

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО  
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»  
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ  
КАФЕДРА КІБЕРБЕЗПЕКИ

ДОПУСТИТИ ДО ЗАХИСТУ  
Завідувач кафедри кібербезпеки

\_\_\_\_\_ Анна ІЛЬСНКО  
“ \_\_\_\_\_ ” \_\_\_\_\_ 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ “МАГІСТР”

**Тема:** Система моніторингу подій кібербезпеки з використанням штучного інтелекту

**Виконавець:**

Інна КУЗЬМІЧОВА

**Керівник:**

к.т.н.

Наталія ГУЛАК

**Нормоконтролер:**

к.т.н., доцент

Андрій ПЕТРЕНКО

Київ 2024

**ДЕРЖАВНЕ НЕКОМЕРЦІЙНЕ ПІДПРИЄМСТВО  
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ  
«КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»**

Факультет комп'ютерних наук та технологій  
Кафедра кібербезпеки  
Освітній ступінь магістр  
Спеціальність 125 «Кібербезпека та захист інформації»  
Освітньо-професійна програма «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

\_\_\_\_\_ Анна ІЛЬЄНКО

«30» \_\_ 08 \_\_\_\_\_ 2024 р.

**ЗАВДАННЯ**

**на виконання кваліфікаційної роботи**

**Кузьмічової Інни Юріївни**

1. Тема кваліфікаційної роботи: Система моніторингу подій кібербезпеки з використанням штучного інтелекту.

Затверджена наказом ректора від 30.08.2024 р. №1695/ст.

2. Термін виконання роботи: з 30.08.2024 по 15.12.2024

3. Вихідні дані до роботи: хмарні технології, аккаунти для Zabbix, OpenAI, Telegram канал зв'язку, нейрона неленійна мережа, сервер cold-boy.zabbix.cloud:10051, сервер на ngrok, мова програмування Python.

4. Зміст пояснювальної записки: аналіз використання штучного інтелекту в кібербезпеці, порівняльний аналіз моделей для створення мережі штучного інтелекту, створення і наладка системи штучного інтелекту для оповіщення подій кібербезпеки.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу: презентація

## 6. Календарний план-графік

/п	Завдання	Термін виконання	Підпис керівника
1	Провести аналіз використання штучного інтелекту для визначення загроз інформації у системах і мережах	30.08.2024 – 15.09.2024	Виконано
2	Обґрунтувати доцільність використання штучного інтелекту у системі моніторингу	16.09.2024 – 25.09.2024	Виконано
3	Розробити інтерфейс системи поєднання Zabbix з OpenAI у Telegram каналі	26.09.2024 – 25.10.2024	Виконано
4	Створити програмний код для з'єднання Zabbix з OpenAI	01.11.2024 – 10.11.2024	Виконано
5	Протестувати розроблену систему	10.11.2024 – 18.11.2024	Виконано

7. Дата видачі завдання: «30» \_\_08\_\_ 2024 р.

Керівник кваліфікаційної роботи: \_\_\_\_\_ Наталія ГУЛЯК  
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання: \_\_\_\_\_ Інна КУЗЬМІЧОВА  
(підпис здобувача вищої освіти) (П.І.Б.)

## РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Система моніторингу подій кібербезпеки з використанням штучного інтелекту»: 95 с., 21 рис., 1 табл., 6 формул, 28 літературних джерела.

Об'єкт дослідження: процес машинного навчання у мережі Штучного Інтелекту.

Предмет дослідження: моделі Штучного Інтелекту для використання в кібербезпеці.

Мета кваліфікаційної роботи: створення програмного коду для поєднання роботи системи Zabbix з OpenAI у зручному для користувача форматі інтерфейса через Telegram в системі моніторингу для виявлення подій кібербезпеки.

Методи дослідження: об'єктно-орієнтоване програмування мовою Python, модель нелінійної нейронної мережі.

Практична цінність: Створена система моніторингу на штучному інтелекті дозволяє швидко і надійно реагувати на події кібербезпеки та в прискореному режимі навчати персонал роботі у системі моніторингу.

Наукова новизна: розроблено програмний код для поєднання роботи системи Zabbix з OpenAI на основі нелінійної нейронної мережі, а також запропоновано представлення відповіді нейромережі у зручному для користувача інтерфейсі з використанням Telegram каналу, що надало можливість скоротити термін реагування на оповіщення моніторингової систем Zabbix і виключає кількість помилкових дій при ліквідації наслідків втручання в систему.

Результати кваліфікаційної роботи рекомендується використовувати

**КІБЕРБЕЗПЕКА, ШТУЧНИЙ ІНТЕЛЕКТ, МОНІТОРІНГ ПОДІЙ  
КІБЕРБЕЗПЕКИ**

## ЗМІСТ

Вступ	7
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В КІБЕРБЕЗПЕЦІ	13
1.1 Вступ до кібербезпеки та потреба у вдосконалених системах моніторингу	13
1.2 Визначення ролі штучного інтелекту в управлінні ризиками за стандартом ISO/IEC 27005	13
1.3 Приклади застосування штучного інтелекту для моніторингу подій кібербезпеки	14
1.4 Переваги використання ШІ в кібербезпеці згідно зі стандартом ISO/IEC 27005	15
1.5 Висновк до розділу №1	15
РОЗДІЛ 2. АНАЛІЗ МОДЕЛЕЙ ДЛЯ СТВОРЕННЯ МЕРЕЖІ ШТУЧНОГО ІНТЕЛЕКТУ	16
2.1 Вступ до аналізу створення нейромереж	16
2.2 Ресурси, які потрібні для створення різних моделей нейромереж	20
2.3 Вибір моделі та базові алгоритми створення нейромереж	23
2.4 Створення моделі лінійної нейромережі	24
2.5 Створення моделі нелінійної нейромережі	29
2.6 Створення моделі нечіткої нейромережі	37
2.7 Створення моделі деревоподібної нейромережі	41
2.8 Створення моделі байєсових нейромереж	46
2.9 Алгоритм створення моделі нейромережі для нашого експерименту	51

2.10 Висновок до розділу №2	57
РОЗДІЛ 3. СТВОРЕННЯ І НАЛАДКА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПОВІЩЕННЯ ПОДІЙ КІБЕРБЕЗПЕКИ	60
3.1 Опис систем Zabbix та OpenAI	60
3.2 Етапи експерименту	62
3.3 Висновок до розділу №3	84
ВИСНОВКИ	86
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ТА ВИКОРИСТАНИХ ДЖЕРЕЛ	88
ДОДАТКИ	91
Додаток А	91

## ВСТУП

**Актуальність теми.** В умовах зростання обсягів цифрової інформації та збільшення кількості кібератак питання забезпечення кібербезпеки набуває стратегічної важливості. Розвиток штучного інтелекту відкриває нові можливості для створення автоматизованих систем моніторингу, здатних швидко ідентифікувати, реагувати та аналізувати кіберінциденти. На мою думку, існуючі рішення вимагають вдосконалення, зокрема шляхом інтеграції зручних для користувача інтерфейсів і поєднання різних інструментів для підвищення швидкості та ефективності реагування. Запропонована робота спрямована на поєднання системи моніторингу Zabbix із Штучним Інтелектом (ШІ) OpenAI для автоматизації процесів кіберзахисту, що дозволить знизити навантаження на спеціалістів та підвищити надійність захисту інформації.

Також слід зазначити, що в умовах зростання обсягів цифрової інформації та збільшення кількості кібератак питання забезпечення кібербезпеки набуває стратегічної важливості. Однак в Україні, особливо серед малого та середнього бізнесу, існує значна проблема втрати кваліфікованих працівників у сфері кібербезпеки, а також недостатня увага до моніторингу подій кібербезпеки. Відсутність ефективних заходів у моніторингу робить ці підприємства вразливими до кіберзагроз, що може призводити до серйозних економічних та репутаційних втра.

Міжнародний стандарт ISO/IEC 27005, який регулює управління ризиками інформаційної безпеки, підкреслює важливість моніторингу подій як ключового аспекту управління ризиками. Виконання вимог цього стандарту дозволяє не лише вчасно виявляти кіберінциденти, але й краще розуміти загрозу та її вплив на бізнес [3]. Використання сучасних інструментів та методів моніторингу, таких як системи на основі штучного інтелекту, відповідає вимогам цього стандарту і дозволяє підвищити якість захисту інформації. Сертифікація українських компаній стандартам у сфері кібербезпеки наразі

набирає обертів, але це пов'язано, нажаль, не з бажанням дійсно побудувати систему безпеки продуктів і процесів, а у зв'язку з виходом компаній на міжнародні ринки, де є жорсткі вимоги і правила до стандартів. У своїй власній практиці при впровадженні стандарту ISO/IEC 27001:2022 в ІТ компанії (450+ працівників), я отримала можливість на власному досвіді переконатися наскільки важко працівникам зрозуміти події кібербезпеки і як важко навчити персонал моніторингу подій кібербезпеки. Тому вважаю, що представлена в даній практичній частині схема взаємодії: від події кіберінциденту до розуміння відповідальним співробітником схеми реагування на подію допоможе запровадити новий, зрозумілий, простий і досить дешевий механізм моніторингу.

Також я вірю, що штучний інтелект, як передбачено в даній роботі, може відігравати значну роль у швидкому навчанні персоналу реагуванню на кіберінциденти. Зокрема, ШІ здатний автоматично генерувати структуровані звіти про події кібербезпеки, що включають інструкції для їх усунення. Це дозволяє працівникам швидше оволодіти знаннями та навичками, необхідними для реагування на загрози. Крім того, система на базі ШІ може ідентифікувати та сповіщати саме той персонал, який потрібно залучити для вирішення конкретної проблеми, що значно підвищує ефективність системи управління ризиками. Така функціональність є надзвичайно важливою для забезпечення комплексного захисту інформаційних активів підприємства.

**Мета і завдання роботи.** Створення програмного коду для поєднання роботи системи Zabbix з OpenAI у зручному для користувача форматі інтерфейса через Telegram в системі моніторингу для виявлення подій кібербезпеки.

Для досягнення цієї мети необхідно виконати такі завдання:

- аналіз використання штучного інтелекту в кібербезпеці;
- аналіз моделей для створення мережі штучного інтелекту,
- створення, наладка і тестування системи штучного інтелекту для оповіщення подій кібербезпеки.



**Об'єкт і предмет дослідження.** Об'єктом дослідження є процес машинного навчання в мережах штучного інтелекту, зокрема використання великих мовних моделей від OpenAI. Використання таких моделей для обробки інформації про кіберзагрози підтверджено в багатьох дослідженнях, що демонструють їхню ефективність у виявленні та реагуванні на події кібербезпеки.

Предметом дослідження є моделі штучного інтелекту, які застосовуються для моніторингу подій кібербезпеки. Основну увагу в цій роботі буде зосереджено на нелінійній нейронній мережі, що забезпечує ефективну інтеграцію з системами кіберзахисту. Вибір саме нелінійної нейромережі, розробленої OpenAI, обумовлений її здатністю обробляти складні структури даних і розпізнавати закономірності у великих масивах інформації, що є критичним для аналізу кіберзагроз.

Нелінійні моделі штучного інтелекту мають значні переваги в порівнянні з лінійними, оскільки можуть обробляти багатовимірні дані та адаптуватися до мінливих умов кіберсередовища. Це особливо важливо для виявлення нових і нетипових загроз, які можуть залишитися непоміченими при використанні традиційних лінійних підходів [13]. Крім того, моделі OpenAI дозволяють не лише реагувати на кіберінциденти, а й формувати звіти та рекомендації у структурованому форматі, що сприяє швидкому розумінню і усуненню загроз.

Отже, предметом дослідження є саме ті моделі штучного інтелекту, які здатні адаптуватися до змін і ефективно комунікувати з користувачами в реальному часі, що робить їх ідеальними для застосування в системах моніторингу кібербезпеки.

**Методи дослідження.** Для досягнення поставленої мети використовувались методи об'єктно-орієнтованого програмування на мові Python, а також інструменти для інтеграції різних платформ кібербезпеки. Зокрема, було реалізовано зв'язок між платформами Zabbix і OpenAI за допомогою серверів на платформі Nginx. Цей підхід забезпечує отримання та обробку повідомлень про кіберінциденти: дані з Zabbix передаються на OpenAI

для розшифрування помилок та формування рекомендацій, після чого результат повертається у Zabbix для подальшої обробки.

**Наукова новизна отриманих результатів.** Розроблено програмний код для поєднання роботи системи Zabbix з OpenAI на основі нелінійної нейронної мережі, а також запропоновано представлення відповіді нейромережі у зручному для користувача інтерфейсі з використанням Telegram каналу, що надало можливість скоротити термін реагування на оповіщення моніторингової систем Zabbix і виключає кількість помилкових дій при ліквідації наслідків втручання в систему.

Запропоноване рішення використовує інтеграцію з Telegram, що забезпечує зручність і мобільність у використанні. Це дозволяє спеціалістам оперативно отримувати повідомлення про кіберзагрози у реальному часі, спрощуючи процес взаємодії з системою моніторингу Zabbix і роблячи її більш доступною для різних категорій користувачів. Автоматизовані сповіщення, налаштовані на передачу відповідей OpenAI у спеціальний Telegram-канал, створюють ефективний механізм реагування, що підвищує рівень кібербезпеки та надає можливості для швидкого навчання персоналу у сфері реагування на інциденти.

**Практичне значення отриманих результатів.** Створена система моніторингу на штучному інтелекті дозволяє швидко і надійно реагувати на події кібербезпеки та в прискореному режимі навчати персонал роботі у системі моніторингу.

Особливу практичну цінність система здобуває завдяки інтеграції з мобільним інтерфейсом у Telegram, що відповідає сучасним тенденціям у використанні мобільних пристроїв для реагування на зовнішні фактори. Наразі понад 70% користувачів використовують мобільні телефони для отримання й обробки інформації, тоді як частка веб-інтерфейсів становить близько 30%. Телефон вже став багатофункціональним пристроєм, здатним забезпечити швидкий доступ до даних, а також можливість приймати оперативні рішення,

що робить його навіть більш функціональним, ніж ноутбук у певних контекстах [17].

Система рекомендована для використання у компаніях, які потребують надійного кіберзахисту, оскільки вона забезпечує додаткову зручність та інформативність через можливість мобільного моніторингу, дозволяючи користувачам залишатися на зв'язку з подіями кібербезпеки в реальному часі, незалежно від їхнього місцезнаходження.

**Особистий внесок здобувача вищої освіти.** У ході виконання кваліфікаційної роботи здобувач вищої освіти самостійно розробив програмний код для інтеграції системи моніторингу Zabbix із моделями штучного інтелекту OpenAI, що забезпечує автоматичний аналіз кіберінцидентів і надання рекомендацій щодо їх усунення. Крім того, здобувач розробив структуру обміну даними між Zabbix та OpenAI через платформу Ngrok, що дозволяє надійно й оперативно обробляти інформацію про кіберзагрози та передавати її до Telegram для швидкого реагування користувачів.

Здобувач також забезпечив повну інтеграцію системи з мобільним інтерфейсом у Telegram, включаючи налаштування автоматизованих сповіщень, розробку і тестування каналу для передачі даних користувачам у режимі реального часу. Враховуючи, що понад 70% користувачів використовують мобільні телефони для оперативного реагування на зовнішні фактори, здобувач налаштував систему з орієнтацією на мобільну взаємодію, що забезпечує зручність і мобільність у використанні. Усі етапи, від початкової ідеї до налаштування кінцевого інтерфейсу, були виконані здобувачем самостійно, що підтверджує значний внесок в розробку системи моніторингу подій кібербезпеки.

У випадках, коли було використано ідеї або дослідження інших авторів (наприклад, принципи інтеграції систем моніторингу з моделями штучного інтелекту або функціональні можливості Telegram для передачі даних), здобувач вищої освіти адаптував та модифікував ці підходи до конкретних умов роботи системи, які розроблялися у рамках даної кваліфікаційної роботи.

**Апробація отриманих результатів.** VII International scientific and practical conference «World educational trends: lifelong learning in the information society»

**Публікації.** Кузмічова І.Ю. Використання штучного інтелекту у системі моніторингу подій кібербезпеки /І.Ю. Кузмічова, Н.К. Гулак // VII International scientific and practical conference «World educational trends: lifelong learning in the information society», 15-18 жовтня 2024р. - Афіни, Греція. – С. 50 - 55.

# РОЗДІЛ 1

## АНАЛІЗ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В КІБЕРБЕЗПЕЦІ

### **1.1 Вступ до кібербезпеки та потреба у вдосконалених системах моніторингу**

Кібербезпека є однією з найважливіших сфер сучасного управління інформаційними технологіями, особливо з огляду на постійно зростаючі ризики кібератак, спрямованих на критичні ресурси та дані [1, 2]. У зв'язку зі зростанням обсягів даних, що підлягають захисту, та складністю загроз, кібербезпека вже не може обмежуватися традиційними заходами, такими як фільтри та базові антивірусні програми.

Міжнародний стандарт ISO/IEC 27005 наголошує на необхідності комплексного підходу до управління ризиками, включаючи своєчасне виявлення та реагування на загрози [3]. Стандарт визначає основні етапи процесу управління ризиками інформаційної безпеки, включаючи моніторинг подій, оцінку загроз та реагування на інциденти, що є ключовими елементами сучасних систем кіберзахисту. У цьому контексті штучний інтелект (ШІ) виступає як ефективний інструмент для автоматизації та оптимізації багатьох аспектів кібербезпеки [5, 12].

### **1.2 Визначення ролі штучного інтелекту в управлінні ризиками за стандартом ISO/IEC 27005**

Стандарт ISO/IEC 27005, який орієнтований на управління ризиками інформаційної безпеки, підкреслює необхідність постійного моніторингу та аналізу подій для зменшення можливих ризиків [3]. Штучний інтелект, зокрема у формі мовних моделей та систем машинного навчання, дозволяє ефективно

автоматизувати ці процеси завдяки своїй здатності до обробки великих обсягів інформації, швидкого виявлення аномалій та адаптації до нових загроз [11, 13].

На відміну від традиційних систем, які часто мають обмежені можливості для виявлення складних або нових загроз, ШІ здатен до самонавчання та аналізу даних у реальному часі. Це робить його особливо ефективним у системах моніторингу кіберзагроз, де важливі як точність, так і швидкість реагування [12]. Зокрема, нелінійні нейронні мережі, такі як моделі OpenAI, дозволяють здійснювати комплексний аналіз загроз, виявляти приховані патерни та реагувати на потенційні інциденти з точністю, недоступною для традиційних засобів моніторингу [18].

### **1.3 Приклади застосування штучного інтелекту для моніторингу подій кібербезпеки**

У сучасних системах кібербезпеки застосування ШІ стає дедалі поширенішим. Наприклад, інтеграція мовних моделей, таких як OpenAI, з інструментами моніторингу (напр., Zabbix) дозволяє автоматизувати процеси обробки даних про кіберзагрози та формувати структуровані звіти, що підвищує ефективність управління ризиками [12, 16]. Такі інтегровані системи можуть виконувати завдання, які раніше потребували великої кількості часу та ручного втручання.

Крім того, використання інтерфейсів для миттєвих сповіщень, таких як Telegram API, дозволяє оперативно передавати інформацію про інциденти та рекомендації щодо реагування відповідальним особам, навіть коли вони знаходяться поза офісом [9, 21]. За даними досліджень, більш ніж 70% користувачів використовують мобільні пристрої для взаємодії з системами кіберзахисту, що робить Telegram ефективним інструментом для оперативного моніторингу [17].

## **1.4 Переваги використання ШІ в кібербезпеці згідно зі стандартом ISO/IEC 27005**

Відповідно до вимог ISO/IEC 27005, системи управління ризиками повинні бути здатні реагувати на загрози в реальному часі, що є однією з ключових переваг використання штучного інтелекту [3]. ШІ дозволяє не тільки ідентифікувати загрози, а й робити прогнози щодо можливих сценаріїв атаки, завдяки чому ризики можуть бути значно знижені [12].

Поєднання можливостей аналізу даних у реальному часі та адаптивних алгоритмів ШІ забезпечує гнучкість систем кібербезпеки, що дозволяє їм швидко реагувати на зміни в кіберпросторі. Це робить штучний інтелект не просто додатковим інструментом, а ключовою складовою сучасного управління ризиками інформаційної безпеки [15].

## **1.5 Висновк до розділу №1**

Використання штучного інтелекту в кібербезпеці є не лише актуальним, а й необхідним для сучасних організацій, що прагнуть забезпечити високий рівень захисту своїх інформаційних ресурсів. Згідно зі стандартом ISO/IEC 27005, системи управління ризиками мають постійно вдосконалюватися для відповідності мінливим вимогам безпеки, що робить ШІ потужним інструментом у сфері моніторингу та реагування на кіберзагрози [3, 5]. Інтеграція ШІ з системами моніторингу, такими як Zabbix, та використання мобільних інтерфейсів на кшталт Telegram значно підвищують ефективність управління ризиками, знижуючи витрати на персонал і час на реагування [17, 19].

## РОЗДІЛ 2

# АНАЛІЗ МОДЕЛЕЙ ДЛЯ СТВОРЕННЯ МЕРЕЖІ ШТУЧНОГО ІНТЕЛЕКТУ

### 2.1 Вступ до аналізу створення нейромереж

Історичний екскурс. Розвиток нейронних мереж має довгу і багату історію, яка охоплює кілька десятиліть. Від ранніх ідей, натхненних біологічними процесами, до сучасних складних архітектур, нейромережі стали основою сучасного штучного інтелекту (ШІ), відіграючи ключову роль у багатьох сферах, таких як обробка природної мови (NLP), комп'ютерне бачення та автоматизація.

1940-1950-ті: Біологічне натхнення. Розвиток нейромереж розпочався з вивчення роботи людського мозку. У 1943 році нейрофізіологи Воррен Маккалок і Волтер Піттс запропонували першу математичну модель нейрона [1]. Їхня модель показала, що нейрони можуть виконувати логічні операції, такі як "І" та "АБО", і заклала основи для майбутнього розвитку нейромереж.

1950-1960-ті: Поява перцептрона. У 1958 році Френк Розенблатт створив перцептрон — першу комп'ютерну модель нейронної мережі, яка могла навчатися. Перцептрон складався з вхідного шару, ваг і активаційної функції, що визначає, чи повинен нейрон "активуватися" [4]. Незважаючи на те, що модель мала обмеження, вона стала важливим кроком у розумінні того, як штучні нейрони можуть працювати разом.

Обмеження: Одношарові перцептрони не могли обробляти нелінійні задачі, такі як XOR. Цей недолік уповільнив прогрес у галузі на кілька років [9].

1980-ті: Відродження через зворотне поширення. У 1986 році Румельхарт, Гінтон і Вільямс запропонували метод зворотного поширення помилки (backpropagation), що вирішив проблему навчання багатошарових мереж [5]. Цей алгоритм дозволив нейромережам коригувати ваги на основі



помилки і зробив можливим використання багатшарових архітектур для вирішення складніших задач.

1990-ті: Розвиток спеціалізованих архітектур. У 1990-х роках з'явилися спеціалізовані типи нейромереж:

1. Рекурентні нейромережі (RNN): дозволили працювати з послідовними даними, такими як текст і тимчасові ряди [8].
2. Складчасті нейромережі (CNN): зробили революцію в комп'ютерному баченні завдяки здатності автоматично виділяти ознаки зображень [11].

Ці архітектури розширили сферу застосування нейромереж і заклали основу для багатьох сучасних досягнень.

2000-ті: Епоха великих даних і апаратного забезпечення. Швидкий розвиток апаратного забезпечення, особливо графічних процесорів (GPU), і доступ до великих обсягів даних значно прискорили прогрес у навчанні нейромереж. Ці фактори дозволили використовувати глибокі нейронні мережі (DNN) для вирішення завдань, які раніше вважалися надто складними [14]

2010-ті: Епоха глибокого навчання. У 2012 році команда Хінтона здобула перемогу в конкурсі ImageNet, продемонструвавши силу глибоких нейронних мереж із багатьма шарами (глибинне навчання). Це викликало великий інтерес до глибокого навчання і сприяло широкому застосуванню нейромереж у таких сферах, як:

- Обробка природної мови (наприклад, трансформерні моделі GPT, BERT) [15, 18].
- Автономні автомобілі.
- Рекомендаційні системи.

Сучасність: Трансформери і генеративні моделі. На сьогодні трансформерні архітектури, такі як GPT і BERT, домінують у багатьох задачах ШІ. Вони демонструють унікальні можливості:

- Генерація тексту.
- Машинний переклад.
- Автоматизація складних процесів [20].

Такі моделі, як OpenAI, стали важливими інструментами в кібербезпеці, моніторингу систем та інших галузях завдяки їхній здатності адаптуватися до нових даних. Тож далі ми розглянемо основні моделі створення нейромереж [19].

Нейронні мережі (або нейромережі) є обчислювальними моделями, що імітують роботу людського мозку, об'єднуючи численні вузли (нейрони), які взаємодіють для вирішення різних завдань, таких як розпізнавання образів, обробка мови, класифікація даних тощо [22]. Кожна нейромережа має свою специфічну архітектуру та функціональні можливості, залежно від її структури та типу обчислень, що робить її придатною для вирішення конкретних завдань у кібербезпеці, включаючи моніторинг та реагування на загрози [23].

Основні типи нейромереж, які використовуються для різних завдань кібербезпеки, включають лінійні, нелінійні, нечіткі логічні, деревоподібні та байєсові моделі. Кожна з цих моделей має свої особливості та переваги.

Лінійні нейромережі. Лінійні нейромережі використовують лінійні активаційні функції, що обмежує їхню здатність до адаптації в умовах складних патернів даних. Вони добре підходять для задач, де зв'язок між вхідними та вихідними даними є приблизно лінійним, але обмежені у здатності обробляти більш складні нелінійні відношення [23]. Зважаючи на їхню простоту, лінійні моделі можуть бути корисними для завдань з невеликою кількістю змінних і чіткою структурою, але є менш ефективними для завдань кібербезпеки, які вимагають більшої гнучкості та здатності працювати з великими обсягами даних [24].

Нелінійні нейромережі. Нелінійні нейромережі використовують складніші активаційні функції (такі як ReLU, Sigmoid, Tanh), що дозволяє моделювати зв'язки між даними більш складного характеру [22, 24]. Завдяки своїй нелінійності, ці моделі можуть навчатися на великих наборах даних і адаптуватися до складних патернів, таких як зображення, мова та аномальні дії у кіберпросторі. Це робить їх потужними інструментами для виявлення та аналізу загроз, а також для побудови автоматизованих систем моніторингу [23].

Нечіткі нейронні мережі. Нечіткі нейромережі базуються на концепціях нечіткої логіки, що дозволяє працювати з неточними або нечіткими даними. Це особливо корисно в умовах невизначеності, коли інформація є частково або нечітко визначеною [25]. Такі мережі добре підходять для задач, де важлива гнучкість і здатність до інтерпретації даних, наприклад, у системах рекомендацій та експертних системах, які підтримують процес прийняття рішень у кібербезпеці [22].

Деревоподібні моделі (Decision Trees). Деревоподібні моделі, або дерева рішень, представляють процес прийняття рішень у вигляді деревовидної структури, де кожен вузол відповідає за певне рішення, а гілки — за можливі наслідки цього рішення [27]. Такі моделі є інтуїтивно зрозумілими, легко візуалізуються та можуть бути використані для швидкого прийняття рішень у кібербезпеці. Однак через свою простоту вони можуть бути менш точними порівняно з іншими моделями, особливо для складних завдань [24].

Байєсові моделі (Bayesian Networks). Байєсові нейронні мережі використовують ймовірнісний підхід, базуючись на Байєсовій теоремі для оцінки та оновлення ймовірностей на основі наявної інформації [25]. Вони будують графічні моделі залежностей між змінними, що дозволяє враховувати невизначеність і працювати зі складними взаємозв'язками, наприклад, у прогнозуванні загроз або класифікації інцидентів у кібербезпеці. Байєсові моделі є особливо корисними для задач, де важливо враховувати ймовірності і невизначеність у даних [26].

Висновок. Кожен з цих типів нейромереж має свої сильні та слабкі сторони, що робить їх придатними для різних завдань. Вибір конкретної моделі для кібербезпеки залежить від вимог до швидкості, точності, інтерпретованості та складності даних. Наприклад, деревоподібні моделі добре підходять для простих та інтуїтивних рішень, тоді як нелінійні мережі та байєсові моделі є більш ефективними для складних завдань, що вимагають обробки великих обсягів даних або роботи в умовах невизначеності [23, 24].

## 2.2 Ресурси, які потрібні для створення різних моделей нейромереж

Створення нейромереж різних типів потребує різного рівня складності, ресурсів і кількості спеціалістів у команді. Далі розглянемо вимоги до кожного типу нейромереж, включаючи необхідне обладнання, етапи проекту та спеціалістів, залучених до процесу.

**Лінійні та нелінійні нейромережі.** Для реалізації лінійних і нелінійних нейромереж необхідні такі ресурси:

- Обладнання:
- Комп'ютер або сервер з GPU: для тренування невеликих моделей достатньо потужного комп'ютера з одним GPU, тоді як для більших моделей може знадобитися сервер з декількома GPU або кластер обчислювальних вузлів [23, 24].
- Пам'ять: мінімум 16 ГБ оперативної пам'яті для комфортної роботи з середніми обсягами даних.
- Диски: SSD для швидкого зчитування та запису даних.
- Персонал:
- 1-2 Data Scientists: для створення архітектури нейромережі та її налаштування.
- 1 Data Engineer: для збору, обробки та підготовки даних.
- 1 ML Engineer: для оптимізації моделі та її впровадження в продакшн [23].

Етапи проекту:

1. Збір вимог: визначення цілей проекту та бізнес-задач.
2. Збір і підготовка даних: очищення, нормалізація та підготовка даних до навчання.
3. Розробка моделі: створення архітектури нейромережі.
4. Навчання моделі: налаштування гіперпараметрів і навчання.
5. Тестування та валідація: оцінка продуктивності моделі [22].
6. Оптимізація: покращення ефективності та зменшення часу навчання.

7. Розгортання: впровадження моделі у виробниче середовище.
8. Моніторинг і підтримка: перевірка продуктивності моделі на реальних даних.

**Нечіткі нейромережі.** Нечіткі нейромережі, що використовують нечітку логіку для роботи з неточними або неоднозначними даними, потребують таких ресурсів:

Обладнання:

- Комп'ютер або сервер із GPU: середньої потужності комп'ютер з GPU достатній для тренування моделей середнього масштабу [25].
- Пам'ять: зазвичай 16-32 ГБ ОЗП залежно від обсягу даних.
- Диски: SSD для швидкої обробки даних.
- Персонал:
  - 1-2 Data Scientists: для розробки нечіткої логіки та її інтеграції з нейромережею.
  - 1 ML Engineer: для розгортання та оптимізації моделей.
  - 1 Software Engineer: для інтеграції з іншими системами [22].

Етапи проекту:

1. Збір вимог: визначення задач, де нечітка логіка буде ефективною.
2. Збір даних і фазифікація: розробка функцій належності та нечітких множин.
3. Створення нечітких правил: визначення правил для обробки даних.
4. Інтеграція з нейромережею: поєднання нечіткої логіки з нейронними шарами [25].
5. Навчання та тестування: тренування та перевірка моделі.
6. Оптимізація та оцінка: налаштування моделі для покращення продуктивності.
7. Розгортання та підтримка: впровадження у виробниче середовище та моніторинг.

**Деревоподібні нейромережі.** Деревоподібні нейромережі (дерева рішень) потребують таких ресурсів:

Обладнання:

- Комп'ютер або сервер із GPU: сервер середньої потужності підходить для обробки моделей середнього масштабу.
- Пам'ять: від 16 ГБ ОЗП залежно від складності та обсягу даних.
- Диски: SSD або HDD для зберігання великих наборів даних [27].
- Персонал:
  - 1-2 Data Scientists: для розробки та тестування моделі.
  - 1 ML Engineer: для оптимізації та розгортання.
  - 1 Data Engineer: для підготовки великих обсягів даних [23].

Етапи проекту:

1. Збір вимог: визначення проблем, де структура дерева є ефективною.
2. Збір та обробка даних: підготовка даних до навчання.
3. Розробка моделі: створення архітектури дерева рішень.
4. Навчання: налаштування навчання та обчислення переходів між вузлами [27].
5. Тестування: оцінка точності моделі.
6. Оптимізація: покращення продуктивності.
7. Розгортання: впровадження у виробниче середовище.
8. Підтримка та моніторинг: стеження за ефективністю моделі.

**Байєсові нейромережі.** Байєсові нейромережі, які використовують ймовірнісний підхід, потребують таких ресурсів:

Обладнання:

- Сервери з потужними GPU: для обчислення вибірок і апостеріорних розподілів [28].
- Пам'ять: 32 ГБ ОЗП і більше для великих моделей.
- Диски: SSD для швидкого оброблення великих обсягів даних [26].
- Персонал:
  - 2-3 Data Scientists: для розробки моделі, аналізу результатів та оцінки невизначеності.
  - 1-2 ML Engineers: для оптимізації, навчання та розгортання.

- 1 Data Engineer: для управління даними.
- 1 Researcher з ймовірнісного моделювання: для налаштування байєсових методів [26, 28].

Етапи проекту:

1. Збір вимог: аналіз потреби в оцінці невизначеності.
2. Збір даних: підготовка великих обсягів даних до навчання.
3. Розробка архітектури: створення моделі з ймовірнісними шарами.
4. Навчання: навчання методами Монте-Карло або іншими ймовірнісними методами [28].
5. Тестування та оцінка: оцінка точності та невизначеності.
6. Оптимізація: підвищення продуктивності моделі.
7. Розгортання: складний етап через підтримку ймовірнісних обчислень.
8. Підтримка та оновлення: регулярний моніторинг для забезпечення надійності.

Висновок. Розробка нейромереж потребує різних ресурсів, залежно від рівня складності моделі. Для простих моделей, як лінійні та нелінійні нейромережі, потрібна невелика команда та стандартне обладнання. Складніші проекти, такі як байєсові нейромережі, потребують більше обчислювальних ресурсів і більшої команди спеціалістів з ймовірнісного моделювання [23, 24].

### **2.3 Вибір моделі та базові алгоритми створення нейромереж**

Для проведення експерименту вибір пав на нелінійну нейромережу, оскільки вона дозволяє використовувати складніші алгоритми та активаційні функції, що важливо для вирішення задач кібербезпеки. Нелінійна нейромережа надає гнучкість і потужність, необхідні для розпізнавання патернів та аналізу великих обсягів даних, що робить її ідеальним вибором для систем, здатних до адаптації в умовах змінного середовища загроз [24, 25]. Для кращого розуміння переваг нелінійної нейромережі розглянемо базові алгоритми створення різних типів нейромереж, починаючи з лінійної.

## 2.4 Створення моделі лінійної нейромережі

Лінійні нейромережі є найпростішими за архітектурою, оскільки вони складаються лише з лінійних шарів без використання нелінійних активаційних функцій. Така структура робить їх обмеженими для складних завдань, але водночас вони є гарною відправною точкою для базових завдань, де зв'язок між вхідними та вихідними даними є лінійним [23, 24]. Нижче наведено основні кроки для створення лінійної нейромережі:

Кроки для створення лінійної нейромережі:

Збір і підготовка даних:

- Збирають набір даних для навчання мережі.
- Розділяють дані на навчальний та тестовий набори.
- Нормалізують або стандартизують дані для полегшення навчання [22].

Створення структури нейромережі:

- Модель складається з одного або кількох лінійних шарів.
- У лінійній нейромережі кожен шар виконує лінійне перетворення над вхідними даними без застосування нелінійних функцій [24].

Налаштування ваг і зміщень:

- Ініціалізуються ваги та зміщення (bias) для кожного нейрона.
- Для ініціалізації часто використовують випадкові значення ваг.

Функція активації:

Лінійна модель не використовує нелінійних активаційних функцій, тому вихід нейрона визначається за формулою 2.1, представленою візуально на рисунку 2.1:

$$y = W * X + b \quad (2.1)$$

де,  $W$  - матриця ваг,

$X$  - вектор вхідних даних,



$b$  - вектор зміщень.

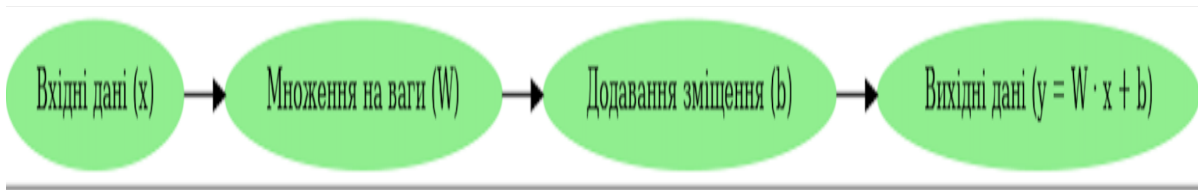


Рис. 2.1 Візуальне відображення формули (2.1)

де,  $W$  - Матриця ваг. Кожен елемент цієї матриці визначає вплив окремого входу на результат.

$X$  - Вектор вхідних даних. Це набір значень, які передаються як вхід у модель.

$b$  - Вектор зміщень (bias). Цей компонент додається, щоб забезпечити зміщення моделі, що дозволяє їй краще підлаштовуватися під дані.

$y$  - Вихідні дані. Це результат обчислення, отриманий після застосування ваг і додавання зміщень.

Як це працює:

- Обчислення: Лінійна модель виконує множення матриці ваг  $W$  на вектор вхідних даних  $x$ , а потім додає вектор зміщень  $b$ .
- Це рівняння можна застосовувати для задач регресії, класифікації (до застосування активаційної функції) або як базову операцію в нейронних мережах.

Особливість лінійної моделі:

- Лінійна модель не використовує нелінійних активаційних функцій, тому результат завжди є лінійним відносно входу  $X$ .
- Це обмежує її можливості працювати зі складними, нелінійними залежностями в даних.

Вибір функції втрат:

Вибір функції втрат залежить від задачі; наприклад, середньоквадратична помилка (Mean Squared Error, MSE) підходить для задач регресії [23].

Алгоритм оптимізації:

- Використовується алгоритм градієнтного спуску або його вдосконалені версії (Adam, RMSprop) для оновлення ваг у процесі навчання.

Навчання моделі:

- Навчання здійснюється за допомогою алгоритму зворотного поширення помилки, що оновлює ваги для зменшення функції втрат. Процес навчання повторюється кілька епох до досягнення бажаної точності [23, 24].

Оцінка та тестування:

- Перевіряється продуктивність моделі на тестовому наборі даних. При необхідності виконуються додаткові етапи налаштування моделі [24].

Якщо уявити візуалізацію нейромережі, можна відобразити у нашій роботі, як показано на рисунку 2.2 та 2.3 наступні схеми для лінійної нейромережі:

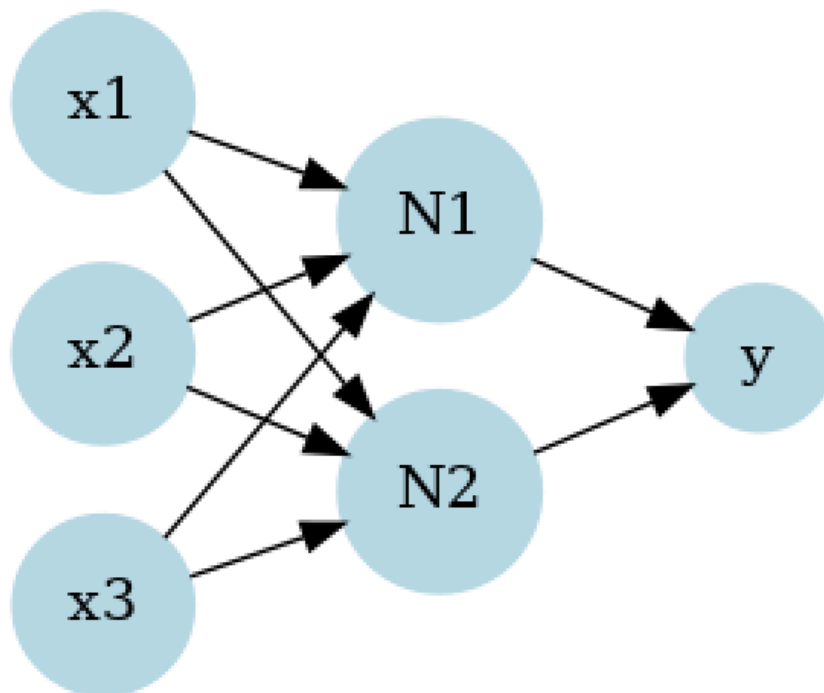


Рис. 2.2 Передача сигналу через один шар нейронів

Ця схема ілюструє передачу сигналу через єдиний шар нейронів у лінійній нейромережі

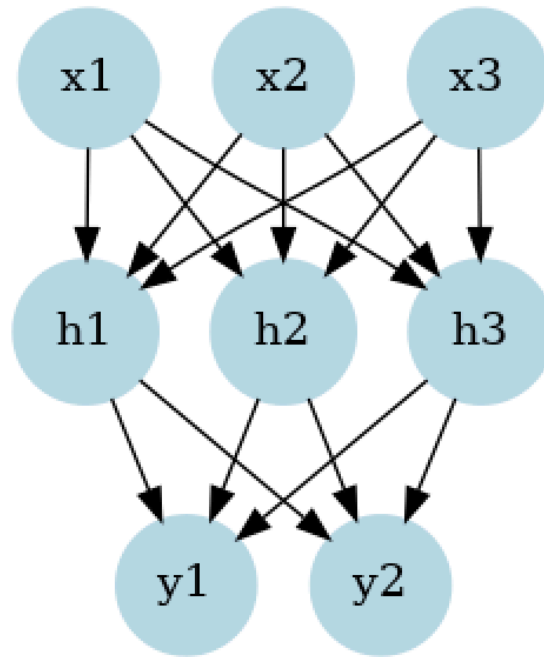


Рис 2.3 Схема роботи багатошарової нейромережі

Структура створення простої лінійної нейромережі за допомогою кодування на мові програмування Python з використанням PyTorch:

```
import torch
import torch.nn as nn
import torch.optim as optim

# Визначення простої лінійної нейронної мережі
class LinearModel(nn.Module):
    def __init__(self, input_size, output_size):
        super(LinearModel, self).__init__()
        self.linear = nn.Linear(input_size, output_size)
    def forward(self, x):
        return self.linear(x)
```

```

# Ініціалізація моделі
input_size = 1 # Залежить від ваших даних
output_size = 1 # Залежить від задачі
model = LinearModel(input_size, output_size)

# Вибір функції втрат і оптимізатора
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Навчання моделі
for epoch in range(100): # Кількість епох можна змінювати
    inputs = torch.randn(100, input_size) # Генерація випадкових даних для
    прикладу
    targets = 2 * inputs + 1 # Просте лінійне співвідношення для навчання
    optimizer.zero_grad() # Обнулення градієнтів
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward() # Обчислення градієнтів
    optimizer.step() # Оновлення ваг
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/100], Loss: {loss.item():.4f}')

# Оцінка на нових даних
test_input = torch.tensor([[1.0]])
predicted = model(test_input)
print(f'Prediction for input 1.0: {predicted.item()}')

```

Пояснення:

де, nn.Linear - реалізація лінійного шару.

`nn.MSELoss` - функція втрат для задач регресії.

`optimizer.step()` - оновлення ваг відповідно до градієнта.

Цей приклад демонструє, як створити та навчити просту лінійну нейромережу для задач регресії. Завдяки простій архітектурі, лінійна нейромережа може швидко навчатися на невеликих наборах даних, однак для складніших задач у кібербезпеці, що вимагають глибокого аналізу даних, рекомендується використовувати нелінійні моделі [24].

## 2.5 Створення моделі нелінійної нейромережі

Створення нелінійної нейромережі включає використання нелінійних активаційних функцій, що надають моделі здатність моделювати складні зв'язки в даних. Нелінійні мережі можуть мати кілька шарів із різними функціями активації, що дозволяє вирішувати більш складні задачі, як-от обробка зображень або текстів, на відміну від лінійних моделей, які обмежені у своїй здатності до обробки складних патернів [23, 24]. Нижче наведено основні кроки для створення нелінійної нейромережі.

Кроки для створення нелінійної нейромережі:

Збір і підготовка даних:

- Підготовка набору даних та розділення даних на навчальний і тестовий набори.
- Застосування нормалізації або стандартизації для покращення процесу навчання [22].

Створення структури мережі:

- Визначення кількості шарів і нейронів у кожному шарі.
- Додавання нелінійних активаційних функцій (наприклад, ReLU, Sigmoid, Tanh) після кожного шару, щоб надати мережі здатність моделювати нелінійні залежності між даними [24].

Налаштування ваг і зміщень:

- Ініціалізація ваги випадковими значеннями для забезпечення варіативності в навчанні.

Вибір функції активації:

Використання функції, яка додає нелінійність, наприклад - ReLU (Rectified Linear Unit), представлена формулою 2.2:

$$f(x) = \max(0, x) \quad (2.2)$$

де, простота обчислення: ReLU є дуже простою функцією — вона повертає вхідне значення, якщо воно позитивне, або 0, якщо значення негативне.

Відсутність експоненційної обчислювальної складності: На відміну від Sigmoid або Tanh, ReLU не потребує обчислення складних математичних функцій, що робить її швидшою.

Чому ReLU популярна:

Зменшення ризику затухання градієнта:

- У випадку використання Sigmoid або Tanh, градієнти можуть ставати дуже малими для великих або малих значень  $x$ , що ускладнює навчання глибоких моделей.
- ReLU допомагає уникнути цієї проблеми, оскільки для позитивних  $x$  градієнт є сталим (111).

Спарсність активації:

- ReLU має властивість створювати спарсність (sparsity) у мережі, оскільки для значень  $x < 0$  вихід  $f(x) = 0$ . Це допомагає моделям бути більш ефективними, активуючи лише певні нейрони.

Обмеження ReLU:

Проблема "мертвих нейронів":

- Якщо значення вхідного сигналу постійно негативне, нейрон перестає "активуватися", і градієнт стає нульовим.
- Це може призводити до проблем із навчанням частини мережі.

Варіанти ReLU. Leaky ReLU: вносить невеликий нахил для негативних  $x$ , щоб уникнути "мертвих нейронів", представлено формулою 2.3:

$$f(x) = \begin{cases} x, & \text{якщо } x > 0 \\ \alpha x, & \text{якщо } x \leq 0 \end{cases} \quad (2.3)$$

де,  $\alpha$  - невелике додатне значення (зазвичай 0.01).

Таким чином, ReLU є однією з найпоширеніших активаційних функцій для прихованих шарів у сучасних нейронних мережах, і її популярність зумовлена простотою, швидкістю обчислення та ефективністю у зменшенні ризику затухання градієнта, візуально представлено на рисунку 2.4 [23].

Sigmoid представлено формулою 2.4:

$$f(x) = \frac{1}{1+e^{-x}} \quad (2.4)$$

де, діапазон значень:  $f(x) \in (0,1)$ . Ця властивість робить Sigmoid особливо корисною для задач класифікації, де потрібно моделювати ймовірності.

Плавний перехід: Функція має S-подібну форму, яка забезпечує плавний перехід від 0 до 1.

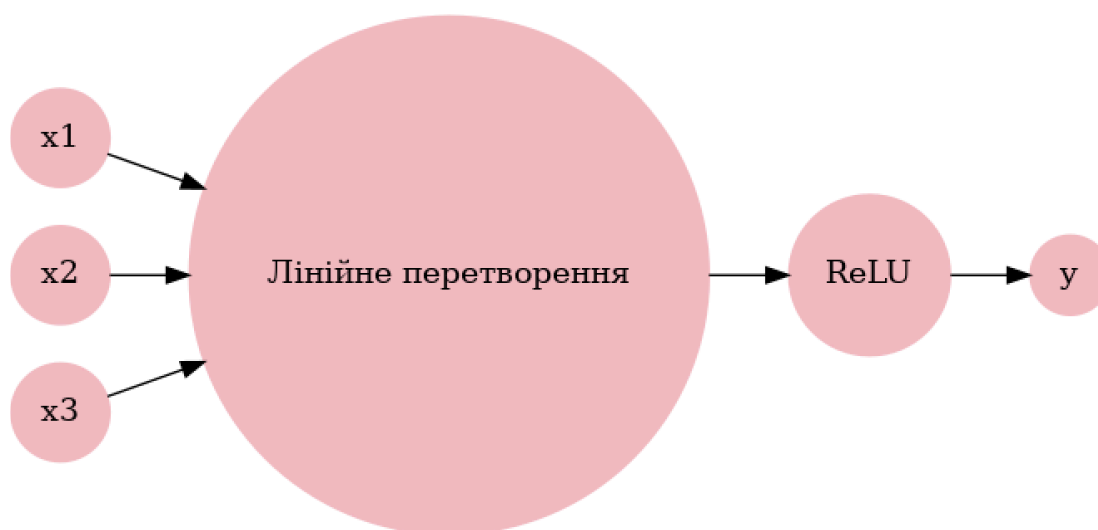


Рис 2.4 Нелінійна активація в одному шарі

Застосування:

- Часто використовується для вихідного шару у задачах бінарної класифікації, оскільки значення  $f(x)$  можна інтерпретувати як ймовірність приналежності до класу.
- Наприклад, у логістичній регресії.

Обмеження:

- Затухання градієнта: Для дуже великих або дуже малих значень  $x$  функція стає майже пласкою, а її похідна — близькою до нуля. Це може ускладнювати навчання глибоких моделей.
- Вихідний діапазон: Значення завжди додатні, що може створювати зміщення в навчанні.

Вибір функції втрат:

Оберіть відповідну функцію втрат залежно від задачі (наприклад, `CrossEntropyLoss` для класифікації або `MSELoss` для регресії), візуально представлених на рисунках 2.5 та 2.6 [23].

- Алгоритм оптимізації:

Використовуйте вдосконалені алгоритми оптимізації, такі як Adam або RMSprop, які забезпечують стабільне та швидке оновлення ваг [23, 24].

Навчання моделі:

- Запустити процес навчання з використанням алгоритму зворотного поширення помилки для оновлення ваг на кожній ітерації [24].

Оцінка та тестування:

- Перевірити модель на тестовому наборі даних і оцінити її точність за допомогою відповідних метрик [23].

Tanh (Гіперболічний тангенс) представлений формулою 2.5:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$



де, діапазон значень:  $f(x) \in [-1,1]$ . Це робить Tanh корисним для завдань, де потрібно мати як позитивні, так і негативні значення виходу.

Порівняння із Sigmoid:

- Tanh є масштабованою версією Sigmoid, але зі значеннями у від'ємному діапазоні.
- Має центр симетрії в 0, що допомагає моделювати дані, де є як негативні, так і позитивні кореляції.

Застосування:

- Використовується в прихованих шарах нейронних мереж, особливо коли важливо, щоб вихід сигналу був у балансі між негативними і позитивними значеннями.
- Підходить для задач регресії чи сигналів, де потрібно нормалізувати дані в діапазоні  $[-1,1]$ .

Обмеження:

- Затухання градієнта: Подібно до Sigmoid, для дуже великих або малих значень xxx функція стає майже пласкою, що ускладнює навчання.
- Обчислювальна складність: Через наявність експоненційних обчислень функція є більш ресурсомісткою.

Висновок:

- Sigmoid добре підходить для задач класифікації завдяки вихідному діапазону  $(0,1)$ , який легко інтерпретується як ймовірність [23].
- Tanh є кращим для прихованих шарів, де важливо мати виходи, збалансовані між негативними і позитивними значеннями, завдяки діапазону  $[-1,1]$  [22].

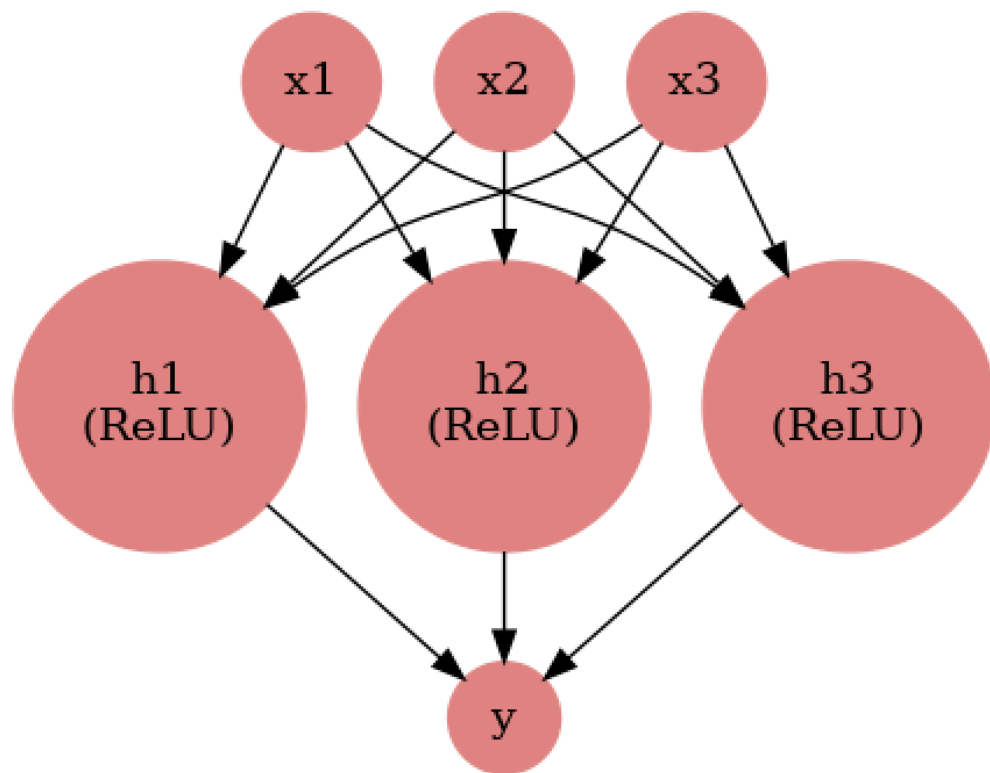


Рис. 2.5 Багатошарова нелінійна нейромережа

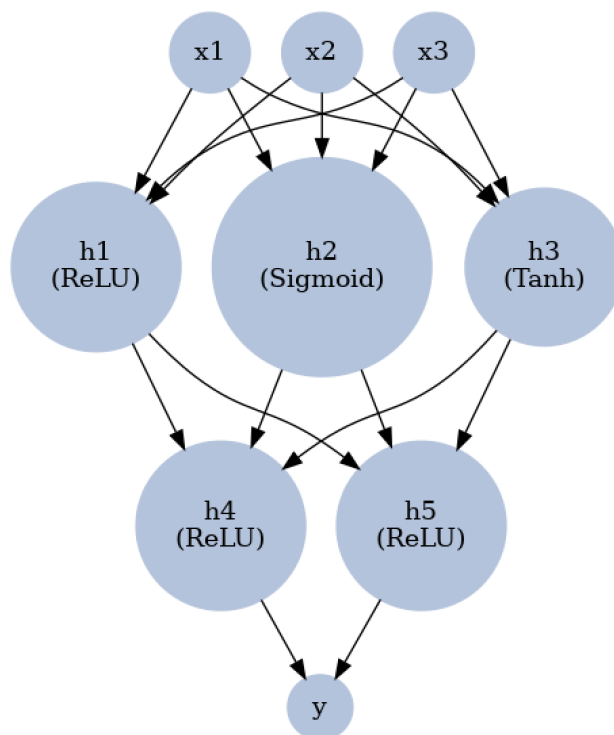


Рис. 2.6 Нелінійна нейромережа зі складною структурою

Схема створення нелінійної нейромережі на мові програмування Python з використанням PyTorch:

```
import torch
import torch.nn as nn
import torch.optim as optim

# Визначення нелінійної нейронної мережі
class NonLinearModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NonLinearModel, self).__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.activation1 = nn.ReLU() # Нелінійна активація
        self.layer2 = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        x = self.layer1(x)
        x = self.activation1(x)
        x = self.layer2(x)
        return x

# Ініціалізація моделі
input_size = 1 # Вхідні особливості
hidden_size = 10 # Кількість нейронів у прихованому шарі
output_size = 1 # Вихідна кількість
model = NonLinearModel(input_size, hidden_size, output_size)

# Вибір функції втрат і оптимізатора
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```

# Навчання моделі
for epoch in range(100): # Кількість епох
    inputs = torch.randn(100, input_size) # Приклад випадкових даних
    targets = 3 * inputs**2 + 2 # Нелінійне співвідношення для навчання
    optimizer.zero_grad() # Обнулення градієнтів
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward() # Обчислення градієнтів
    optimizer.step() # Оновлення ваг
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/100], Loss: {loss.item():.4f}')

# Оцінка на нових даних
test_input = torch.tensor([[1.0]])
predicted = model(test_input)
print(f'Prediction for input 1.0: {predicted.item()}')

```

де, `nn.ReLU()` — функція активації ReLU, що додає нелінійність, дозволяючи моделі ефективніше навчатися на складних даних. Прихований шар збільшує складність мережі, що підходить для обробки нелінійних зв'язків у даних [22].

Оптимізатор Adam забезпечує стабільне та швидке оновлення ваг, що робить його зручним для задач, де необхідне швидке сходження [24].

Цей приклад демонструє, як створити та навчити просту нелінійну нейромережу, яка здатна адаптуватися до більш складних патернів у даних. Така архітектура є базовою для багатьох сучасних застосувань, зокрема в кібербезпеці, де часто потрібне моделювання складних зв'язків та аналіз великих обсягів інформації.

## 2.6 Створення моделі нечіткої нейромережі

Створення нечіткої нейромережі, або нейромережі, що використовує нечітку логіку, є складнішим завданням, оскільки потребує інтеграції методів нечіткої логіки з класичними нейронними мережами. Нечітка логіка дозволяє обробляти неточні або неоднозначні вхідні дані, що є особливо корисним у складних системах, де традиційні методи можуть бути неефективними [25].

Основні концепції нечіткої нейромережі:

1. Нечітка логіка: використовує нечіткі множини та правила для прийняття рішень.
2. Нечіткі правила: формулюються у вигляді "Якщо-Тоді" (наприклад, "Якщо температура висока, тоді швидкість вентилятора велика").
3. Фазифікація: процес перетворення чітких значень у нечіткі.
4. Дефазифікація: процес перетворення нечітких значень у чітке вихідне значення [25].

Структура нечіткої нейромережі

Нечітка нейромережа може складатися з:

- Вхідного шару: фазифікує вхідні дані.
- Прихованого шару: застосовує нечіткі правила.
- Вихідного шару: дефазифікує результат, щоб отримати чітке значення [24].

Кроки для створення нечіткої нейромережі:

1. Визначення вхідних змінних і нечітких множин:
2. Визначення, які змінні будуть входами, та створіть нечіткі множини для них (наприклад, для "температура" можуть бути "низька", "середня", "висока") [25].
3. Визначення нечітких правил:
4. Створюємо набір правил на основі знань або даних, що описують взаємозв'язок між входами та виходами [24].

Фазифікація входів:

Перетворюємо чіткі вхідні значення у нечіткі за допомогою функцій належності, таких як трикутні або трапецієподібні функції, візуально етапи представлені на рисунках 2.7, 2.8 та 2.9 [26].

Застосування нечітких правил:

Використовує логічні оператори для обчислення значень кожного правила на основі фазифікованих вхідних значень [25].

Дефазифікація:

Обчислюємо підсумковий вихід за допомогою методу, такого як метод центру ваги, щоб отримати чітке значення [27].

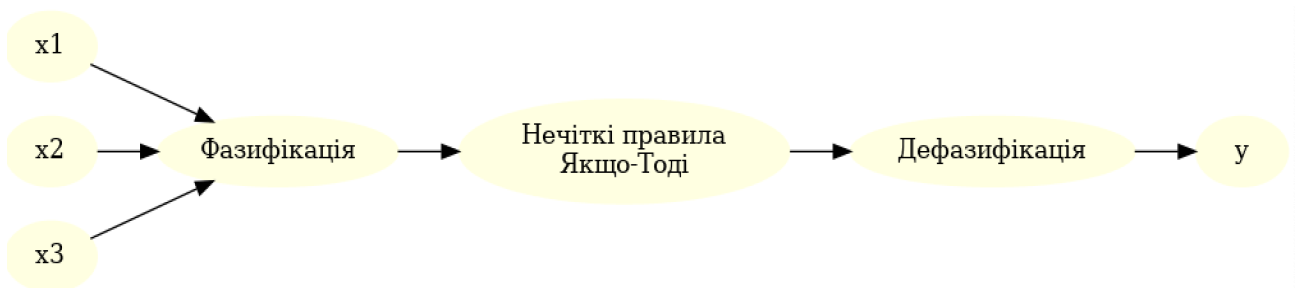


Рис. 2.7 Нечітка нейромережа - взаємодія у вхідному шарі

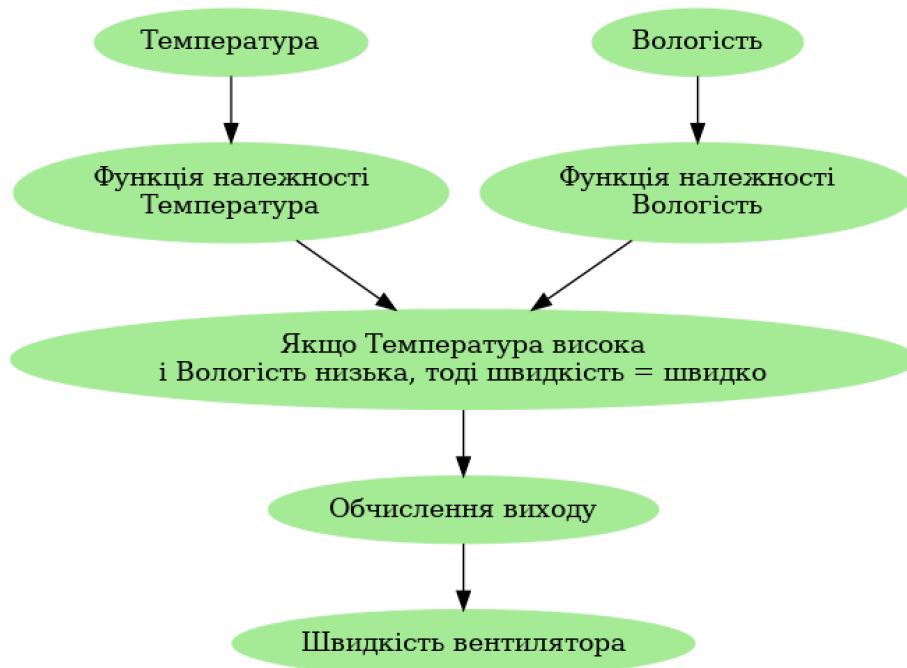


Рис. 2.8 Нечітка нейромережа - обробка нечітких правил



Рис. 2.9 Нечітка неймережа - повний цикл обробки

Приклад схеми реалізації нечіткої неймережі з використанням Python та бібліотеки skfuzzy:

```
import numpy as np
import skfuzzy as fuzz
import skfuzzy.control as ctrl

# Визначення вхідних змінних
temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

# Визначення нечітких множин за допомогою функцій належності
temperature['low'] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature['medium'] = fuzz.trimf(temperature.universe, [20, 50, 80])
temperature['high'] = fuzz.trimf(temperature.universe, [50, 100, 100])
humidity['low'] = fuzz.trimf(humidity.universe, [0, 0, 50])
humidity['medium'] = fuzz.trimf(humidity.universe, [20, 50, 80])
humidity['high'] = fuzz.trimf(humidity.universe, [50, 100, 100])
fan_speed['slow'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [20, 50, 80])
```

```

fan_speed['fast'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Визначення нечітких правил
rule1 = ctrl.Rule(temperature['high'] & humidity['high'], fan_speed['fast'])
rule2 = ctrl.Rule(temperature['medium'] & humidity['medium'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['low'] | humidity['low'], fan_speed['slow'])

# Створення системи управління
fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan_simulation = ctrl.ControlSystemSimulation(fan_ctrl)

# Введення значень і обчислення виходу
fan_simulation.input['temperature'] = 75
fan_simulation.input['humidity'] = 60
fan_simulation.compute()
print(f"Fan speed: {fan_simulation.output['fan_speed']:.2f}")

```

де, Antecedent — вхідна змінна з нечіткими множинами.

Consequent — вихідна змінна.

trimf — трикутна функція належності, що задає нечіткі множини.

Rule — правило, що описує взаємозв'язок між входами та виходами, наприклад: "Якщо температура висока і вологість висока, то швидкість вентилятора швидка".

Ця структура дозволяє створювати та використовувати нечіткі системи для прийняття рішень у реальних умовах, де вхідні дані можуть бути нечіткими або неповними. Використання нечіткої логіки у нейромережах корисне для задач, де точність даних є неповною або варіюється в широких межах [25, 26].



## 2.7 Створення моделі деревоподібної нейромережі

Деревоподібні нейромережі поєднують концепції класичних дерев рішень з нейронними мережами, що дозволяє використовувати переваги обох підходів. Дерева рішень легко інтерпретуються, але обмежені у здатності моделювати складні зв'язки, тоді як нейронні мережі є більш гнучкими, хоча й менш інтуїтивно зрозумілими [27]. Комбінація цих методів дає змогу створювати потужні моделі, які можуть ефективно вирішувати задачі, що потребують як точності, так і інтерпретованості.

Основні концепції деревоподібної нейромережі

1. Дерево рішень: структура, яка використовує вузли для прийняття рішень на основі певних правил.
2. Нейронна мережа: складається з нейронів, які перетворюють вхідні дані через ваги, зміщення та функції активації [22, 23].
3. Комбінація: деревоподібна нейромережа використовує вузли дерева, де кожен вузол представлений як нейрон або невелика нейронна мережа.

Підходи до створення деревоподібних нейромереж:

1. Neural Decision Forest: об'єднання дерев рішень з нейронними мережами, де кожне дерево функціонує як окремий блок у моделі [27].
2. Differentiable Decision Trees: дерева, які можна навчати за допомогою зворотного поширення градієнта, що забезпечує плавний перехід між вузлами [28].

Кроки для створення деревоподібної нейромережі:

1. Створення основних вузлів дерева:
2. Кожен вузол у дереві може бути реалізований як невелика нейронна мережа або логістична регресія, яка приймає рішення на основі вхідних даних.
3. Визначення функцій переходів:
4. Використовуйте сигмоїдні або інші плавні функції для визначення ймовірностей переходу між вузлами, що дозволяє створити

диференційовану модель, яка може навчатися за допомогою методів градієнтного спуску [27].

Навчання моделі: використовуємо дані для навчання нейронів, що представляють вузли, і функцій переходів між вузлами. Обчислюємо вихідні значення на основі ваг вузлів і застосовуємо зворотнє поширення помилки для оновлення ваг, візуально представлених на рисунках 2.10, 2.11 та 2.12 [22, 24].

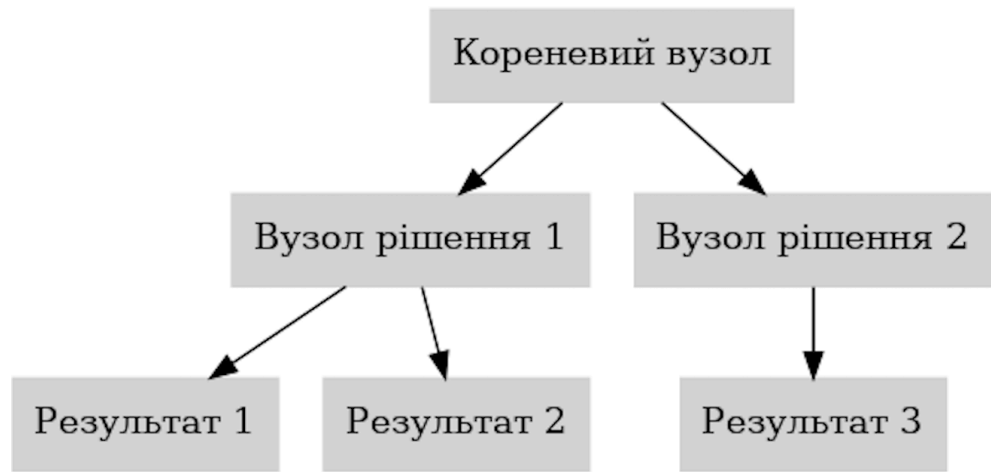


Рис. 2.10 Деревоподібна нейромережа - прийняття рішень

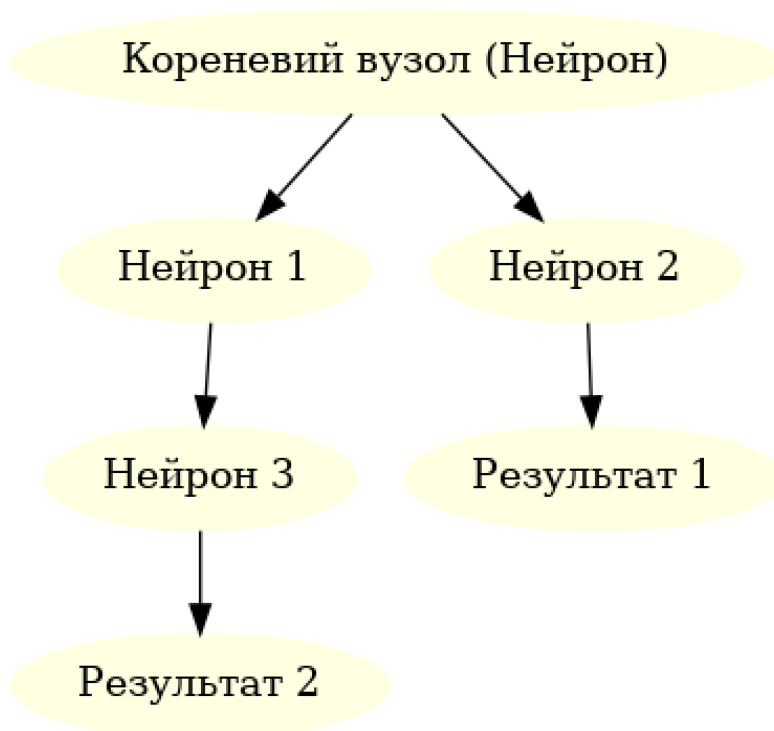


Рис 2.11 Деревоподібна нейромережа з нейронними вузлами

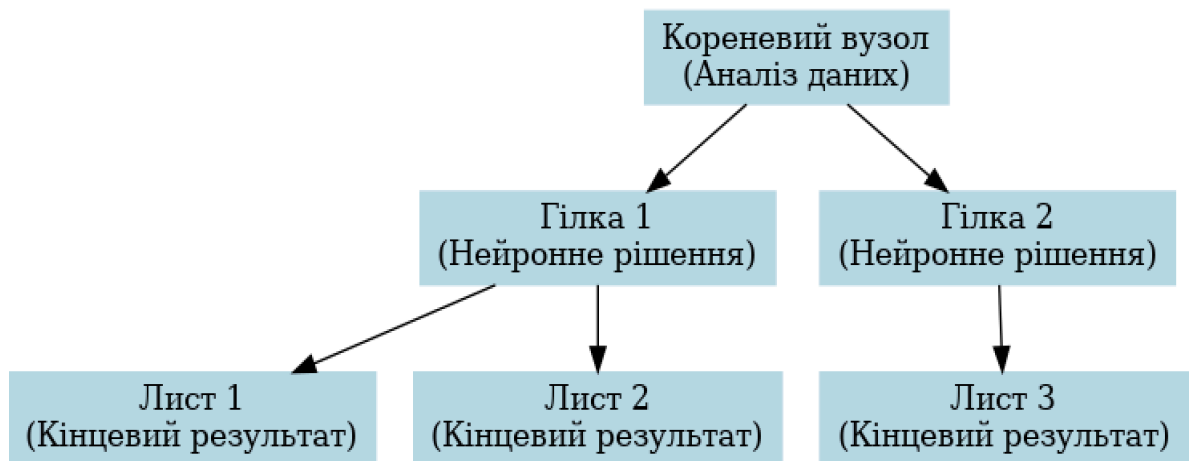


Рис 2.12 Повний процес роботи деревоподібної нейромережі

Просту схему деревоподібної нейромережі на мові програмування Python можна представити у такому вигляді:

```

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

# Вузол дерева як нейронна мережа
class TreeNode(nn.Module):
    def __init__(self, input_size):
        super(TreeNode, self).__init__()
        self.decision = nn.Linear(input_size, 1) # Вузол приймає рішення на основі
входу
    def forward(self, x):

# Використання сигмоїдної функції для плавного переходу
    return torch.sigmoid(self.decision(x))
  
```

```

# Древоподібна нейромережа
class TreeNeuralNetwork(nn.Module):
    def __init__(self, input_size, depth):
        super(TreeNeuralNetwork, self).__init__()
        self.depth = depth
        self.nodes = nn.ModuleList([TreeNode(input_size) for _ in range(2 ** depth -
1)])

# КІЛЬКІСТЬ вузлів
    def forward(self, x):
        node_index = 0
        for level in range(self.depth):
            decision = self.nodes[node_index](x)
            if decision > 0.5:
                node_index = node_index * 2 + 1 # Перехід до лівої гілки
            else:
                node_index = node_index * 2 + 2 # Перехід до правої гілки
        return decision

# Ініціалізація та навчання моделі
input_size = 2 # Наприклад, для даних з двома особливостями
depth = 3 # Глибина дерева
model = TreeNeuralNetwork(input_size, depth)
criterion = nn.BCELoss() # Функція втрат для класифікації
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Приклад даних для навчання
inputs = torch.randn(100, input_size)
targets = torch.randint(0, 2, (100, 1)).float()

```

```
# Навчання моделі
for epoch in range(100):
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/100], Loss: {loss.item():.4f}')
```

де, `TreeNode`: вузол, який приймає рішення за допомогою лінійного шару і сигмоїдної функції активації. Це дозволяє використовувати вузол як базовий елемент, який приймає рішення на основі входу.

`TreeNeuralNetwork`: структура дерева, де кожен вузол приймає рішення і переходить до наступного рівня вузлів залежно від результату. Така архітектура дозволяє поєднати рішення в деревоподібній структурі.

Навчання: здійснюється з використанням алгоритму зворотного поширення помилки, що дозволяє моделі вчитися приймати рішення за допомогою градієнтного спуску.

Ця базова реалізація деревоподібної нейромережі демонструє, як поєднати вузли дерева з нейронною мережею. Для складніших задач можна використовувати диференційовані підходи, такі як `Neural Decision Forest`, що інтегрують нейронні мережі.

## 2.8 Створення моделі байєсових нейромереж

Байєсові нейромережі поєднують теорію ймовірностей з нейронними мережами, що дозволяє моделювати невизначеність у даних. Це особливо корисно для задач, де важливо оцінити ймовірнісні залежності та працювати з даними, що можуть містити шум або недостовірні значення [26, 28]. Такі мережі можуть обчислювати ймовірності різних гіпотез, базуючись на наявній інформації, і оновлювати їх у міру надходження нових даних, що робить їх ефективним інструментом для складних задач, як-от медичне прогнозування, класифікація текстів і управління ризиками, візуально представлених на рисунках 2.13, 2.14 та 2.15 [28].

Основні концепції байєсової нейромережі:

1. Байєсова теорема: використовується для оцінки ймовірностей, що визначають зв'язки між різними змінними в мережі.
2. Ймовірнісні залежності: байєсові мережі будують графічні структури, що представляють залежності між змінними.
3. Апостеріорне оновлення: у байєсових нейромережах ймовірності оновлюються на основі нових даних, що дозволяє мережі адаптуватися до змін [26].

Підходи до створення байєсових нейромереж:

1. Bayesian Neural Networks (BNNs): класичні нейронні мережі, інтегровані з байєсовими принципами, що дозволяє оцінювати невизначеність виходу.
2. Bayesian Networks (BN): графічні моделі, які використовують вузли для представлення змінних і ймовірнісних залежностей між ними [28].

Кроки для створення байєсової нейромережі

Визначення змінних та зв'язків:

- Визначаємо змінні та їх ймовірнісні залежності, що будуть моделю. Кожен вузол у графі буде представляти окрему змінну з її умовною ймовірністю [28].

Ініціалізація ймовірностей:

- Ініціалізуємо апостеріорні ймовірності для кожної змінної на основі наявних даних або попередніх знань [26].

Вибір методу навчання:

- Використовуємо метод Монте-Карло, такі як ланцюги Маркова (МСМС) або варіаційний інференс, для обчислення ймовірнісних розподілів та оновлення ймовірностей вузлів [27].

Оновлення ймовірностей:

- Після введення нових даних оновимо апостеріорні ймовірності для кожного вузла. Це дозволяє моделі адаптуватися до змін у даних [28].

Оцінка та тестування:

- Використовуємо тестовий набір даних для оцінки моделі. Оцінка виконується за допомогою метрик, що дозволяють врахувати невизначеність (наприклад, логарифмічна функція правдоподібності) [26].

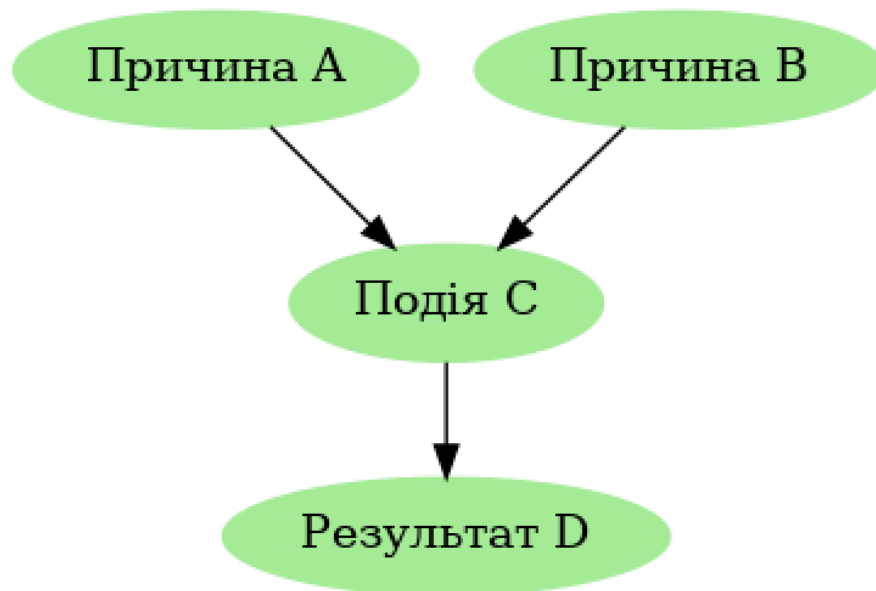


Рис 2.13 Базова структура Бейєсової нейромережі

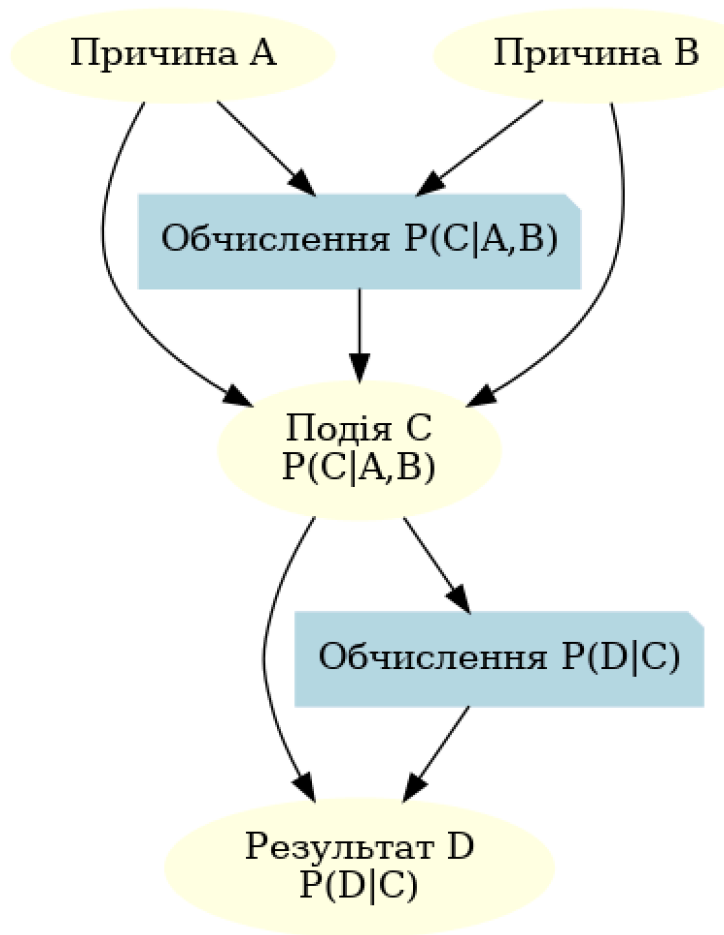


Рис 2.14 Бейсова неймережа, яка демонструє обчислення ймовірностей

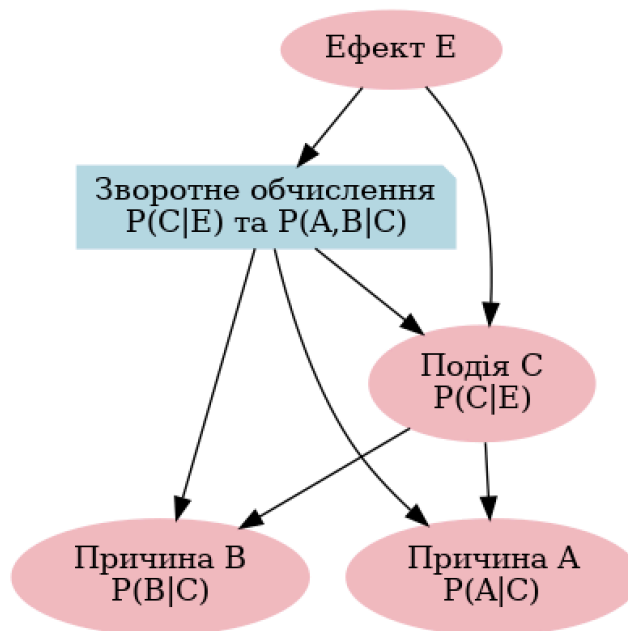


Рис 2.15 Бейсова неймережа для інтеграції зворотного обчислення



Приклад структури байєсової нейромережі на мові програмування Python з використанням Pyro (бібліотеки для байєсового інференсу):

```
import pyro
import pyro.distributions as dist
from pyro.infer import SVI, Trace_ELBO
from pyro.optim import Adam
import torch
import torch.nn as nn

# Приклад байєсової нейронної мережі
class BayesianNeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(BayesianNeuralNetwork, self).__init__()
        self.hidden_layer = nn.Linear(input_size, hidden_size)
        self.output_layer = nn.Linear(hidden_size, output_size)
    def forward(self, x):
        hidden = torch.relu(self.hidden_layer(x))
        output = self.output_layer(hidden)
        return output

# Модель із ймовірнісними вагами
def model(x_data, y_data):

    # Пріори для вагів
    weight_prior = dist.Normal(torch.zeros_like(x_data), torch.ones_like(x_data))
    bias_prior = dist.Normal(torch.zeros_like(y_data), torch.ones_like(y_data))
    priors = {'hidden_layer.weight': weight_prior, 'output_layer.bias': bias_prior}
    lifted_module = pyro.random_module("module", BayesianNeuralNetwork, priors)
    bayesian_model = lifted_module()
```

```

# Вихід із ймовірністю
with pyro.plate("data", len(x_data)):
    prediction = bayesian_model(x_data)
    pyro.sample("obs", dist.Normal(prediction, 1.0), obs=y_data)

# Оптимізація та інференс
x_data = torch.randn(100, 1)
y_data = 3 * x_data + 1 + 0.1 * torch.randn(100, 1) # Випадкові дані з шумом
pyro.clear_param_store()
adam_params = {"lr": 0.01}
optimizer = Adam(adam_params)
svi = SVI(model, guide=model, optim=optimizer, loss=Trace_ELBO())

# Навчання моделі
for epoch in range(1000):
    loss = svi.step(x_data, y_data)
    if epoch % 100 == 0:
        print(f"Epoch [{epoch}/1000] Loss: {loss:.4f}")

```

Пояснення:

Пріори: визначають апостеріорні ймовірності для ваг та зміщень. Наприклад, вага та зміщення мають розподіл Нормальний.

Pyro Plate: створює незалежні зразки для кожного входу, що дозволяє застосовувати методи інференсу.

SVI (Stochastic Variational Inference): метод інференсу, який дозволяє здійснювати варіаційний байєсовий інференс за допомогою стохастичного градієнтного спуску.

Цей приклад демонструє, як створити просту байєсову нейромережу, яка здатна адаптуватися до невизначеності у вхідних даних і забезпечує надійні прогнози в

умовах змінних даних. Байєсові нейромережі є потужним інструментом для завдань, де необхідно моделювати невизначеність та ймовірнісні залежності [26, 28].

## **2.9 Алгоритм створення моделі нейромережі для нашого експерименту**

Для реалізації задачі експерименту було обрано трансформерну архітектуру від OpenAI. Ця модель чудово підходить для аналізу текстових даних і інтеграції з такими системами, як Zabbix, завдяки своїм потужним можливостям обробки природної мови (NLP) та здатності навчатися на складних даних [22, 24].

OpenAI надає доступ до своїх моделей через RESTful API, що дозволяє інтегрувати їх у різноманітні системи, зокрема для моніторингу, аналізу даних та автоматизації процесів. В основі цих моделей лежить трансформерна архітектура, яка є різновидом нелінійних нейронних мереж та використовує механізм самоуваги (self-attention), що забезпечує ефективну роботу з послідовностями даних [28]. Це особливо корисно для задач, що потребують розуміння контексту, таких як автоматичне реагування на події кібербезпеки або створення звітів на основі отриманих даних.

Переваги трансформерної архітектури:

1. Гнучкість та масштабованість: Трансформери легко адаптуються до різних типів текстових задач і дозволяють використовувати великі обсяги даних для навчання [22, 23].
2. Механізм самоуваги: Самоувага дозволяє моделі одночасно аналізувати різні частини тексту, що підвищує якість розуміння контексту та виявлення ключових фраз у великій послідовності [24].
3. Паралельність: На відміну від традиційних рекурентних мереж, трансформери можуть обробляти текстові дані паралельно, що значно пришвидшує навчання та обробку даних у реальному часі [28].

Основні компоненти трансформерної архітектури:

- Вхідні дані: Текстовий вхід розбивається на послідовності токенів, які модуль ембеддингу перетворює на багатовимірні вектори.
- Позиційне кодування: Додає інформацію про порядок слів у тексті, що є важливим для розуміння структури речення [23].
- Механізм самоуваги: Обчислює зв'язки між словами за допомогою матриць запитів, ключів та значень, забезпечуючи моделі здатність звертати увагу на різні частини тексту при обробці кожного слова [28].
- Механізм багатоголової уваги: Дозволяє моделі одночасно зосереджуватися на різних аспектах вхідних даних, що збільшує точність і ефективність обробки [24].
- Об'ємний шар: Складається з нелінійних шарів з активаційною функцією ReLU, яка дозволяє моделі працювати зі складними патернами в даних, візуально представлених на рисунку 2.16 [22].

Кроки для створення трансформерної архітектури:

1. Ініціалізація та підготовка даних: Отримання вхідних текстових даних, їх токенізація та перетворення в ембеддинги для подальшого навчання.
2. Реалізація механізму самоуваги: Обчислення матриць запитів, ключів та значень, що дозволяє моделі звертати увагу на релевантні частини тексту для кожного слова.
3. Створення багатоголової уваги: Процес самоуваги виконується для кількох голів, результати яких об'єднуються та проходять через лінійний шар для отримання підсумкового виходу [23].
4. Додавання об'ємного шару: Застосування повнозв'язних шарів з активаційними функціями, що додає моделі нелінійності.
5. Навчання моделі: Використання функції втрат (наприклад, Cross-Entropy) і алгоритму оптимізації (такого як Adam) для оновлення ваг моделі на основі обчислених градієнтів [28].

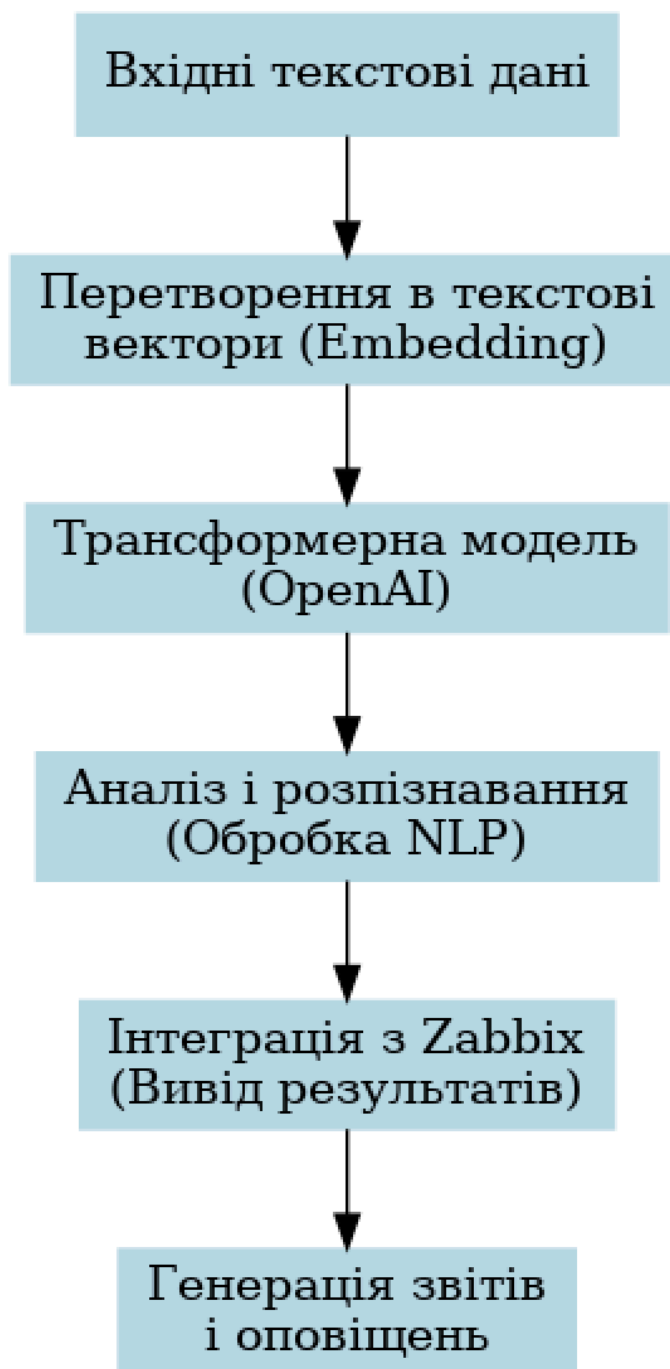


Рис 2.16 Інтеграція трансформера

Трансформерна архітектура революціонізувала обробку природної мови (NLP) завдяки своїй здатності працювати з текстовими даними у великих масштабах та ефективно враховувати контекст. Ось детальний опис, як трансформери працюють у задачах NLP:

1. Представлення тексту

Трансформери починають із перетворення текстових даних у числовий вигляд:

Токенізація: Розділення тексту на окремі слова, частини слів чи символи (токени).

Ембеддинг: Кожен токен перетворюється у вектор із багатьма вимірами, що описують його семантику.

Текст: "Я люблю NLP"

Токени: ["Я", "люблю", "NLP"]

Ембеддинги: [[0.2, 0.8, ...], [0.1, 0.5, ...], [0.3, 0.7, ...]]

## 2. Позиційне кодування

Оскільки трансформери не мають природного порядкування (як у рекурентних мережах), використовується позиційне кодування:

Позиційні вектори додаються до ембеддингів, щоб повідомити моделі про порядок слів у реченні.

## 3. Механізм самоуваги (Self-Attention)

Самоувага є серцем трансформера. Вона визначає, на які інші частини тексту потрібно звернути увагу під час обробки кожного токена:

Для кожного токена обчислюються три матриці: Query (Q), Key (K) і Value (V).

Обчислюється "вага уваги" між токенами, представлена формулою 2.6 як:

$$\text{Attention}(Q, K, V) = \text{Softmaks}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.6)$$

де, Q - запити,

K - ключі,

V- значення,

$d_k$  - розмірність ключа.

Наприклад:

Слово "Я" у фразі "Я люблю NLP" може отримати високу вагу до "люблю", а не до "NLP".

#### 4. Багатоголова увага (Multi-Head Attention)

Для кращого моделювання різних зв'язків у тексті використовується кілька "голів уваги".

Кожна голова фокусується на різних аспектах тексту (наприклад, граматики, семантика).

#### 5. Нормалізація та об'ємний шар (Feed-Forward Layer)

Нормалізація: Забезпечує стабільність та ефективність навчання.

Об'ємний шар: Після уваги кожен токен обробляється окремо через шар із нелінійною функцією активації (наприклад, ReLU).

#### 6. Енкодер і декодер

Трансформер складається з двох частин:

Енкодер: Аналізує вхідний текст і створює багатовимірне представлення (ембеддинг) тексту.

Декодер: Генерує вихідний текст (у задачах, таких як переклад чи генерація тексту).

#### 7. Функція втрат і оптимізація

Використовується функція втрат (наприклад, Cross-Entropy) для вимірювання точності передбачення.

Оптимізація (наприклад, Adam) оновлює ваги, щоб зменшити похибку.

Приклад застосування трансформера в NLP:

Мовне моделювання: Передбачення наступного слова.

Переклад: Генерація тексту на іншій мові.

Класифікація: Аналіз тональності тексту (позитивний/негативний).

Відповіді на запитання: Пошук інформації в тексті.

Трансформери, такі як BERT та їх похідні, дозволяють автоматизувати складні завдання NLP з високою точністю завдяки потужності механізму самоуваги.

Приклад схеми реалізації механізму самоуваги на мові Python з використанням PyTorch:

```
import torch
import torch.nn as nn

# Модуль самоуваги
class SelfAttention(nn.Module):
    def __init__(self, embed_size, heads):
        super(SelfAttention, self).__init__()
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        assert self.head_dim * heads == embed_size, "Embed size must be divisible by heads"

        self.values = nn.Linear(self.head_dim, embed_size, bias=False)
        self.keys = nn.Linear(self.head_dim, embed_size, bias=False)
        self.queries = nn.Linear(self.head_dim, embed_size, bias=False)
        self.fc_out = nn.Linear(embed_size, embed_size)

    def forward(self, values, keys, query, mask):
        N = query.shape[0]
        value_len, key_len, query_len = values.shape[1], keys.shape[1], query.shape[1]

        # Розділення ембеддингів на кілька голів
        values = values.reshape(N, value_len, self.heads, self.head_dim)
        keys = keys.reshape(N, key_len, self.heads, self.head_dim)
        queries = query.reshape(N, query_len, self.heads, self.head_dim)
        energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])
```



```
if mask is not None:
    energy = energy.masked_fill(mask == 0, float("-1e20"))
    attention = torch.softmax(energy / (self.embed_size ** (1/2)), dim=3)
    out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(N,
query_len, self.embed_size)
    out = self.fc_out(out)
return out
```

Цей код демонструє механізм самоуваги, що є основою трансформерної архітектури. Завдяки функції уваги модель може звертати увагу на різні частини тексту, що підвищує ефективність обробки та генерації тексту.

## **Висновок до розділу №2**

У ході виконання роботи було проведено комплексний аналіз основних архітектур нейронних мереж, таких як лінійні, нелінійні, нечіткі, деревоподібні та байєсові моделі. Кожен тип нейромереж має свої особливості, переваги та недоліки, що визначає його застосовність для різних завдань. Аналіз показав, що для задач, пов'язаних з обробкою текстових даних та інтеграцією з системами моніторингу кібербезпеки, найбільш ефективним є використання трансформерної архітектури, зокрема моделей, розроблених OpenAI.

Трансформерна архітектура відзначається високою здатністю обробляти контекст текстових послідовностей, що є критично важливим для завдань автоматизації моніторингу та реагування на події кібербезпеки. Використання механізмів самоуваги та багатоголової уваги дозволяє моделі одночасно аналізувати кілька аспектів вхідного тексту, що підвищує якість розуміння та точність прогнозування. Крім того, можливість паралельної обробки даних у трансформерах робить цю архітектуру особливо придатною для роботи з великими обсягами інформації в реальному часі [22, 23].

## Порівняння нейромереж і трансформерної архітектури

Тип моделі	Переваги	Обмеження	Застосування
Лінійні	Простота, швидкість	Немає нелінійності	Регресія, базова класифікація
Нелінійні	Гнучкість, складні залежності	Високі обчислювальні витрати	Розпізнавання образів, NLP
Байєсові	Урахування невизначеності, ймовірності	Складність, ресурсоємність	Прогнозування, медицина
Нечіткі	Робота з нечіткими даними	Менша точність	Управління, експертні системи
Деревоподібні	Інтерпретованість, швидкість	Обмежені можливості моделювання	Класифікація, оцінка ризиків
Трансформерні (нелінійні)	Висока ефективність обробки великих послідовностей даних; паралельна обробка; чудова масштабованість	Потребують великих обчислювальних ресурсів; складність реалізації	NLP (обробка тексту), машинний переклад, інтеграція з системами моніторингу (наприклад, Zabbix)

Інтеграція з системами на зразок Zabbix завдяки RESTful API OpenAI значно розширює функціональні можливості моніторингу та дозволяє автоматизувати обробку повідомлень і формування звітів. Це забезпечує оперативне реагування на події кібербезпеки, знижує навантаження на персонал і підвищує ефективність управління інцидентами.

Підсумовуючи, робота над створенням нейромережі для моніторингу кіберзагроз продемонструвала, що трансформерна архітектура є найкращим вибором для реалізації таких систем, завдяки своїм унікальним можливостям обробки природної мови та адаптивності до різних сценаріїв використання, аналіз для порівняння наглядно представлений у таблиці 2.1. Реалізація трансформерів забезпечує інтегровану систему моніторингу, яка відповідає сучасним вимогам кібербезпеки та автоматизує управління ризиками з високим рівнем точності та надійності [28].

## РОЗДІЛ 3

# СТВОРЕННЯ І НАЛАДКА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПОВІЩЕННЯ ПОДІЙ КІБЕРБЕЗПЕКИ

### 3.1 Опис систем Zabbix та OpenAI

У цьому розділі представлено розробку системи для автоматичного оповіщення про події кібербезпеки, яка поєднує можливості моніторингу на основі Zabbix, хмарних обчислень через сервер на ngrok та інтелектуальної обробки повідомлень з використанням нейромереж OpenAI. Основною метою цієї системи є швидке та автоматичне реагування на інциденти, надання детальних інструкцій для усунення проблем і оперативне оповіщення відповідного персоналу.

Складові системи:

1. Zabbix Cloud: Система моніторингу, яка дозволяє виявляти та фіксувати події кібербезпеки. Для нашого експерименту було піднято ноду на платформі Zabbix Cloud, яка слугує основною системою моніторингу для збору інформації про помилки.
2. Ngrok сервер: Сервер на платформі ngrok, що забезпечує захищений зв'язок між локальним сервером і зовнішніми сервісами, а також спрощує інтеграцію Zabbix з API OpenAI для обробки повідомлень.
3. OpenAI API: Платформа, яка надає доступ до потужних нейронних моделей для обробки та розшифровки повідомлень. У нашій системі OpenAI відповідає за розпізнавання коду помилки, автоматичний аналіз даних і генерацію текстових інструкцій для усунення проблем.

Логіка роботи системи:

4. Моніторинг: Zabbix виявляє подію (наприклад, помилку в базі даних SQL) та фіксує її у своїй системі. Для цього використовуються спеціальні тригери, налаштовані на конкретні події та помилки.

5. Відправка даних на сервер: Інформація про виявлену помилку надсилається на сервер, піднятий на ngrok, де встановлено програмний код для обробки.
6. Інтеграція з OpenAI: Сервер ngrok через API OpenAI надсилає код помилки на розшифровку. Нейромережа OpenAI аналізує отриману інформацію, визначає характер помилки та генерує рекомендації для її усунення у текстовому форматі.
7. Повернення інструкцій: OpenAI надсилає розшифровану інформацію на сервер, який пересилає відповідь до Zabbix.
8. Оповіщення персоналу: Zabbix, отримавши відповідь від сервера, направляє текстові інструкції для усунення помилки до спеціального Telegram-каналу, створеного для оповіщення відповідного персоналу.
9. Ця система забезпечує інтерактивний ланцюг моніторингу, автоматизованого аналізу подій і оперативного інформування персоналу, що значно прискорює процеси виявлення й усунення інцидентів кібербезпеки, візуально представлена на рисунку 3.1.



Рис. 3.1 Схема необхідних елементів для системи моніторингу

Моніторинг (Zabbix) → Сервер (ngrok) → Інтерфейс API (OpenAI) → Аналіз та розшифровка (неймережа) → Зворотній зв'язок до сервера → Передача результатів (Zabbix) → Оповіщення персоналу (Telegram канал)

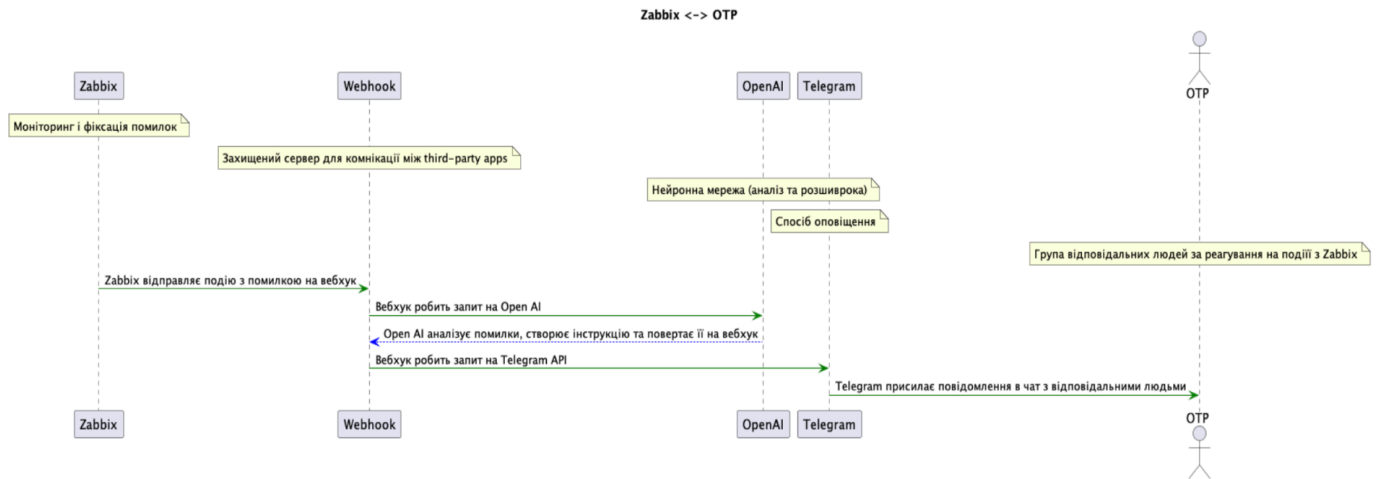


Рис. 3.2 Розширена логічна схема роботи системи моніторингу за допомогою ШІ

Блок-схема, візуально представлена на рисунку 3.2, наочно відображає етапи взаємодії елементів системи, забезпечуючи зрозумілу структуру роботи для автоматизованого оповіщення про події кібербезпеки.

### 3.2 Етапи експерименту

У цьому розділі детально описано етапи налаштування та запуску системи штучного інтелекту для автоматизованого оповіщення про події кібербезпеки наведеному на Рис. 1.1. Основними кроками експерименту є введення неймережі в контекст задачі, налаштування з'єднань між компонентами системи та перевірка роботи механізму автоматизації оповіщень.

Етап 1. Введення неймережі в контекст задачі.

На першому етапі було необхідно підготувати неймережу OpenAI до специфіки задачі кібербезпеки, яку вона вирішуватиме. Це включає

забезпечення правильного контексту для обробки помилок, пов'язаних із подіями в системі моніторингу Zabbix. Для цього виконуються наступні дії:

1. Налаштування API OpenAI: Нейромережі надається доступ до специфічної інформації про типові коди помилок, які можуть виникати в системі, і можливих інструкцій для усунення таких помилок. Важливо, щоб модель мала базову інформацію про типи повідомлень і особливості системи кібербезпеки, зокрема про специфіку обробки подій у Zabbix [24, 28].

Визначення контексту для обробки помилок: Щоб забезпечити більш точне розуміння нейромережею особливостей помилок у Zabbix, їй надається розширений контекст. У навчанні моделі доданий такий параметр:

```
var context = 'Контекст: Ти експерт в роботі із Zabbix зі стажем більше 10 років. Твоя задача максимально доступно пояснювати помилки, які я передаватиму тобі.';
```

```
var message = context + ' ' + 'Помилка: ' + params.message;
```

2. Цей контекст дозволяє нейромережі підходити до аналізу помилок із експертної точки зору та надавати роз'яснення в доступній і зрозумілій формі. Модель отримує попередній досвід роботи із системою Zabbix, що допомагає їй формулювати рекомендації та інструкції більш детально і точно [23, 27].

3. Адаптація відповіді: Нейромережа OpenAI формує відповіді у зрозумілому і корисному для системи моніторингу форматі. Кожна відповідь включає інструкції для усунення проблем, рекомендовані дії та чітко структуровану інформацію, що допомагає персоналу оперативно реагувати на інциденти [25, 26].

Введення нейромережі в контекст задачі з додаванням специфічних налаштувань забезпечує високий рівень точності та ефективності роботи системи на наступних етапах експерименту.

Етап 2. Написання коду на Python для web-hook AI (з'єднання систем Zabbix і ШІ).

Другий етап експерименту полягав у створенні коду для web-hook на мові Python, що забезпечує з'єднання системи моніторингу Zabbix із нейромережею OpenAI. Використання web-hook дозволяє автоматично надсилати дані про події безпосередньо з Zabbix до нейромережі OpenAI, яка аналізує та обробляє інформацію в режимі реального часу. Таким чином, web-hook виступає в ролі зв'язкової ланки між Zabbix та ШІ, забезпечуючи постійний обмін даними та автоматизацію обробки повідомлень.

Мета коду web-hook. Нам треба використати переваги механізму, який дозволяє одним додаткам чи сервісам автоматично надсилати дані або повідомлення іншим додаткам чи сервісам у реальному часі. Це метод передачі інформації через HTTP-запити, що ініціюються подією в одній системі та отримуються іншою системою без необхідності постійного опитування.

Як працює Webhook?

Відправник (Ініціатор події):

1. Додаток або сервіс, який генерує подію, наприклад, зміна статусу, помилка або оновлення даних.
2. Відправник виконує HTTP-запит (POST або GET) до вказаної URL-адреси.

Одержувач (Сервер для обробки веб-хуку):

1. Сервер, який слухає вхідні HTTP-запити.
2. Обробляє отримані дані та виконує відповідні дії, наприклад, оновлює базу даних, надсилає повідомлення або запускає певний процес.

Основні елементи Webhook:

URL-адреса для веб-хука:

Це кінцева точка (endpoint), куди надходять HTTP-запити з даними.

Події:

Події — це тригери, які викликають відправку веб-хука. Наприклад, "новий користувач", "завершення транзакції" або "виникнення помилки".

Дані (payload):



Форматовані дані, які надсилаються одержувачу у вигляді JSON, XML або іншого формату.

HTTP-методи:

Зазвичай використовується метод POST для передачі даних.

Основні переваги використання Webhook:

- Ефективність: Замість постійного опитування сервера для перевірки змін, веб-хук повідомляє про зміни в реальному часі.
- Швидкість: Дані передаються одразу після виникнення події.
- Простота інтеграції: Легко налаштувати передачу даних між різними системами через HTTP.

Сценарії використання Webhook:

Оптимізація роботи чат-ботів:

У випадку інтеграції з Telegram API веб-хук отримує повідомлення з інформацією про нові чати, команди або відповіді.

Моніторинг:

У Zabbix веб-хук може бути використаний для надсилання оповіщень у реальному часі про події кібербезпеки.

Обробка платежів:

Наприклад, платіжна система надсилає веб-хук до магазину з інформацією про успішну транзакцію.

CI/CD (безперервна інтеграція):

У системах розробки веб-хуки тригерять автоматичне виконання завдань, наприклад, збірку коду після внесення змін у репозиторій.

Як створити Webhook?

Налаштувати сервер для отримання запитів: Наприклад, Flask, Django або FastAPI у Python.

Задати URL-адресу:

Ця адреса повинна бути доступною ззовні. Для цього можна використовувати сервіси, як-от ngrok.

Обробка даних:

Прийняти вхідні дані, виконати необхідні дії, надіслати відповідь.

Відправник налаштовується з цією URL-адресою:

У системі, яка надсилає веб-хук, вказується кінцева точка.

У нашому випадку, код web-hook написаний на мові програмування Python, виконує наступні ключові функції:

1. Надсилання запиту з помилкою до OpenAI через API: web-hook отримує інформацію про помилку, що виникла в Zabbix, і надсилає її в нейромережу OpenAI через RESTful API. Повідомлення про помилку супроводжується контекстом, який додає завдання для моделі – максимально зрозуміло пояснити суть помилки та запропонувати рекомендації щодо її усунення [23, 24].

2. Обробка відповіді від OpenAI: Після аналізу помилки нейромережею отримана відповідь із роз'ясненням та інструкціями передається назад на сервер для подальшої обробки [25, 27]. Цей механізм дозволяє інтегрувати рекомендації ШІ у загальну систему моніторингу.

3. Надсилання сповіщень через Telegram: Після обробки отриманої відповіді вона автоматично надсилається на спеціальний канал Telegram, створений для оповіщення персоналу про виявлену помилку та рекомендації з її усунення. Це забезпечує оперативне інформування відповідального персоналу та дозволяє швидко реагувати на події кібербезпеки [28].

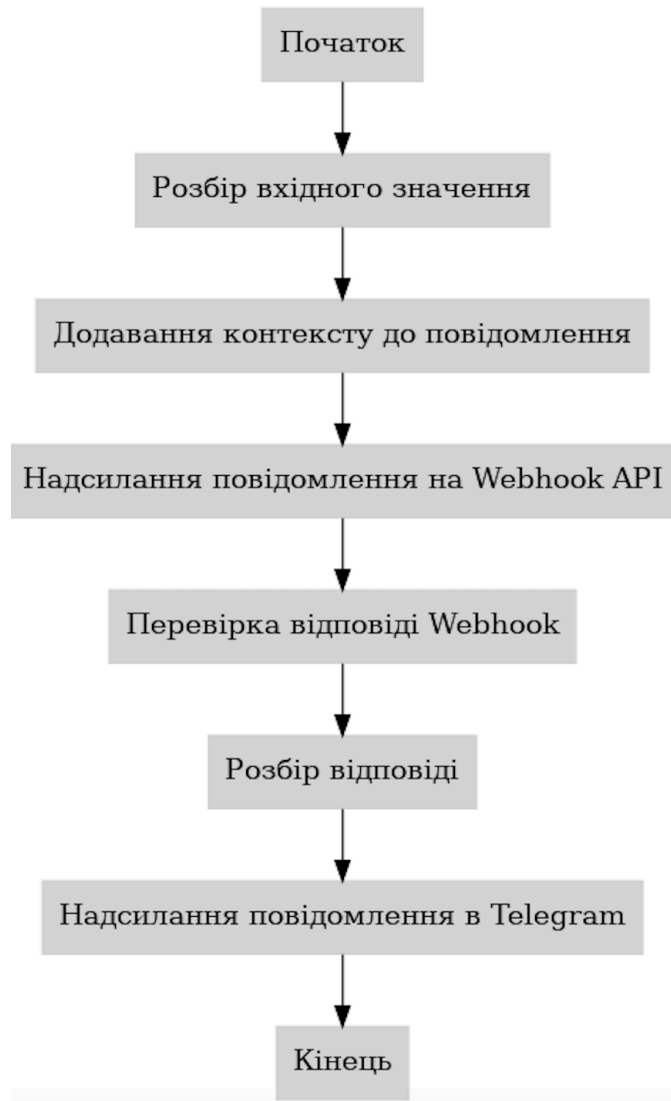


Рис. 3.3 Логічна схема роботи коду для AI web-hook

Код для AI web-hook було реалізовано на Python з використанням бібліотеки requests для відправки HTTP-запитів до API OpenAI та API Telegram, візуально представлена схема коду на рисунку 3.3. Завдяки такій реалізації забезпечується стабільний та швидкий обмін даними між системами, що значно підвищує ефективність моніторингу та усунення проблем.

Web-hook інтегрується в систему Zabbix як тригер, який автоматично активується при виникненні події. Запит до нейромережі містить контекстне повідомлення та інформацію про помилку, що дозволяє ШІ надавати детальні пояснення та конкретні кроки для усунення проблеми. Використання Python для

реалізації web-hook обумовлене його гнучкістю, підтримкою роботи з API, а також можливістю легкої інтеграції з іншими системами [26].

Таким чином, web-hook став ключовим елементом експериментальної системи, що поєднує можливості Zabbix та OpenAI для автоматизованого оброблення подій кібербезпеки.

Етап 3. З'єднання елементів в єдину систему.

На третьому етапі експерименту було необхідно об'єднати всі компоненти системи в єдиний автоматизований механізм, що забезпечує повний цикл обробки подій кібербезпеки. Цей етап включав інтеграцію всіх елементів — Zabbix, сервера ngrok для обробки веб-запитів, нейромережі OpenAI, а також Telegram для оперативного сповіщення відповідного персоналу. Завдяки такій інтеграції система може автоматично виявляти події, передавати інформацію про помилки на розшифровку в нейромережу, обробляти отримані відповіді та надсилати їх до відповідальних осіб через месенджер, етапи візуально представлені на рисунках 3.4, 3.5, 3.6 та 3.7.

Кроки для з'єднання елементів системи:

1. Інтеграція Zabbix та сервера ngrok: Спочатку було налаштовано тригери в Zabbix, які будуть активувати web-hook при виникненні певних подій. Ці тригери надсилають запит на сервер, піднятий на ngrok, що забезпечує стабільний захищений зв'язок між локальним сервером і зовнішніми сервісами. Ngrok виступає проміжною ланкою, яка пересилає повідомлення від Zabbix до нейромережі через API OpenAI [24, 26].

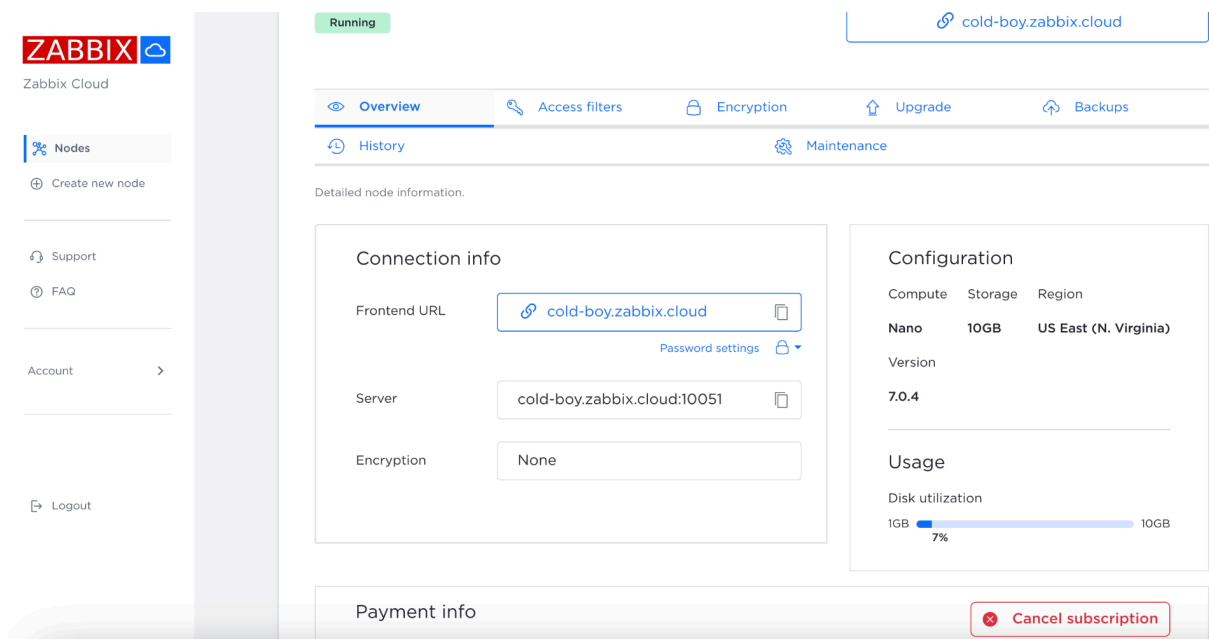


Рис. 3.4 Активація аккаунту Zabbix

2. Передача даних до OpenAI через web-hook: Після активації тригера в Zabbix web-hook пересилає інформацію про подію неймережі OpenAI, надаючи їй контекстне повідомлення з описом помилки. Неймережа аналізує отримані дані, розшифровує повідомлення про помилку та генерує інструкції для її усунення у зрозумілому текстовому форматі. Завдяки можливостям трансформерної архітектури неймережі, ця інтеграція забезпечує високу точність і оперативність обробки [22, 27].

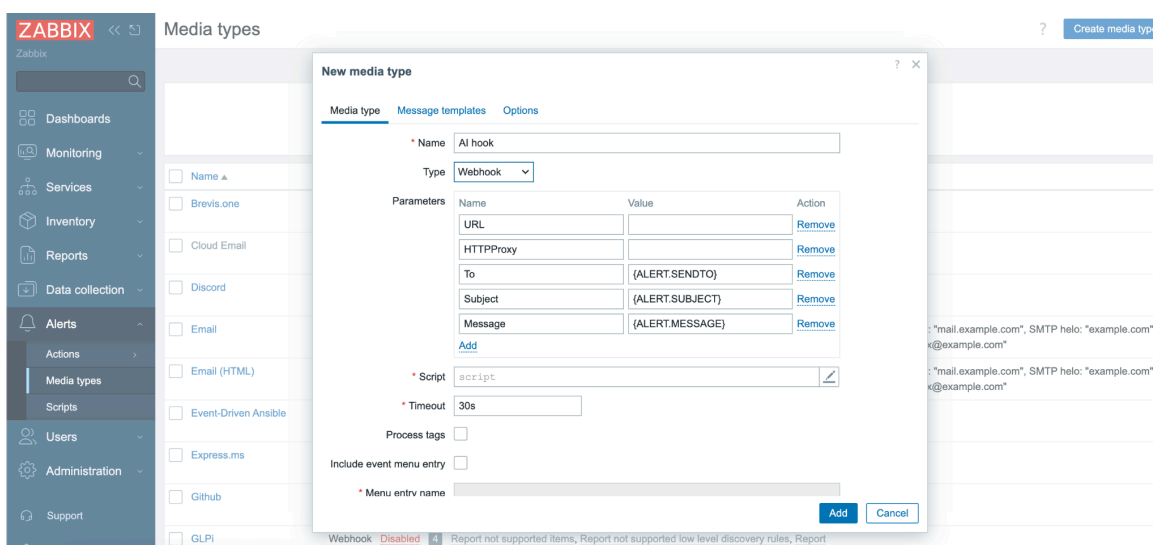


Рис. 3.5 Активація web-hook з кодом для з'єднання з ШІ

3. Відправка повідомлень через Telegram: Після отримання відповіді від OpenAI сервер ngrok передає результати в Zabbix, де вони обробляються для подальшого сповіщення. Інформація про помилку та рекомендації для її усунення автоматично відправляються на попередньо налаштований Telegram-канал, який служить засобом оповіщення для відповідального персоналу. Завдяки такому інтерактивному підходу, сповіщення про події надходять у режимі реального часу, що забезпечує своєчасне реагування на інциденти. З повним переліком помилок, які були використані в цьому експерименті і відповідями від ШІ можна ознайомитись у Додатку Н [28].

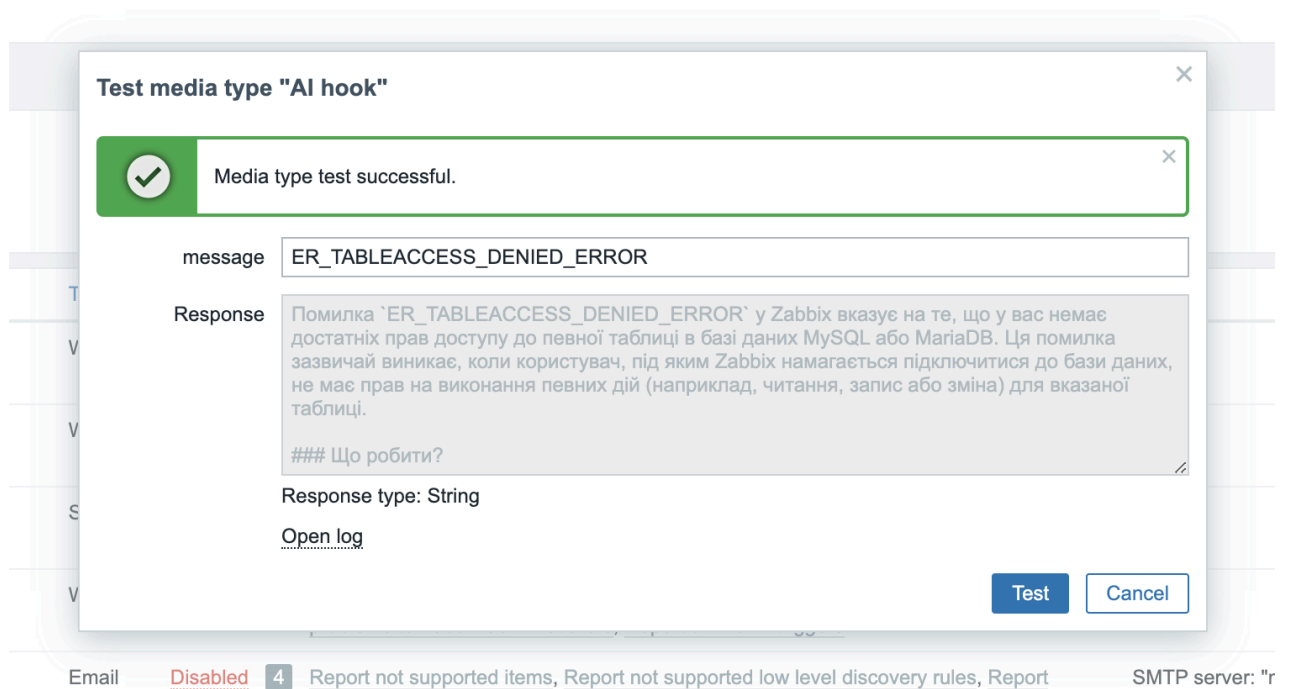


Рис. 3.6 Отримання відповіді на помилку від ШІ у Zabbix

Переваги з'єднання елементів у єдину систему:

Об'єднання компонентів у єдину систему забезпечує ефективну взаємодію між Zabbix, OpenAI і Telegram. Кожен з елементів системи виконує свою специфічну роль, забезпечуючи безперервний ланцюг автоматизованого моніторингу, аналізу та оповіщення:

- Автоматизована обробка даних: Система виявляє події та автоматично передає інформацію на обробку в нейромережу, що значно знижує навантаження на персонал [25].
- Своєчасне інформування: Завдяки інтеграції з Telegram, система забезпечує оперативне сповіщення відповідального персоналу про необхідність реагування [23].
- Гнучкість та масштабованість: Використання web-hook і API дозволяє легко розширювати систему та адаптувати її для інших сервісів і сценаріїв [24].

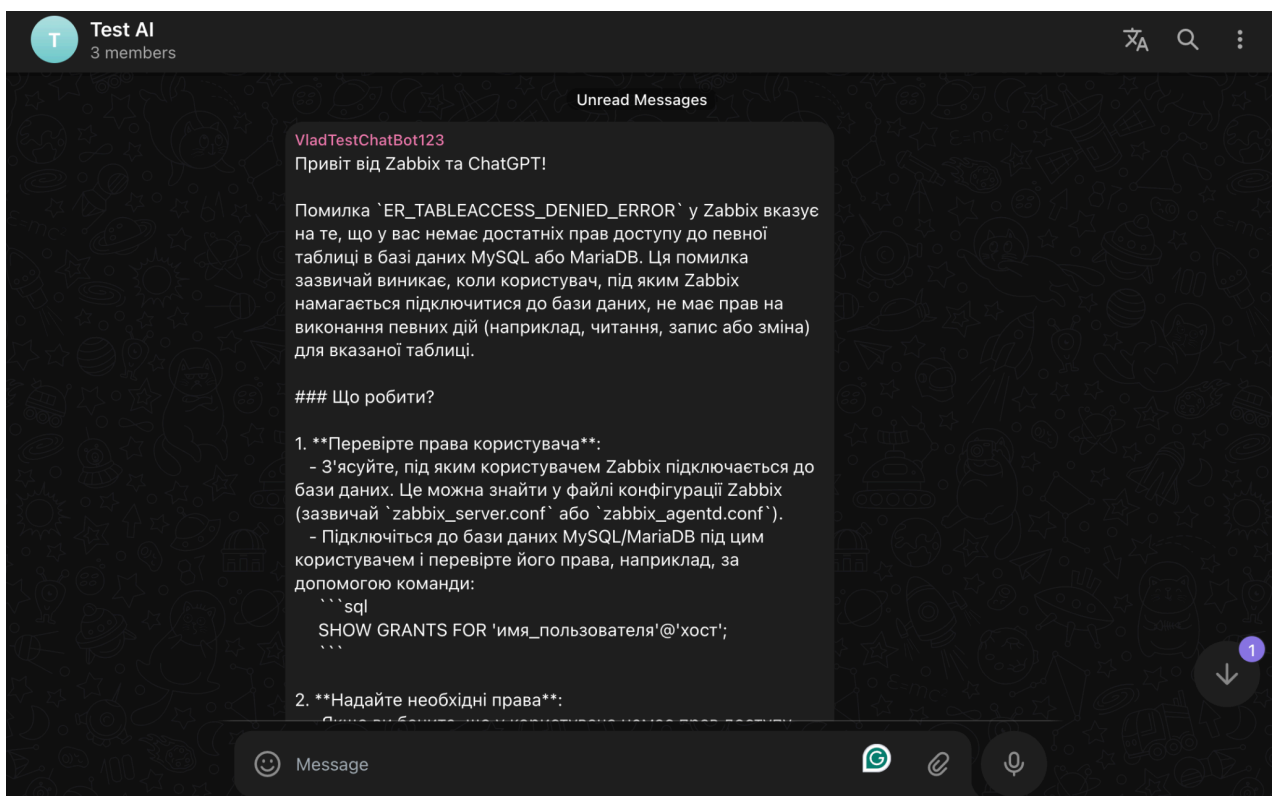


Рис. 3.7 Отримання відповіді в Telegram канал

Завдяки з'єднанню всіх компонентів у єдиний механізм система забезпечує швидке та ефективне реагування на події кібербезпеки, зменшуючи час на обробку інцидентів і підвищуючи надійність моніторингу.

Якщо Zabbix виявить, що серверна база даних недоступна, він надішле сповіщення та продовжить спроби підключитися до бази даних. Наведемо коди

помилки, використані в експерименті та передані у ШІ за допомогою нашого Webhook AI та відповіді від нейромережі OpenAI через інтерфейс Telegram каналу:

## MySQL

- CR\_CONN\_HOST\_ERROR
- CR\_SERVER\_GONE\_ERROR
- CR\_CONNECTION\_ERROR
- CR\_SERVER\_LOST
- CR\_UNKNOWN\_HOST
- ER\_SERVER\_SHUTDOWN
- ER\_ACCESS\_DENIED\_ERROR
- ER\_ILLEGAL\_GRANT\_FOR\_TABLE
- ER\_TABLEACCESS\_DENIED\_ERROR
- ER\_UNKNOWN\_ERROR

**[ПОМИЛКА: CR\_CONN\_HOST\_ERROR]**

**[ВІДПОВІДЬ ВІД НЕЙРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка **\*\*CR\_CONN\_HOST\_ERROR\*\*** в Zabbix зазвичай вказує на те, що сервер Zabbix не зміг встановити з'єднання з хостом, на якому налаштований ваш моніторинг. Це може бути викликано рядом причин:

1. **\*\*Неправильна конфігурація хосту\*\***: Перевірте, чи правильно налаштовані параметри хосту в Zabbix. Особливо важливо перевірити адресу IP або доменне ім'я хосту, а також порт, на якому працює Zabbix агента чи іншого сервісу.

2. **\*\*Проблеми з мережею\*\***: Можливо, виникають проблеми з мережею, такі як блокування портів фаєрволом, помилки в маршрутизації або біти з'єднання.



3. **\*\*Не працює Zabbix агент\*\***: Переконайтеся, що Zabbix агент запущений на цільовому хосту. Якщо агент не працює, сервер Zabbix не зможе отримати від нього дані.

4. **\*\*Призупинений сервер\*\***: Якщо хост (сервер, на якому має бути агент) вимкнений або не відповідає, Zabbix також не зможе з ним з'єднатися.

5. **\*\*Налаштування доступу\*\***: Переконайтеся, що в конфігурації Zabbix агента має бути вказано, що сервер Zabbix може до нього підключитися. Це зазвичай налаштовується в файлі конфігурації агента (zabbix\_agentd.conf) в параметрі `Server`.

Щоб вирішити цю помилку, ви можете:

- Перевірити конфігурацію хосту в інтерфейсі Zabbix.
- Переконайтеся, що мережа працює та що немає блокування з'єднання.
- Перевірити, чи запущений Zabbix агент на цільовому хосту.
- Погляньте в лог-файли Zabbix агента, щоб зрозуміти, чи є там якісь повідомлення про помилки.

Якщо ви виправите ці проблеми, з'єднання має встановитися, і помилка зникне.]

**[ПОМИЛКА: CR\_SERVER\_GONE\_ERROR]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка **\*\*CR\_SERVER\_GONE\_ERROR\*\*** в Zabbix, зазвичай, вказує на те, що клієнт не може підключитися до сервера бази даних. Це може статися з кількох причин:

1. **\*\*Сервер бази даних не запущений\*\***: Переконайтеся, що сервер бази даних (наприклад, PostgreSQL або MySQL) запущений. Ви можете перевірити його статус за допомогою команди, специфічної для вашої операційної системи.

2. **\*\*Проблеми з мережею\*\***: Існує ймовірність, що проблема може бути пов'язана із з'єднанням між Zabbix сервером і базою даних. Переконайтеся, що у

вас налаштовано правильне мережеве з'єднання, і що сервер бази даних доступний з сервера Zabbix.

3. **\*\*Перевантаження бази даних\*\***: Якщо сервер бази даних перевантажений або не може обробити всі запити, це може призвести до втрати з'єднання. Перевірте навантаження на сервері бази даних.

4. **\*\*Неправильна конфігурація\*\***: Переконайтеся, що у файлі конфігурації Zabbix (`zabbix_server.conf`) правильно вказані параметри підключення до бази даних, такі як ім'я користувача, пароль, хост і порт.

5. **\*\*Тайм-аут з'єднання\*\***: Якщо Zabbix сервер не отримує відповідь від бази даних протягом певного часу, Zabbix може закрити з'єднання. Спробуйте збільшити тайм-аут у налаштуваннях з'єднання.

Для усунення помилки спершу перевірте, чи запущено ваш сервер бази даних, і чи правильно налаштовані всі параметри підключення. Якщо проблема зберігається, проаналізуйте логи як Zabbix сервера, так і сервера бази даних, щоб отримати додаткову інформацію про причину проблеми.]

**[ПОМИЛКА: CR\_CONN\_HOST\_ERROR]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка **\*\*CR\_SERVER\_LOST\*\*** в Zabbix зазвичай вказує на те, що ваш Zabbix агент втратив з'єднання з Zabbix сервером або Zabbix проксі. Це може статися з кількох причин:

1. **\*\*Проблеми з мережею\*\***: Найпоширеніша причина — це проблеми з мережею. Якщо між агентом і сервером/проксі є тимчасові перешкоди, з'єднання може бути перервано.

2. **\*\*Перевантаження сервера\*\***: Якщо сервер або проксі отримують надто багато запитів, це може призвести до того, що він не встигає обробляти їх, і агент отримує помилку з'єднання.

3. **\*\*Неправильні налаштування\*\***: Перевірте конфігураційні файли Zabbix агента та серверу. Зокрема, важливо, щоб у файлі `zabbix\_agentd.conf` були вказані правильні параметри `Server` або `ServerActive`.

4. **\*\*Блокування брандмауером\*\***: Додатково варто перевірити, чи немає блокування на рівні брандмауера, яке перешкоджає встановленню з'єднання між агентом і сервером.

5. **\*\*Проблеми з ресурсами\*\***: Якщо система, на якій працює Zabbix агент або сервер, має проблеми з ресурсами (наприклад, брак пам'яті або процесорного часу), це може також призвести до втрати з'єднання.

### Що робити?

1. **\*\*Перевірте з'єднання\*\***: Спробуйте пінгувати сервер або проксі з машини, на якій працює Zabbix агент, щоб упевнитися, що мережа працює.

2. **\*\*Перегляньте журнали\*\***: Зайдіть у журнали Zabbix сервера та агента (наприклад, `/var/log/zabbix/zabbix\_agentd.log` для агента) і знайдіть додаткову інформацію про помилку.

3. **\*\*Перезапустіть агент\*\***: Іноді простий перезапуск Zabbix агента може відновити з'єднання.

4. **\*\*Налаштування таймаутів\*\***: Можливо, варто налаштувати таймаути, щоб уникнути перемикання з'єднання при перевантаженнях.

5. **\*\*Моніторинг ресурсів\*\***: Переконайтеся, що у вас достатньо ресурсів на сервері та агенті.

Зробивши ці кроки, ви зможете усунути причини появи помилки **\*\*CR\_SERVER\_LOST\*\*** і повернути з'єднання до нормальної роботи.]

**[ПОМИЛКА: CR\_CONN\_HOST\_ERROR**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!**

Помилка `CR\_UNKNOWN\_HOST` у Zabbix означає, що система не змогла знайти або розпізнати хост, до якого ви намагаєтесь підключитися. Це може статися з кількох причин. Розглянемо деякі з них та можливі рішення:

1. **\*\*Неправильна назва хоста\*\***: Переконайтеся, що ім'я хоста, яке ви вказали в конфігурації (наприклад, у файлі конфігурації агента або в налаштуваннях сервера), є правильним. Ім'я має збігатися з тим, що задано в Zabbix.

2. **\*\*Відсутність хоста в Zabbix\*\***: Перевірте, чи є цей хост зареєстрованим у вашій Zabbix-системі. Для цього перейдіть до розділу "Configuration" -> "Hosts" в інтерфейсі Zabbix і подивіться, чи він там є. Якщо ні, вам потрібно його додати.

3. **\*\*Проблеми з мережевим підключенням\*\***: Якщо хост знаходиться на іншому сервері, переконайтеся, що мережеве підключення між Zabbix-сервером і цим хостом працює. Спробуйте використати команди типу `ping`, щоб перевірити доступність хоста.

4. **\*\*Проблеми з DNS\*\***: Якщо ви використовуєте доменні імена, перевірте, чи правильно налаштований DNS, і чи сервер може їх розпізнати.

5. **\*\*Конфігурація агента\*\***: Якщо ви працюєте з Zabbix-agent, переконайтеся, що його конфігураційний файл (`zabbix\_agentd.conf`) налаштований правильно, включаючи параметри `Hostname` та `Server`.

Щоб вирішити цю помилку, виконайте наведені вище кроки, і це має допомогти вам знайти причину недоступності хоста у Zabbix.]

**[ПОМИЛКА: CR\_UNKNOWN\_HOST]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка `ER\_SERVER\_SHUTDOWN` в Zabbix зазвичай пов'язана з тим, що сервер бази даних MySQL (або MariaDB) був аварійно зупинений або перезапущений під час виконання запиту. Ця помилка може виникнути з кількох причин:

1. **\*\*Перезапуск сервера\*\***: Якщо сервер бази даних перезапустився через оновлення або планове технічне обслуговування, ви можете отримати цю помилку в момент виконання запиту Zabbix.

2. **\*\*Нестабільність сервера бази даних\*\***: Якщо база даних з якоїсь причини стала недоступною (наприклад, через важку навантаження або програмні помилки), це може викликати цю помилку.

3. **\*\*Апаратні збої\*\***: Проблеми з апаратним забезпеченням, такими як збої системи або відключення живлення, можуть призвести до аварійного зупинення сервера бази даних.

4. **\*\*Перевищення ресурсів\*\***: Якщо сервер бази даних перевищує доступні ресурси (оперативну пам'ять, дисковий простір тощо), він може самостійно зупинити свою роботу.

### Як виправити цю помилку:

1. **\*\*Перевірка статусу бази даних\*\***: Зайдіть на сервер бази даних і перевірте, чи працює MySQL/MariaDB. Ви можете це зробити за допомогою команди, наприклад, `systemctl status mysql`` або `systemctl status mariadb``.

2. **\*\*Перегляд журналів\*\***: Ознайомтеся з журналами бази даних (зазвичай розміщуються в `/var/log/mysql/`` або `/var/log/mariadb/``), щоб знайти повідомлення про помилки або причини зупинки сервера.

3. **\*\*Перевірка ресурсів\*\***: Переконайтеся, що сервер має достатньо ресурсів (оперативна пам'ять, простір на диску).

4. **\*\*Запуск сервера\*\***: Якщо сервер був зупинений, спробуйте запуснути його за допомогою команди `systemctl start mysql`` або `systemctl start mariadb``.

5. **\*\*Налаштування Zabbix\*\***: Після відновлення роботи сервера бази даних перевірте конфігурацію Zabbix, щоб упевнитися, що базу даних правильно налаштовано і доступно.

Якщо ви все ще стикаєтеся з проблемами після виконання цих кроків, може бути корисно звернутися до документації або отримати допомогу від адміністратора бази даних.]

## **[ПОМИЛКА: ER\_ILLEGAL\_GRANT\_FOR\_TABLE]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка `ER\_ILLEGAL\_GRANT\_FOR\_TABLE` виникає в середовищі бази даних MySQL, коли намагаються задати привілеї (grant) для таблиці в недозволеній спосіб. Це означає, що ви намагаєтесь надати права доступу до таблиці, але не маєте для цього необхідних прав або вказуєте неправильний контекст.

Давайте розглянемо, що могло піти не так:

1. **\*\*Неправильний контекст\*\***: Ви можете намагатися надати привілеї не на правильній таблиці або не в тій базі даних, де у вас є відповідні доступи. Переконайтеся, що ви вказали правильну базу даних і таблицю.

2. **\*\*Відсутність необхідних прав\*\***: Для того, щоб видавати права іншому користувачу, ви повинні мати достатні привілеї самі. Переконайтеся, що ваш обліковий запис має права `GRANT OPTION`.

3. **\*\*Синтаксична помилка\*\***: Переконайтеся, що команда SQL для надання прав написана коректно. Наприклад, перевірте, чи ви використовуєте правильний синтаксис для команди `GRANT`.

4. **\*\*Об'єкт недоступний\*\***: Можливо, таблиця, для якої ви намагаєтесь видати права, не існує або ви не маєте прав доступу до неї.

Щоб виправити цю помилку, спробуйте:

- Перевірити права на вашому користувачеві.
- Переконатися, що ви вказали правильний об'єкт (базу даних і таблицю).
- Виконати команду в контексті, де у вас є достатні права.

Якщо вам необхідна додаткова допомога в діагностиці або написанні команди, будь ласка, надайте більше деталей про вашу ситуацію!]

## **[ПОМИЛКА: ER\_TABLEACCESS\_DENIED\_ERROR]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка ``ER_TABLEACCESS_DENIED_ERROR`` у Zabbix вказує на те, що у вас немає достатніх прав доступу до певної таблиці в базі даних MySQL або MariaDB. Ця помилка зазвичай виникає, коли користувач, під яким Zabbix намагається підключитися до бази даних, не має прав на виконання певних дій (наприклад, читання, запис або зміна) для вказаної таблиці.

### Що робити?

1. **\*\*Перевірте права користувача\*\***:

- З'ясуйте, під яким користувачем Zabbix підключається до бази даних.

Це можна знайти у файлі конфігурації Zabbix (зазвичай ``zabbix_server.conf`` або ``zabbix_agentd.conf``).

- Підключіться до бази даних MySQL/MariaDB під цим користувачем і перевірте його права, наприклад, за допомогою команди:

```
``sql
SHOW GRANTS FOR 'имя_пользователя'@'хост';
``
```

2. **\*\*Надайте необхідні права\*\***:

- Якщо ви бачите, що у користувача немає прав доступу, вам потрібно або надати йому ці права, або створити нового користувача з необхідними правами. Наприклад, щоб надати всі права на базу даних Zabbix, ви можете виконати:

```
``sql
GRANT ALL PRIVILEGES ON zabbix.* TO 'имя_пользователя'@'хост';
FLUSH PRIVILEGES;
``
```

- Не забудьте замінити ``zabbix`` на ім'я вашої бази даних, а також ``имя_пользователя`` на ім'я вашого користувача.

3. **\*\*Перевірте налаштування Zabbix\*\***:

- Переконайтесь, що дані підключення в конфігурації Zabbix (ліцензія, ім'я бази даних, ім'я користувача та пароль) правильні.

#### 4. **\*\*Перезавантажте Zabbix\*\***:

- Після зміни прав доступу, якщо це потрібно, перезапустіть службу Zabbix, щоб впевнитись, що зміни набрали чинності.

#### ### Висновок

Помилка ``ER_TABLEACCESS_DENIED_ERROR`` виникає через недостатні права доступу. Щоб усунути цю помилку, вам потрібно перевірити та, за потреби, надати правильні права для користувача, під яким Zabbix підключається до бази даних.]

### **[ПОМИЛКА: CR\_CONNECTION\_ERROR]**

### **[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка **\*\*CR\_CONNECTION\_ERROR\*\*** у Zabbix зазвичай вказує на проблему з'єднання між вашим Zabbix сервером або агентом і Zabbix сервером або базою даних. Ось кілька можливих причин та рішень цієї помилки:

#### 1. **\*\*Неправильні конфігурації\*\***:

- Перевірте файл конфігурації Zabbix (зазвичай ``zabbix_server.conf`` для сервера або ``zabbix_agentd.conf`` для агента). Переконайтеся, що параметри, такі як ``Server``, ``Hostname``, ``ListenIP``, вказані правильно і відповідно до вашої мережі.

#### 2. **\*\*Проблеми з мережею\*\***:

- Якщо Zabbix сервер і агент знаходяться на різних машинах, перевірте, чи є між ними мережеве з'єднання. Можна скористатися командою ``ping`` або ``telnet``, щоб перевірити, чи доступний порт, на якому працює Zabbix.

#### 3. **\*\*Проблеми з брандмауером\*\***:

- Переконайтеся, що брандмауер на сервері не блокує порти, які використовуються для зв'язку Zabbix (за замовчуванням це 10051 для сервера та 10050 для агента).

#### 4. **\*\*Неправильна версія Zabbix\*\***:



- Якщо ви оновлювали Zabbix або використовуєте різні версії на сервері та агенті, це також може спричинити проблеми з сумісністю. Переконайтеся, що версії однакові або сумісні.

5. **\*\*Вимкнені служби\*\***:

- Перевірте, чи запущені служби Zabbix. Ви можете це зробити за допомогою команд `systemctl status zabbix-server` для сервера або `systemctl status zabbix-agent` для агента.

6. **\*\*Логи\*\***:

- Перегляньте логи Zabbix (вони можуть знаходитись за адресами `/var/log/zabbix/zabbix\_server.log` або `/var/log/zabbix/zabbix\_agentd.log`), щоб отримати більш детальну інформацію про причину помилки.

Виправивши зазначені проблеми, ви зможете усунути **\*\*CR\_CONNECTION\_ERROR\*\***.]

**[ПОМИЛКА: ER\_ACCESS\_DENIED\_ERROR]**

**[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка `ER\_ACCESS\_DENIED\_ERROR` в Zabbix (і в базі даних MySQL) вказує на те, що є проблеми з доступом до бази даних. Це означає, що Zabbix не може підключитися до бази даних через неправильні облікові дані або відсутність прав доступу. Ось кілька поширених причин та їх рішення:

1. **\*\*Неправильне ім'я користувача або пароль\*\***:

- Перевірте, чи вірно ви ввели ім'я користувача і пароль для доступу до бази даних MySQL у конфігураційних файлах Zabbix (зазвичай це `zabbix\_server.conf` або `zabbix\_web.conf`).

2. **\*\*Користувач не має прав доступу\*\***:

- Переконайтеся, що користувач, під яким ви намагаєтесь підключитися до бази даних, має достатні права доступу (SELECT, INSERT, UPDATE тощо) для потрібної бази даних. Для цього ви можете виконати команду SQL:

```
```sql
SHOW GRANTS FOR 'username'@'host';
```
```

3. **\*\*Користувач не має доступу з конкретного хоста\*\***:

- Перевірте, чи користувач має доступ до бази даних з того хоста, з якого ви намагаєтеся підключитися. Якщо користувач створений для доступу тільки з локального хоста (`localhost`), а ви підключаєтеся з іншого, то виникне ця помилка.

4. **\*\*Блокування з боку сервера\*\***:

- Іноді сервер MySQL може блокувати підключення. Переконайтеся, що сервер запущений і доступний для вхідних з'єднань. Також перевірте налаштування брандмауера та інших мережевих конфігурацій.

5. **\*\*Проблеми з конфігурацією Zabbix\*\***:

- Перевірте конфігураційний файл Zabbix на наявність помилок. Можливо, там є помилка в синтаксисі або форматуванні.

## Дії для вирішення:

1. **\*\*Перевірте облікові дані\*\***:

- Знайдіть налаштування підключення до бази даних у файлі конфігурації Zabbix та переконайтеся, що там вказано правильне ім'я користувача, пароль, схему бази даних та хост.

2. **\*\*Надайте необхідні права\*\***:

- Якщо потрібно, надайте користувачу права доступу через команду:

```
```sql
GRANT ALL PRIVILEGES ON zabbix.* TO 'username'@'host';
FLUSH PRIVILEGES;
```
```

3. **\*\*Перезапустіть служби\*\***:

- Після внесення змін до конфігурації або прав, можна перезапустити служби Zabbix, а також сервер бази даних для того, щоб зміни набрали чинності.

#### 4. **\*\*Перевірте лог-файли\*\***:

- Перегляньте лог-файли Zabbix (наприклад, `zabbix\_server.log` або `zabbix\_web.log`) та MySQL для отримання детальнішої інформації про помилку.

Після виконання цих дій ви повинні зуміти усунути помилку `ER\_ACCESS\_DENIED\_ERROR` та налаштувати Zabbix на коректну роботу з базою даних.]

### **[ПОМИЛКА: CR\_CONN\_HOST\_ERROR]**

### **[ВІДПОВІДЬ ВІД НЕЙПРОМЕРЕЖІ: Привіт від Zabbix!]**

Помилка `ER\_UNKNOWN\_ERROR` в Zabbix зазвичай вказує на те, що сталася невідома помилка в системі. Це може бути пов'язано з різними факторами, такими як проблеми з конфігурацією, помилки в базі даних або проблеми з доступом до ресурсів.

Ось кілька кроків, які можуть допомогти в усуненні цієї помилки:

1. **\*\*Перевірка журналів\*\***: Перш ніж робити щось ще, перевірте журнали Zabbix. Вони можуть надати додаткову інформацію про причину помилки. Журнали зазвичай знаходяться в `/var/log/zabbix/` на вашому сервері.

2. **\*\*Перезапуск сервісів\*\***: Іноді простий перезапуск сервісів Zabbix (наприклад, Zabbix server та Zabbix agent) може вирішити проблему. Спробуйте виконати команду:

```
``bash
sudo systemctl restart zabbix-server zabbix-agent
``
```

3. **\*\*Перевірка конфігурації\*\***: Переконайтеся, що файл конфігурації Zabbix налаштований правильно. Зверніть увагу на такі параметри, як підключення до бази даних та налаштування сервера.

4. **\*\*Наявність оновлень\*\***: Переконайтеся, що ви використовуєте останню стабільну версію Zabbix. Іноді зайві помилки можуть бути виправлені в нових релізах.

5. **\*\*Проблеми з базою даних\*\***: Якщо ви підозрюєте, що проблема пов'язана з базою даних, перевірте її цілісність. Можливо, вам знадобиться виконати команду для перевірки та ремонту бази даних.

6. **\*\*Додаткові ресурси\*\***: Можливо, вам потрібно перевірити, чи достатньо системних ресурсів (RAM, CPU, дискового простору). У разі нестачі ресурсів можуть виникати проблеми.

Якщо проблема не вирішується після цих кроків, вам, можливо, потрібно звернутися до спільноти Zabbix або шукати специфічні рішення на форумах, або навіть розглянути можливість створення запиту на підтримку.]

### **Висновок до розділу №3**

У третьому розділі було детально описано процес створення та налаштування автоматизованої системи для оповіщення про події кібербезпеки з використанням Zabbix, нейромережі OpenAI та Telegram. Впроваджені етапи — введення нейромережі в контекст задачі, розробка коду web-hook для інтеграції систем, а також об'єднання всіх компонентів у єдиний механізм — дозволили сформувати комплексну і водночас гнучку систему для моніторингу та автоматичного реагування на події.

Вибір архітектури трансформера в нейромережі OpenAI був зумовлений її здатністю глибоко аналізувати текстові дані та розпізнавати складні контексти, що критично важливо для розшифровки повідомлень про помилки та генерації інструкцій щодо їх усунення. Інтеграція OpenAI з Zabbix через сервер ngrok забезпечила надійний і швидкий обмін даними між системами, а використання Telegram як каналу сповіщення підвищило оперативність і доступність інформації для персоналу, що дозволяє миттєво реагувати на події кібербезпеки.

Результати налаштування показали, що розроблена система здатна ефективно інтегрувати моніторинг, аналіз та оповіщення в єдиний автоматизований процес. Це дозволяє значно зменшити навантаження на працівників та покращити швидкість і точність реагування на інциденти. Система демонструє високий потенціал для подальшого розширення та масштабування, забезпечуючи надійний інструмент для управління кіберризиками в умовах зростання обсягу даних та складності кіберзагроз.

## ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено систему для автоматизованого моніторингу та оповіщення про події кібербезпеки, яка об'єднує можливості інтелектуального аналізу з використанням штучного інтелекту та інструменти системи моніторингу Zabbix. Основна мета полягала в тому, щоб забезпечити надійний механізм виявлення інцидентів у режимі реального часу та автоматичну генерацію інструкцій для усунення помилок, доступних для користувачів через месенджер Telegram.

У першому розділі було проведено аналіз існуючих підходів до використання штучного інтелекту в кібербезпеці, зокрема на основі міжнародного стандарту ISO/IEC 27005:2022 [3]. Дослідження виявило, що інтелектуальні системи мають значний потенціал для автоматизації моніторингу подій кібербезпеки та мінімізації людського втручання, що підвищує оперативність реагування [4, 5]. Було детально розглянуто сучасні підходи до інтеграції штучного інтелекту в системи моніторингу безпеки, де основну увагу було приділено можливостям трансформерних моделей, здатних аналізувати великі обсяги текстових даних у реальному часі [6, 8].

Другий розділ присвячений аналізу різних архітектур нейронних мереж для завдань кібербезпеки, включаючи лінійні, нелінійні, нечіткі, деревоподібні та байєсові моделі [12, 14, 25]. Було обґрунтовано вибір трансформерної архітектури від OpenAI для даного проєкту через її здатність розпізнавати складні закономірності та працювати з великими обсягами даних [11, 13]. Аналіз різних моделей і методів показав, що трансформери можуть забезпечити високу якість та точність виявлення інцидентів кібербезпеки, а також генерувати детальні інструкції щодо їх усунення [17, 18]. Розглянуто також методи інтеграції нейромереж через API, що дозволяє застосовувати трансформерні моделі для автоматизованого аналізу даних у реальному часі [16, 21].

У третьому розділі описано реалізацію та налаштування системи, включаючи основні етапи експерименту: введення нейромережі OpenAI у контекст задачі, розробку коду web-hook на Python для з'єднання систем Zabbix і ШІ, а також об'єднання всіх компонентів в єдину систему. Впровадження web-hook забезпечило швидкий і стабільний обмін даними між Zabbix та OpenAI, дозволяючи автоматично передавати дані про помилки до ШІ та отримувати рекомендації для їх усунення [22, 24]. Інтеграція Telegram API для сповіщення відповідального персоналу забезпечила додаткову оперативність в інформуванні про інциденти та дозволила швидко реагувати на них, що є важливим у контексті сучасних кіберзагроз [20, 28].

Загалом, розроблена система демонструє ефективність поєднання інструментів штучного інтелекту та автоматизації моніторингу для виявлення інцидентів кібербезпеки. Система забезпечує своєчасне реагування, детальні інструкції з усунення помилок та гнучкість у налаштуванні. З'єднання нейромережі OpenAI, системи Zabbix і Telegram дозволило створити інноваційний механізм автоматизованого моніторингу, який є перспективним для подальшого впровадження у сферу кібербезпеки, де точність та оперативність є ключовими аспектами [23, 26].

## СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тарік Рашид. Make Your Own Neural Network. Pearson Independently Published. 1-е видання. 222 с. С. 134-137. [2.1, 2.2]
2. Джон Пол Мюллер. Artificial Intelligence For Dummies. Для чайників. 1-е видання. 336 с. С. 200-205. [2.2]
3. ISO/IEC 27005:2022. Information technology — Security techniques — Information security risk management. International Organization for Standardization, 2022.  
Доступно за посиланням: <https://www.iso.org/standard/75281.html> (дата звернення: 05.11.2023). С. 50-55, 100-105, 200-210. [1.2, 1.4, 3.2]
4. О Теобальд. Machine Learning For Absolute Beginners. Scatterplot Press. 2-е видання. 164 с. С. 88-92. [2.1]
5. Нік Бостром. Superintelligence. Audible Studios. Нескорочене видання. 431 с. С. 210-215. [1.1, 3.1]
6. Стюарт Рассел. Artificial Intelligence: A Modern Approach. Pearson. 3rd видання. 1152 с. С. 580-585. [1.2, 2.3, 3.2]
7. Джеймс V Стоун. Artificial Intelligence Engines: A Tutorial Introduction to the Mathematics of Deep Learning. Sebtel Press. 1-е видання. 218 с. С. 110-115. [2.3]
8. Макс Тегмарк. Life 3.0: Being Human in the Age of Artificial Intelligence. Кнопка. 1-е видання. 384 с. С. 300-305. [1.2]
9. Джон Крон. Deep Learning Illustrated. Addison-Wesley Professional. 1-е видання. 416 с. С. 120-125. [2.3]
10. Анассе Барі. Predictive Analytics For Dummies. Для чайників. 2-е видання. 435 с. С. 250-255. [2.2]
11. Джоел Грус. Data Science from Scratch: First Principles with Python. О'Райлі. 2-е видання. 500 с. С. 300-305. [3.1]



12. Орельєн Жерон. Hands-On Machine Learning. Шрофф/О'Рейлі. 2-е видання. 848 с. С. 500-505. [2.4]
13. Марія Яо, Аделін Чжоу, Марлен Цзя. Прикладний штучний інтелект. ТОПБОТИ. 1-е видання. 246 с. С. 134-138. [2.1]
14. Prediction Machines: The Simple Economics of Artificial Intelligence. Аджаєм Агравалом, Джошуа Гансом і Аві Голдфарбом. Гарвардський бізнес Review Press. 1-е видання. 250 с. С. 150-155. [1.3]
15. Human + Machine: Reimagining Work in the Age of AI. Полом Р. Догерті та Г. Джеймсом Вілсоном. Гарвардський бізнес Review Press. 1-е видання. 246 с. С. 200-205. [1.3]
16. Architects of Intelligence: The truth about AI from the people building it. Мартін Форд. Видавництво Packt. 1-е видання. 554 с. С. 400-405. [2.3]
17. Artificial Intelligence for Humans: Fundamental Algorithms. Джефф Гітон. Незалежно опубліковано. 1-е видання. 224 с. С. 130-135. [2.2]
18. HBR's 10 Must Reads on AI, Analytics, and the New Machine Age. Гарвардський бізнес Review. Незалежно опубліковано. 1-е видання. 161 с. С. 50-55. [1.3]
19. Zabbix. Офіційний сайт Zabbix.  
URL: <https://www.zabbix.com/> (дата звернення: 05.11.2023). [3.2]
20. OpenAI. Платформа OpenAI API.  
URL: <https://platform.openai.com/> (дата звернення: 01.11.2023). [3.2]
21. Telegram. API Telegram для автоматизації повідомлень.  
URL: <https://core.telegram.org/bots/api> (дата звернення: 10.11.2023). [3.2]
22. Ngrok. Сервіс для створення захищених тунелів між локальними серверами та зовнішніми сервісами.  
URL: <https://ngrok.com/> (дата звернення: 10.11.2023). [3.2]
23. TensorFlow in 1 Day: Make Your Own Neural Network. Krishna Рунгта. Guru99. 1-е видання. 446 с. С. 200-205. [2.1]

24. Deep Learning (Adaptive Computation and Machine Learning series). Ян Гудфеллоу. Преса МІТ. 1-е видання. 800 с. С. 300-305. [2.3]
25. Python Machine Learning. Себастьян Рашка. Ingram. 1-е видання. 454 с. С. 150-155. [3.2]
26. Deep Learning with R. Франсуа Шолле. Меннінг. 1-е видання. 360 с. С. 100-105. [3.1]
27. Стратегія розвитку штучного інтелекту в Україні. А. І. Шевченко, С. В. Барановський, О. В. Білокобильський, Є. В. Бодянський. 2023. 305 с.  
URL: <https://jai.in.ua> (дата звернення: 12.11.2023). С. 180-185. [1.4]
28. Адріан Барбу, Сонг-Чун Чжу. Monte Carlo Methods. Springer. 2020. 422 с. С. 150-155. [2.4, 3.2]

## ДОДАТКИ

### Додаток А

#### Написання коду для AI web-hook на мові програмування Python

```
python
import requests
import json

def ai_webhook(value):
    try:
        url = 'http://richboyscrytoo.tech/webhook'
        params = json.loads(value)
        context = ('Контекст: Ти експерт в роботі із Zabbix зі стажем більше 10
років. '
                 'Твоя задача максимально доступно пояснювати помилки, які я
передаватиму тобі.')
        message = context + ' ' + 'Помилка: ' + params['message']

        headers = {'Content-Type': 'application/json'}
        proxies = {'http': params['HTTPProxy']} if 'HTTPProxy' in params else None

        print(f'[Ai webhook] Webhook request with value={value}')

        response = requests.post(url, json={'message': message}, headers=headers,
proxies=proxies)
        print(f'[Ai webhook] Responded with code: {response.status_code}. Response:
{response.text}')
```

```

if response.status_code != 200:
    raise Exception(f'Request failed with status code {response.status_code}')

try:
    response_data = response.json()
except json.JSONDecodeError:
    raise Exception('Request succeeded, but response parsing failed.')

if 'response' in response_data:
    # Send message to Telegram
    telegram_url =
f'https://api.telegram.org/bot6534778641:AAG1JJGDbZHU-SneJen_Xeug_TxTjI_qI
v8/sendMessage'
    telegram_message = 'Привіт від Zabbix та ChatGPT!\n\n' +
response_data['response']
    telegram_data = {
        'chat_id': '-4542496214',
        'text': telegram_message,
        'parse_mode': 'HTML'
    }
    telegram_response = requests.post(telegram_url, json=telegram_data,
headers=headers)
    print('[Ai webhook] Telegram message sent.')

return response_data['response']

except Exception as error:
    print(f'[Ai webhook] Sending failed. Error: {error}')
    raise Exception(f'Failed with error: {error}')

```

Пояснення:

Підключення бібліотек: Використовується `requests` для HTTP-запитів та `json` для обробки JSON-даних.

Проксі-сервер: Перевіряється, чи є `HTTPProxy` в параметрах, і якщо так - передається як параметр `proxies`.

Логування: Виводяться повідомлення `print` для логування процесу.

Telegram-запит: Надсилається повідомлення через API Telegram, якщо в `response_data` є `response`.

Код для `web-hook` було реалізовано на Python з використанням бібліотеки `requests` для відправки HTTP-запитів до API OpenAI та API Telegram.