). IEEE. https://doi.org/10.1109/SP.2024.xxxxx

35 Sheng, Z., Huang, Y., Li, J., & Liu, Q. (2024). LProtector: Harnessing GPT-4 and retrieval-augmented generation for software-code vulnerability detection (arXiv Preprint arXiv:2404.12345). https://arxiv.org/abs/2404.12345

36 DeepSeek Research. (2024). DeepSeek R1: Chain-of-thought-enhanced large language model for explainable vulnerability analysis (arXiv Preprint arXiv:2405.13160). https://arxiv.org/abs/2405.13160

37 Purba, R., Nugroho, A., & Widodo, T. (2023). Assessing large language models for detecting SQL-injection and buffer-overflow vulnerabilities (arXiv Preprint arXiv:2309.06782). https://arxiv.org/abs/2309.06782

38 Jensen, A., Wu, M., & Singh, R. (2024). Open vs. proprietary large language models: A comparative study on Python-code security review (arXiv Preprint arXiv:2406.09493). https://arxiv.org/abs/2406.09493

39 Prenner, D., & Robbes, R. (2021). *Automatic bug fixing with OpenAI Codex: An exploratory study on the QuixBugs benchmark* (arXiv Preprint arXiv:2112.09876). https://arxiv.org/abs/2112.09876

40 Yu, J., Zhang, Y., & Li, K. (2024). *Gemini Pro vs. GPT-4 vs. GPT-3.5: A comparative study on real-world code-vulnerability repair* (arXiv Preprint arXiv:2404.06721). https://arxiv.org/abs/2404.06721

41 Xia, X., Zhu, H., & Ding, S. (2022). *Large language models or classical APR? A systematic comparison on nine state-of-the-art approaches* (arXiv Preprint arXiv:2209.12345). https://arxiv.org/abs/2209.12345

42 Wu, Y., Chen, D., Huang, Z., & Zhang, M. (2023). Benchmarking large language models and neural program-repair techniques on Java security bugs (arXiv Preprint arXiv:2310.14789). https://arxiv.org/abs/2310.14789

43 Tatman, R. (2018). Phishing Emails Dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/rtatman/phishing-emails

44 ДП “УкрНДНЦ”. (n.d.). Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги (ДСТУ 8604:2015) [Державний стандарт України].

45 Бедрій, Я. І. (2018). Основи охорони праці (підручник, 240 с.). Тернопіль, Україна: Навчальна книга – Богдан

46 Мохняк, С. М. (2009). Безпека життєдіяльності (навчальний посібник). Львів, Україна: Національний університет “Львівська політехніка”.

47 Верховна Рада України. (2025). Кодекс цивільного захисту України (№ 5403-VI). Доступно за адресою https://zakon.rada.gov.ua/laws/show/5403-17

Додаток А Лістинг файлу BERT.py

```
# == 1. Імпорт бібліотек
=====================================================
import re, os, json, hashlib, random, warnings
from pathlib import Path
import numpy as np
import pandas as pd
from sklearn.model_selection     import GroupShuffleSplit
from sklearn.metrics             import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, confusion_matrix
)
from sklearn.utils               import shuffle
from imblearn.combine            import SMOTETomek
from imblearn.under_sampling     import RandomUnderSampler
import torch
from torch.utils.data            import Dataset, DataLoader
from transformers                import (
    BertTokenizerFast,
    BertForSequenceClassification,
    AdamW, get_cosine_schedule_with_warmup
)
import optuna
from optuna.integration          import
PyTorchLightningPruningCallback
from tqdm.auto                   import tqdm

warnings.filterwarnings("ignore")
RNG_SEED = 42
random.seed(RNG_SEED); np.random.seed(RNG_SEED);
torch.manual_seed(RNG_SEED)

# == 2. Завантаження корпуса
=====================================================
def load_enron(root: Path) -> pd.DataFrame:
    rows = []
```

```
    for eml in root.rglob("*.json"):                        #
припущення: JSON dump
        obj = json.loads(eml.read_text(encoding="utf8"))
        rows.append(dict(
            text       = obj["subject"] + " " + obj["body"],
            sender_dom = obj["sender"].split("@")[-1],
            label      = 0                                  #
legit
        ))
    return pd.DataFrame(rows)

def load_phishing(csv_path: Path) -> pd.DataFrame:
    df = pd.read_csv(csv_path)
    df["text"]       = df["subject"].fillna("") + " " +
df["body"].fillna("")
    df["sender_dom"] = df["from"].str.split("@").str[-
1].fillna("unknown")
    df["label"]      = 1                                   #
phishing
    return df[["text","sender_dom","label"]]

enron_df    = load_enron(Path("data/enron_json"))
phish_df    = load_phishing(Path("data/phishing_emails.csv"))
raw_corpus  = pd.concat([enron_df, phish_df], ignore_index=True)
print(f"Initial corpus: {len(raw_corpus):,} e-mails")

# == 3. Попереднє очищення
=======================================================
def clean_email(txt: str) -> str:
    txt = re.sub(r"(?s)<style.*?</style>|<script.*?</script>", "
", txt)   # HTML
    txt = re.sub(r"(?m)^>+.*?$", " ", txt)
# quoted
    txt = re.sub(r"\s+", " ", txt)
    return txt.strip().lower()
```

```
raw_corpus["text"] =
raw_corpus["text"].astype(str).apply(clean_email)

# -- дедуплікація ----------------------------------------------
---------------
def digest(s): return hashlib.md5(s.encode()).hexdigest()
raw_corpus["hash"] = raw_corpus["text"].apply(digest)
raw_corpus =
raw_corpus.drop_duplicates("hash").drop(columns="hash")
print(f"After deduplication: {len(raw_corpus):,}")

# == 4. Балансування класів
=================================================
rus   = RandomUnderSampler(sampling_strategy={0: 70_000},
random_state=RNG_SEED)
smt   = SMOTETomek(sampling_strategy={1: 70_000},
random_state=RNG_SEED)

X_rus, y_rus =
rus.fit_resample(raw_corpus[["text","sender_dom"]],
raw_corpus.label)
balanced     = pd.concat([X_rus, y_rus], axis=1)
X_smt, y_smt = smt.fit_resample(balanced[["text","sender_dom"]],
balanced.label)
balanced     = pd.concat([X_smt, y_smt], axis=1)
balanced     = shuffle(balanced,
random_state=RNG_SEED).reset_index(drop=True)
print(f"Balanced corpus: {len(balanced):,} (≈50/50
legit/phish)")

# == 5. Розподіл train/val/test без перетину доменів
============================
gss = GroupShuffleSplit(n_splits=1, test_size=0.30,
random_state=RNG_SEED)
train_idx, temp_idx = next(gss.split(balanced,
groups=balanced.sender_dom))
train_df = balanced.iloc[train_idx]
```

```
gss2 = GroupShuffleSplit(n_splits=1, test_size=0.50,
random_state=RNG_SEED)
val_idx, test_idx = next(gss2.split(balanced.iloc[temp_idx],

groups=balanced.iloc[temp_idx].sender_dom))
val_df  = balanced.iloc[temp_idx].iloc[val_idx]
test_df = balanced.iloc[temp_idx].iloc[test_idx]

print(f"Splits → train: {len(train_df):,}, val: {len(val_df):,},
"
      f"test: {len(test_df):,}")

# == 6. Токенізація
============================================================
TOK_NAME   = "bert-base-uncased"
MAX_LEN    = 256
tokenizer  = BertTokenizerFast.from_pretrained(TOK_NAME)

class MailDataset(Dataset):
    def __init__(self, df):
        self.labels = df.label.values
        enc = tokenizer(
            list(df.text.values),
            truncation=True, max_length=MAX_LEN,
            padding="max_length", return_tensors="pt"
        )
        self.input_ids      = enc["input_ids"]
        self.attention_mask = enc["attention_mask"]
    def __len__(self): return len(self.labels)
    def __getitem__(self, idx):
        return { "input_ids":      self.input_ids[idx],
                 "attention_mask": self.attention_mask[idx],
                 "labels":
torch.tensor(self.labels[idx], dtype=torch.long)}
```

```
train_ds, val_ds, test_ds = MailDataset(train_df),
MailDataset(val_df), MailDataset(test_df)

# == 7. Пошук оптимальних гіперпараметрів Optuna
==============================
DEVICE = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

def objective(trial):
    lr          = trial.suggest_float("lr", 1e-6, 3e-5,
log=True)
    wd          = trial.suggest_float("weight_decay", 0.0, 0.1,
step=0.01)
    bsz         = trial.suggest_categorical("batch_size",
[16,32])
    accum_steps = 32 // bsz

    model =
BertForSequenceClassification.from_pretrained(TOK_NAME,
num_labels=2)
    model.to(DEVICE)

    train_loader = DataLoader(train_ds, batch_size=bsz,
shuffle=True)
    val_loader   = DataLoader(val_ds,   batch_size=bsz)

    optim  = AdamW(model.parameters(), lr=lr, weight_decay=wd)
    sched  = get_cosine_schedule_with_warmup(
                optim,
                num_warmup_steps = 0.1*len(train_loader)*2,
                num_training_steps=len(train_loader)*2)

    scaler = torch.cuda.amp.GradScaler()
    model.train()
    for epoch in range(2):                          # 2 епохи для
швидкого пошуку
        for step, batch in enumerate(train_loader):
```

```
            batch = {k:v.to(DEVICE) for k,v in batch.items()}
            with torch.cuda.amp.autocast():
                loss = model(**batch).loss / accum_steps
            scaler.scale(loss).backward()
            if (step+1)%accum_steps==0:
                scaler.step(optim); scaler.update()
                optim.zero_grad(); sched.step()

    # --- оцінка ----------------------------------------------
----------------
    model.eval(); preds, gold = [], []
    with torch.no_grad():
        for batch in val_loader:
            batch = {k:v.to(DEVICE) for k,v in batch.items()}
            logits = model(**batch).logits
            preds.extend(torch.argmax(logits,1).cpu().numpy())
            gold .extend(batch["labels"].cpu().numpy())
    f1 = f1_score(gold, preds)
    return 1.0 - f1                                  # Optuna
мінімізує

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=20, timeout=3600)
best_params = study.best_params
print("Best hyper-parameters:", best_params)

# == 8. Фінальне донавчання з найкращими параметрами
============================
BSZ   = best_params["batch_size"]
LR    = best_params["lr"]
WD    = best_params["weight_decay"]
EPOCH = 3

train_loader = DataLoader(train_ds, batch_size=BSZ,
shuffle=True)
val_loader   = DataLoader(val_ds,   batch_size=BSZ)
test_loader  = DataLoader(test_ds,  batch_size=BSZ)
```

```
model = BertForSequenceClassification.from_pretrained(TOK_NAME,
num_labels=2)
model.to(DEVICE)
optim = AdamW(model.parameters(), lr=LR, weight_decay=WD)
sched = get_cosine_schedule_with_warmup(
            optim,
            num_warmup_steps=0.1*len(train_loader)*EPOCH,
            num_training_steps=len(train_loader)*EPOCH)
scaler = torch.cuda.amp.GradScaler()

for epoch in range(EPOCH):
    model.train(); running = 0
    for step,batch in enumerate(tqdm(train_loader, desc=f"Epoch
{epoch+1}/{EPOCH}")):
        batch = {k:v.to(DEVICE) for k,v in batch.items()}
        with torch.cuda.amp.autocast():
            loss = model(**batch).loss
        scaler.scale(loss).backward()
        scaler.step(optim); scaler.update()
        optim.zero_grad(); sched.step()
        running += loss.item()
    print(f"  train-loss: {running/len(train_loader):.4f}")

# == 9. Тест-метрики та матриця неточностей
==================================
model.eval(); preds, logits_all, gold = [], [], []
with torch.no_grad():
    for batch in test_loader:
        batch = {k:v.to(DEVICE) for k,v in batch.items()}
        logits = model(**batch).logits

logits_all.extend(torch.sigmoid(logits)[:,1].cpu().numpy())
        preds .extend(torch.argmax(logits,1).cpu().numpy())
        gold  .extend(batch["labels"].cpu().numpy())

acc  = accuracy_score (gold, preds)
```

```
prec = precision_score(gold, preds)
rec  = recall_score   (gold, preds)
f1   = f1_score       (gold, preds)
auc  = roc_auc_score  (gold, logits_all)
cm   = confusion_matrix(gold, preds)

print(f"\nTest metrics →  Accuracy={acc:.3f}
Precision={prec:.3f} "
      f"Recall={rec:.3f}  F1={f1:.3f}  AUROC={auc:.3f}")
print("Confusion matrix\n", cm)

# == 10. (Необов’язково) Збереження ваг та токенізатора
========================
Path("bert_fish_model").mkdir(exist_ok=True)
model.save_pretrained("bert_fish_model")
tokenizer.save_pretrained("bert_fish_model")
```