МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Олексій Горський

"____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

МАГІСТРА

Тема: "Методика вирішення правових конфліктів при повторному використанні програмних артефактів, які згенеровані засобами штучного інтелекту"

Виконавець: Архипченко Ірина Юріївна

Керівник: к.т.н. Ходаков Данііл Вікторович

Нормоконтролер: Трофимчук Вікторія Миколаївна

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма Інженерія програмного забезпечення
Форма навчання заочна

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олексій Горський
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студентки
Архипченко Ірини Юріївни

1. Тема кваліфікаційної роботи: "Методика вирішення правових конфліктів при повторному використанні програмних артефактів, які згенеровані засобами штучного інтелекту" затверджена наказом ректора від 04.10.2023 р. № 2034/ст.

2. Термін виконання проекту: з 04.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи: розробити методику вирішення правових конфліктів, при використанні програмних артефактів, згенерованих інструментами штучного інтелекту

4. Зміст пояснювальної записки:
   1. Аналіз національних та міжнародних нормативних актів, що регулюють право інтелектуальної власності, авторське право, патентне право для оцінки можливостей правового захисту використання програмних артефактів, згенерованих штучним інтелектом.
   2. Запропонована методика вирішення правових конфліктів, при використанні програмних артефактів, згенерованих інструментами штучного інтелекту.
   3. Розробка комплаєнс веб-платформи для тестування запропонованої методики.
   4. Застосування розробленої методики на практиці.

5. Перелік обов'язкових слайдів презентації:

1.Тема, об’єкт дослідження, предмет дослідження, методи дослідження, виконавець, керівник.

2. Правові проблеми використання артефактів, згенерованих штучним інтелектом.

3. Опис запропонованої методики.

4. Архітектура розробленої комплаєнс веб-платформи.

5. Демонстрація розробленої комплаєнс веб-платформи.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Ознайомлення з постановкою задачі та пошук літературних джерел. Складання графіку роботи.	04.10.23	
2.	Підготовка та написання 1 розділу. Представлення керівнику	04.10.23 – 08.10.23	
3.	Підготовка та написання 2 розділу. Представлення керівнику керівнику	08.10.23 – 18.10.23	
4.	Підготовка та написання 3 розділу. Представлення керівнику керівнику	19.10.23 – 03.11.23	
5.	Підготовка та написання 4 розділу. Представлення керівнику керівнику	04.11.23 – 13.11.23	
	Підготовка та написання 5 розділу та висновків. Представлення керівнику керівнику	13.11.23 – 29.11.23	
6.	Редагування та друк пояснювальної записки, графічного матеріалу Відправлення ПЗ для перевірки на плагіат одним файлом.	05.12.23 – 15.12.023	
7.	Проходження нормо-контролю, перепліт пояснювальної записки. Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	12.12.2023 – 16.12.2023	
8.	Попередній захист дипломної роботи	20.12.2023	
9.	Отримання рецензії.	21.12.2023	
10.	Здати секретарю ДЕК: ПЗ, ГМ, CD-R з електронними версіями ПЗ, ГМ, презентацію, відгук керівника, рецензію, довідку про успішність, 2 папки, 2 конверта.	22.12.2023	
11.	Захист дипломної роботи перед ЕК	25.12.2023	

Дата видачі завдання: 04.10.2023 р.

Керівник дипломної роботи: _____ к.т.н. Данііл ХОДАКОВ

Завдання прийняла до виконання: _____ Ірина АРХИПЧЕНКО

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Методика вирішення правових конфліктів при повторному використанні програмних артефактів, які згенеровані засобами штучного інтелекту»: 121 сторінка, 1 таблиця, 126 використаних джерел, 4 додатки.

Об'єкт дослідження – програмні артефакти, створені за допомогою інструментів штучного інтелекту, та правові правові норми, що регулюють їх використання.

Мета дипломної роботи – розробка надійної та ефективної методології запобігання та вирішення правових конфліктів при використанні програмних артефактів, створених штучним інтелектом.

Методи дослідження – метод аналізу, метод синтезу, структурний і функціональний методи.

Галузь застосування та ступінь впровадження матеріалів дипломної роботи: результати магістерської роботи можуть бути використання при розробці пропрієтарного та open-source програмного забезпечення, коли при розробці використовуються генератори програмних артефактів на основі штучного інтелекту.

ШТУЧНИЙ ІНТЕЛЕКТ, ПРАВОВІ ПРОБЛЕМИ, ПРОГРАМНІ АРТЕФАКТИ, ГЕНЕРАТОР КОДУ, АВТОРСЬКЕ ПРАВО, ЛІЦЕНЗУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ З ВІДКРИТИМ ВИХІДНИМ КОДОМ, ПРОПРІЄТАРНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ABSTRACT

Explanatory note to the thesis “Methodology for resolving legal conflicts in the reuse of software artifacts generated by artificial intelligence”: 121 pages, 1 table, 126 references, 4 appendices.

The object of the research is AI generated software artifacts and legal regulations governing their use.

The purpose of the thesis is to develop a reliable and effective methodology for preventing and resolving legal conflicts during usage and distribution of software artifacts generated by AI tools.

Research methods include analysis, synthesis, structural and functional methods.

The field of application and degree of implementation of thesis materials: the results of the master’s thesis can be used in the development of proprietary and open source software, when AI code generators are used during software development.

ARTIFICIAL INTELLIGENCE, LEGAL ISSUES, SOFTWARE ARTIFACTS, CODE GENERATOR, COPYRIGHT, LICENSING, OPEN SOURCE SOFTWARE, PROPRIETARY SOFTWARE.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ
ВСТУП
РОЗДІЛ 1. ШТУЧНИЙ ІНТЕЛЕКТ В СУЧАСНІЙ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
1.1. Історична еволюція штучного інтелекту і його вплив на процес розробки програмного забезпечення
1.2. Огляд програмних артефактів, створених штучним інтелектом, та їхнього потенційного впливу на індустрію
1.3. Переваги та ризики використання програмних артефактів, згенерованих штучним інтелектом
РОЗДІЛ 2. ПРОБЛЕМИ ПРАВА ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ НА ПРОГРАМНІ АРТЕФАКТИ ЗГЕНЕРОВАНІ ШТУЧНИМ ІНТЕЛЕКТОМ
2.1. Проблеми авторського права
2.2. Патентоспроможність
РОЗДІЛ 3. ПРОГРАМНИЙ КОД, СТВОРЕНИЙ ШТУЧНИМ ІНТЕЛЕКТОМ: ПРОБЛЕМИ З ДЖЕРЕЛАМИ ТА ЛІЦЕНЗУВАННЯМ
3.1. Ліцензування в контексті пропрієтарного та open source програмного забезпечення
3.2. Проблема несумісності ліцензій та методи вирішення
РОЗДІЛ 4. МЕТОДОЛОГІЯ ДОТРИМАННЯ ПРАВОВИХ ВИМОГ ТА ВИРІШЕННЯ ПРАВОВИХ КОНФЛІКТІВ ПРИ ВИКОРИСТАННІ АРТЕФАКТІВ СТВОРЕНИХ ШТУЧНИМ ІНТЕЛЕКТОМ
4.1. Огляд та загальна структура методології
4.2. Основний зміст методології
4.2.1. Правові дослідження та аналіз чинного законодавства
4.2.2. Оцінка ризиків
4.2.3. Ведення документація та відстеження
4.2.4. Виявлення потенційних правових проблем та система комплаєнсу
4.2.5. Розробка механізму вирішення спорів
4.2.6. Забезпечення навчання та освіти
4.2.7. Впровадження системи моніторингу та стратегії вдосконалення
4.2.8. Етапи впровадження
РОЗДІЛ 5. ПРОЕКТУВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ КОМПЛАЄНС ПЛАТФОРМИ
5.1. Мета розробки платформи
5.2. Загальний огляд платформи
5.3. Вибір інструментів і технологій для проектування та розробки платформи
5.3.1. Планування, дизайн архітектури та UI/UX дизайн
5.3.2. Інструменти розробки
5.3.3. Front-end стек
5.3.4. Back-End стек

5.3.5. Інші сервіси та API
5.3.6. Розгортання та хостинг
5.4. Архітектурні рішення та імплементація
5.4.1. Архітектура розгортання
5.4.2. Взаємодія компонентів системи
5.4.3. Структура бази даних та взаємодія її сутностей
5.4.4. Потік створення рекомендацій для користувача
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК 1. Діаграма 1: Deployment Diagram
ДОДАТОК 2. Діаграма 2: Components Diagram
ДОДАТОК 3. Діаграма 3: Entity Relationship Diagram
ДОДАТОК 4. Діаграма 4: Guide Creation Sequence Diagram

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ШІ – штучний інтелект

ПП - програмний продукт

ПА - програмний артефакт

ПЗ - програмне забезпечення

ПС - програмна система

КПВ - компоненти повторного використання

ІВ - інтелектуальна власність

ПІВ - право інтелектуальної власності

ВСТУП

В епоху цифрової трансформації штучний інтелект глибоко проник у різні галузі. Розвиток ШІ також вплинув на сферу інженерії програмного забезпечення: інженерам доступні різноманітні інструменти, які пришвидшують та оптимізують процеси розробки програмних продуктів. Програмні артефакти, згенеровані ШІ інтегруються і повторно використовуються програмістами. Оскільки ШІ продовжує розвиватися, виникає питання: чи безпечно використовувати програмні артефакти, згенеровані ШІ з правової точки зору?

Проблема дослідження. Невизначеність правового статусу програмних артефактів, створених ШІ, і їх повторне використання створює складні юридичні проблеми пов'язані з правами інтелектуальної власності, авторськими та суміжними правами, упередженістю моделей ШІ та ліцензуванням програмного забезпечення. Існує гостра потреба у вивченні та створенні чітких методологій для вирішення цих проблем, забезпечення правової ясності та сприяння відповідальним інноваціям.

Предмет і об'єкт дослідження. *Предметом* дослідження є юридичні колізії та правові проблеми, що виникають у зв'язку з повторним використанням програмних артефактів, створених за допомогою ШІ. *Об'єктом* дослідження є самі програмні артефакти, створені за допомогою ШІ, інструменти-генератори на основі ШІ, та різні правові норми, що регулюють використання і розповсюдження, створених програмних артефактів.

Актуальність теми дослідження. У сучасному технологічному ландшафті стрімкий розвиток штучного інтелекту не лише сигналізує про глибокі зміни в тому, як ми розробляємо та впроваджуємо програмні рішення, але й підкреслює нагальну потребу в надійному правовому регулюванні. Штучний інтелект, за своєю природою, створює унікальні

виклики. Його здатність автономно генерувати нові програмні артефакти ставить складні питання щодо традиційних правових концепцій, таких як власність, авторське право та відповідальність. Вирішення цих питань є не просто предметом академічної інтриги, воно має нагальні наслідки для зацікавлених сторін галузі, юристів-практиків та суспільства в цілому.

Існуючий розрив між технологічними можливостями та правовою визначеністю перешкоджає інноваціям і породжує етичні та правові дилеми. Без чітко визначеної методології для вирішення цих проблем існує ризик гальмування інновацій, а творці та користувачі систем штучного інтелекту опиняться в пастці правової невизначеності. Більше того, оскільки різні галузі економіки дедалі ширше застосовують ШІ, масштаби і частота цих правових конфліктів зростатимуть, що робить вирішення цих питань нагальним завданням.

Дослідження прагне створити модель, якою можуть керуватися користувачі систем ШІ, гарантуючи, що продукти ШІ будуть використовуватися відповідально і з належною увагою до всіх залучених зацікавлених сторін.

Наукова новизна та огляд досліджень за обраною темою. Дослідження юридичних проблем, які виникають при використанні програмних артефактів, створених штучним інтелектом на сьогдні залишаються фрагментованими та недостатніми, оскільки успіх та популярність генеративного ШІ розвинулись порівняно нещодавно. Крім того, на сьогодні немає чітких законодавчих підходів та стратегій щодо регулювання проблем використання продуктів, створених системами ШІ.

Проблеми ***авторського права*** на роботи, створені генеративним ШІ піднімав ***Хан Вонґ*** у своєму дослідженні. Автор розглядає зростаючу роль ШІ у створенні “оригінальних” літературних, художніх і музичних творів. У статті він порушує важливі питання про застосування авторського права до творів, створених штучним інтелектом, і стверджує, що такі твори

повинні бути захищені авторським правом, оскільки вони можуть відповідати мінімальному ступеню оригінальності. Однак у статті стверджується, що ШІ не може розглядатися як автор згідно з цивільним правом і теорією автора-фізичної особи. Натомість авторство має належати користувачам ШІ [120].

Ерана Бопче у своїй роботі досліджує складнощі авторського права, що стосуються творів, створених ШІ. Авторка досліджувала питання захисту авторським право творів, створених ШІ, а також наслідки передачі авторства на згенеровані твори штучному інтелекту або ж його користувачам. У статті розглядаються правові, філософські та практичні проблеми визнання ШІ автором, зокрема наслідки для теорії авторського права, авторських прав та юридичного визначення автора. Дослідниця пропонує вважати власника системи ШІ фактичною особою, відповідальною за дотримання авторських прав, відповідальною за будь-які порушення, а також вправі визначати економічні права, пов’язані з творами, створеними за допомогою ШІ [23].

У своїй роботі ***Хезер Мулесон-Сенді*** досліджує, як контент, створений ШІ, зокрема з ChatGPT, інтегрується в академічне письмо та видавничу справу. Дослідниця підкреслює складність визначення авторства і права власності на тексти, створені штучним інтелектом. Вона стверджує, що штучний інтелект не людина, а отже, він не може бути автором текстів. Крім того, у статті зазначається, що ШІ може створювати тексти, проте йому бракує інтелектуальних здібностей, що призводить до таких проблем, як неточності і спотворення контенту, який він генерує [83].

Джанкарло Фрозіо досліджує теоретичне обґрунтування інтелектуальної власності у контексті штучного інтелекту, зокрема питання про те, чи можуть твори, створені штучним інтелектом, бути

юридично захищеними в рамках існуючих режимів авторського права [104, с. 158-178].

Джеймса Гатто згадує проблеми ***ліцензування*** програмного забезпечення. У своїй статті він піднімає три ключові правові питання: чи є використання відкритого коду для навчання моделей ШІ порушенням авторських прав і чи вимагає воно дотримання умов ліцензії на відкрите програмне забезпечення; чи можуть розробники зіткнутися з претензіями щодо порушення авторських прав при використанні результатів роботи генераторів коду зі штучним інтелектом у зв'язку з обмежувальним характером ліцензій проектів, на яких тренувалась модель ШІ; критично важливе значення дотримання вимог ліцензій у проектах з відкритим кодом. Автор пропонує практичні рішення для зменшення цих ризиків [51].

У своїй статті автори ***Шрікант Джандхьяла, Джинву Кім*** та ***Арпіта Бхаттачарія*** також стурбовані наслідками використання продуктів генераторів коду на основі ШІ, моделі яких тренувались на проектах з відкритим вихідним кодом. Використання коду, згенерованого ШІ, може ненавмисно задіяти захищений авторським правом код, що потенційно може призвести до порушення авторських прав або проблем з похідними творами. Однією з нагальних проблем є ризик "забруднення" пропрієтарного коду ліцензованим кодом на основі open source ліцензій, особливо якщо модель ШІ була навчена на коді під copyleft ліцензіями, такими як GPL, які можуть накладати на похідний код зобов'язання щодо розповсюдження коду. У статті також обговорюється потенціал генераторів коду ШІ рекомендувати точні копії даних, на яких вони були треновані. Це факт накладає такі самі ліцензійні вимоги до згенерованого коду, які містяться в ліцензіях тренувальних даних [71].

Джуню Чен, Норіхіро Йошида, Хіроакі Такада розглядають складнощі ліцензування наборів даних у системах машинного навчання з

акцентом на комерційній життєздатності та законності цих наборів даних. Крім того, у їх роботі обговорюється ліцензування програмного забезпечення, розрізняючи приватні стратегії захисту авторського права, які обмежують доступ користувачів, і неприватні стратегії, які дозволяють більш відкрите використання і розповсюдження. Також згадується важливість ліцензій з відкритим вихідним кодом у розробці та розповсюдженні програмного забезпечення і те, як системи машинного навчання все частіше використовуються в комерційних цілях [24].

Деякі автори піднімають ***питання упередженості*** систем ШІ. Так, ***Аліна Ґлаубітц*** висвітлює дискримінацію меншин інструментами штучного інтелекту, яка часто є результатом упередженості їхніх навчальних даних або програмування. Серед яскравих прикладів – інструмент найму Amazon, що дискримінує жінок, рекламний алгоритм Facebook, який показує оголошення про нерухомість лише білим користувачам, а також урядові інструменти розпізнавання облич, які помилково ідентифікують чорношкірих жінок частіше, ніж білих чоловіків. Виклик для судів полягає у визначенні відповідальності за дискримінаційну шкоду, спричинену ШІ, що набуватиме дедалі більшого значення в міру того, як системи ШІ інтегруватимуться в різні індустрії. Для вирішення цієї проблеми в статті пропонується теорія алгоритмічної відповідальності в рамках деліктного права, заснована на обов'язку обережності (*duty of care*) [60].

Значення дослідження. Результати цього дослідження виходять далеко за академічні межі, формуючи практичні, правові та етичні аспекти ШІ в розробці програмного забезпечення.

Оскільки роль ШІ в суспільстві зростає, *законодавчі органи* в усьому світі стикаються з проблемою розробки правових норм, які були б одночасно справедливими та адаптивними до мінливого характеру технологічного прогресу. Це дослідження пропонує цим органам

обґрунтовану основу для створення або вдосконалення законодавства щодо артефактів, створених штучним інтелектом. Воно може слугувати орієнтиром, допомагаючи законодавцям передбачати потенційні правові конфлікти та встановлювати чіткі, дієві правила щодо ролі ШІ у створенні програмного забезпечення.

Для *компаній, які все більше покладаються на ШІ* при розробці програмного забезпечення, та інженерів, які працюють у цих організаціях, це дослідження прояснює правовий ландшафт. Розуміючи правові наслідки використання коду, створеного штучним інтелектом, вони можуть приймати обґрунтовані рішення, захищаючи свої продукти від потенційних судових спорів. Це не тільки допомагає в управлінні ризиками, але й заохочує етичне та відповідальне використання ШІ, гарантуючи, що продукти розробляються в чітких правових рамках.

У міру того, як штучний інтелект все більше вкорінюється в розробку програмного забезпечення, *юридична професія* неминуче зіткнеться зі зростаючою кількістю справ, пов’язаних зі спорами щодо коду, створеного за допомогою штучного інтелекту. Це дослідження дає юристам всебічне розуміння нюансів і тонкощів потенційних юридичних конфліктів. Озброєні цими знаннями, вони зможуть надавати більш точні юридичні консультації, ефективно представляти своїх клієнтів та сприяти розвитку юриспруденції, яка відповідає реаліям сучасної розробки програмного забезпечення.

Для окремих розробників та інженерів розуміння правового ландшафту артефактів, створених штучним інтелектом, має першорядне значення. Це дослідження розширює їхні знання, дозволяючи їм відповідально і законно використовувати результати ШІ у своїх проектах.

Значення цього дослідження, таким чином, багатогранне. Воно долає розрив між швидким технологічним прогресом і правовим ландшафтом, що розвивається повільніше, сприяючи створенню

середовища, в якому процвітають інновації, поважаються права і зводяться до мінімуму правові конфлікти.

Мета дослідження. Метою дослідження є запропонувати надійну та ефективну методологію вирішення правових конфліктів при повторному використанні програмних артефактів, створених ШІ.

Методологія дослідження. У дослідженні використані наступні методи:

1. *Структурний і функціональний методи* для всебічного розуміння природи артефактів програмного забезпечення, створеного штучним інтелектом. Цей метод допоможе деконструювати архітектуру, компоненти та функціональність артефактів. Розуміючи притаманну їм структуру та функції, стає легше зрозуміти, де можуть виникнути потенційні правові проблеми, особливо при повторному використанні цих артефактів у різних контекстах.

2. Враховуючи безліч змінних – від типу використовуваної моделі ШІ до типу створеного програмного артефакту – буде застосовано *описовий багатовимірний аналіз даних* (Descriptive Multivariate Analysis). Цей метод допоможе зрозуміти закономірності, зв'язки та потенційні залежності між багатьма змінними в наборі даних. Виявивши ці закономірності, дослідження може визначити конкретні сценарії, які можуть бути більш вразливими до правових конфліктів.

3 *Причинно-наслідковий аналіз статистичних даних* щодо програмних артефактів, створених ШІ для визначення змінних, які спричиняють зростання кількості позовів, пов'язаних з використанням таких артефактів.

4. *Порівняльно-правовий синтез* міжнародного законодавства, законодавства США, Великобританії та України щодо питань прав інтелектуальної власності, зосереджуючись на тому, як кожна з них вирішує проблеми, пов'язані з результатами роботи ШІ. Шляхом

протиставлення та порівняння дослідження прагне виявити найкращі практики та висвітлити прогалини в існуючих нормативно-правових актах.

5. *Порівняльний та прогнозний аналіз даних судової практики США та рішень Бюро авторського права США* з метою визначення майбутніх тенденцій у сфері використання артефактів, створених ШІ. Порівняно з іншими державами, США має більш розвинену судову та досудову практику з розв'язання цих питань.

6. *Розробка та апробація ітеративної методології:* На основі інформації, отриманої за допомогою вищезазначених методів, буде сформульовано початкову методологію. Потім ця методологія буде протестована і перевірена за допомогою серії гіпотетичних і реальних сценаріїв.

Поєднання цих методів забезпечує як глибину, так і широту у вирішенні завдань дослідження. Завдяки поєднанню теоретичного аналізу та практичної перевірки, дослідження має на меті створити методологію, яка буде надійною для застосування в реальних умовах.

Структура дослідження. *Першій частині* дослідження присвячений Розділ 1, який зосереджений на історичній еволюції штучного інтелекту, а також впливі ШІ на сферу інженерії програмного забезпечення. У цій частині представлений детальний огляд та класифікація програмних артефактів, згенерованих інструментами ШІ разом із переліком переваг та ризиків використання таких артефактів.

Основна частина дослідження міститься у другому і третьому розділах. Вона зупиняється на основних правових питаннях, які є ключовими у правомірному використанні артефактів, згенерованих ШІ – проблеми права інтелектуальної власності, авторське право, а також проблеми ліцензування програмного забезпечення, яке використовує такі артефакти.

Фінальна частина дослідження представлена Розділом 4 і Розділом 5. Четвертий розділ присвячений розробленій методології дотримання правових вимог та вирішення правових конфліктів при використанні артефактів, створених ШІ. Останній розділ роботи присвячений опису проектування, розробки та тестування комплаєнс веб-платформи, яка є прикладним втіленням розробленої методології.

РОЗДІЛ 1. ШТУЧНИЙ ІНТЕЛЕКТ В СУЧАСНІЙ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Історична еволюція штучного інтелекту і його вплив на процес розробки програмного забезпечення

Функції та популярність штучного інтелекту зростають з кожним днем. Програми штучного інтелекту значно розвинулися за останні кілька років і знайшли своє застосування майже в кожному секторі бізнесу.

Штучний інтелект – це галузь комп'ютерних наук. Вона передбачає розробку комп'ютерних програм для виконання завдань, які в іншому випадку потребують людського інтелекту. Алгоритми ШІ можуть займатися навчанням, сприйняттям, вирішенням проблем, розумінням мови та/або логічними міркуваннями [105, с. 3].

Штучний інтелект у науковій літературі розділяють на декілька видів – вузький (Narrow or Weak AI), сильний (Strong or Generalized AI) і супер або свідомий (Super or Conscious AI).

Слабкий штучний інтелект – це система штучного інтелекту, розроблена і навчена для виконання конкретного завдання і застосовується в конкретній галузі. Наприклад, до них відносяться мовні перекладачі, рекомендаційні системи, віртуальні асистенти, та інтелектуальні фільтри спаму [64, с. 336].

Сильний штучний інтелект, також відомий як *загальний штучний інтелект,* – це система ШІ з узагальненими когнітивними здібностями людини. Потужна програма ШІ, зіткнувшись з невідомою проблемою, здатна знайти рішення без втручання людини [64, с. 336].

Супер штучний інтелект – це ШІ зі свідомістю на рівні людини, що вимагатиме від нього самосвідомості [124]. Оскільки ще й досі йдуть академічні та наукові дискусії щодо визначення свідомості, перспектива

створення свідомого штучного інтелекту у найближчому майбутньому є сумнівною.

Історія штучного інтелекту починається в 1950 році, коли британський вчений Алан Тьюрінг вперше підняв питання про можливий інтелект машин у своїй статті “Обчислювальні машини і розум”. Він описав тест, метою якого було визначення точки, в якій машини стають рівними людині за рівнем інтелекту [18].

1950-ті роки були присвячені розвитку нейромереж та введення термінів “штучний інтелект” і “машинне навчання” [18].

У наступні роки алгоритми машинного навчання вдосконалювалися, а комп’ютерні програми стали краще вирішувати проблеми. У 1966 Джозеф Вейценбаум створив перший віртуальний співрозмовник ELIZA, який став прототипом сучасних віртуальних асистентів і чат-ботів [18].

Хоча основні принципи були доведені, програми штучного інтелекту були доволі обмеженими і примітивним. Основною перешкою на шляху розвитку штучного інтелекту став брак обчислювальних потужностей, необхідних для виконання важливих завдань. Комп’ютери не могли зберігати достатньо інформації або обробляти її достатньо швидко. Саме через це фінансування проектів пов’язаних із ШІ також зменшилося, і дослідження призупинились на ціле десятиліття [110, с.17 – 21].

У 1980-ті фінансовий потік і розвиток алгоритмів відродив сферу ШІ. Цей період можна назвати ренесансом нейронних мереж, оскільки обчислювальні потужності дозволяли проводити дослідження у цій сфері. Джон Хопфілд і Девід Румельхарт популяризували методи “глибокого навчання” у своїх дослідженнях присвячених нейронним мережам, які дозволили комп’ютерам навчатися на власному досвіді. З іншого боку, Едвард Фейгенбаум представив експертні системи, які імітували процес

прийняття рішень людиною-експертом [18]. Також, на початку 1980-х років Японія оголосила про амбітний “Проект п’ятого покоління”, який був розроблений для проведення прикладних передових досліджень у галузі ШІ [80, с. 4].

Пізніше, у 1990-х і 2000-х роках, штучний інтелект розвивався, незважаючи на брак державного фінансування та низький суспільний резонанс. Провідну роль у розвитку технологій машинного навчання взяли на себе великі високотехнологічні компанії, яким вдалося досягти значних успіхів. Наприклад, у 1997 році чинний чемпіон світу з шахів, гросмейстер Гарі Каспаров зазнав поразки від комп’ютерної програми Deep Blue від IBM, яка грала в шахи [63, с. 3].

З часом увага знову повернулась до штучних нейронних мереж у вигляді глибокого навчання. У 2015 році Alpha Go, програма, розроблена Google, перемогла чемпіона світу з гри в го. Довгий час вважалося, що комп’ютери не можуть обіграти суперника у цій грі [63, с. 8-9].

Глибоке навчання і штучні нейронні мережі лежать в основі більшості додатків, які ми знаємо як “штучний інтелект”, наприклад, алгоритмів розпізнавання зображень, які використовує Facebook, а також алгоритмів розпізнавання мови і голосу в безпілотних автомобілях [63, с. 9].

Нещодавною значний прогрес у розвитку штучного інтелекту зробила компанія OpenAI. OpenAI – це дослідницька організація, заснована у 2015 році, що займається створенням безпечних та корисних систем штучного інтелекту, які можуть допомогти вирішити деякі з найбільших світових викликів [12]. З моменту свого заснування організація досягла значних успіхів у просуванні передових технологій машинного навчання та обробки природної мови, а її робота привернула увагу та похвалу експертів у цій галузі.

Наприклад, мовна модель ChatGPT і відповідний чат-бот, розроблений компанією OpenAI, побив усі рекорди популярності і використання. ChatGPT призначений для генерування відповідей на основі отриманих даних. Він був навчений на великій кількості текстових даних, що дозволяє йому розуміти і генерувати зв'язні та контекстно-релевантні відповіді на широкий спектр запитів і підказок. Він набрав мільйон користувачів всього за кілька днів після свого запуску 30 листопада 2022 року. Таке безпрецедентне зростання робить ChatGPT найшвидше зростаючим споживчим додатком в історії [17].

Багато компаній інтегрували свої продукти з ChatGPT для покращення функціоналу, пришвидшення обробки інформації та покращення користувацького досвіду. Крім того, популярність цього продукту стимулювали технічних гігантів як Google [13; 54], Microsoft [126] на створення і покращення своїх мовних моделей.

Поява інструментів, які дозволяють писати інструкції мовою зрозумілою кожному сприяло появі такого терміну як *інженерія запитів* – Prompt Engineering – це спосіб створення інструкцій для ШІ, які ефективно допомагають ChatGPT генерувати бажаний результат. Інженерія запитів передбачає розуміння поведінки моделі та коригування вхідних даних, щоб керувати відповідями моделі [82].

Можливості і потужність таких мовних моделей також *трансформує сферу розробки програмного забезпечення* від процесу планування і створення вимог для ПЗ до самої розробки, тестування і релізу програмного продукту. Особливого впливу зазнав саме етап написання програмного коду. Раніше розробники та тестувальники повинні були самостійно займатись пошуком рішень у випадку виникнення проблем при написанні коду. Основними джерелами пошуку інформації були віддалені публічні репозиторії, наприклад на платформі GitHub, або ж подібний до форуму StackOverflow, який дозволяє шукати

допомоги у інших розробників. На сьогодні, розробники ПЗ можуть створити запит для інструментів як ChatGPT і з великою ймовірністю швидко отримати потрібне рішення.

Крім того, розробка програмного забезпечення завжди вимагає написання шаблонного, так званого boilerplate коду. Раніше, автоматизувати цей процес могли хіба що інструменти створені спеціально для певної мови або фреймворку, наприклад Angular CLI, для генерації шаблону нових компонентів. Зараз є можливість інтегрувати інструменти штучного інтелекту навіть у редактори початкового коду, які дозволяють як автоматизувати написання шаблонного коду, так і аналізувати вже написаний код, доповнювати і модифікувати основну бізнес логіку програмного продукту.

Так, GitHub Copilot ділиться рекомендаціями, заснованими на контексті та стильових конвенціях проекту. Програміст має змогу переглядати рядки коду, заповнювати пропозиції щодо функцій і вирішувати, які з них прийняти, відхилити або відредагувати. GitHub Copilot інтегрується безпосередньо у редактор, включаючи Neovim, JetBrains IDE, Visual Studio та Visual Studio Code, і є досить швидким для використання під час набору тексту [58].

Таким чином, простеживши шлях ШІ від його концептуальних витоків до сучасних досягнень, можна побачити, що він відіграє трансформаційну роль у розробці програмного забезпечення. Переставши бути просто допоміжним інструментом, штучний інтелект перетворився на ключового партнера, відкриваючи нову еру винахідливості та раціональних практик у цій галузі.

1.2. Огляд програмних артефактів, створених штучним інтелектом, та їхнього потенційного впливу на індустрію

Останні розробки в галузі штучного інтелекту призвели до зміни парадигми розробки програмного забезпечення та технологічної індустрії загалом. Окрім автоматизації традиційних процесів, ШІ також уможливив створення програмних артефактів. Артефакти програмного забезпечення можна розглядати як матеріальні побічні продукти, що з'являються протягом життєвого циклу розробки програмного забезпечення.

А.О. Аронов, А.І. Дзюбенко визначають **програмний артефакт** як "реальну порцію інформації з програмування, яка може створюватися, змінюватися і використовуватися у діяльності, пов'язаній з виготовленням програмних продуктів (далі ПП) різного призначення. Артефакти є формалізованим результатом діяльності розробників програм, які відображають певну функціональність" [1, с. 87].

Д. Фернандез, Б. Пензенштадлер та ін. у своїй роботі надають два визначення артефакту у сфері програмної інженерії:

1. "Артефакт – це продукт праці, який створений, модифікований або використаний серією операцій, які мають цінність для виконання певної функції. Артефакти підлягають контролю якості та контролю версій і мають певні типи. Вони ієрархічно структуровані на елементи контенту, які визначають окремі сфери відповідальності і є результатом виконання одного завдання [47, с. 2783]."
2. "Артефакт – це самодостатній результат роботи, що має контекстно-залежне призначення і являє собою фізичну репрезентацію, синтаксичну структуру та семантичний зміст, що формують три рівні сприйняття [47, с. 2784]."

Програмне забезпечення – це найважливіша зі складових частин комп'ютерного забезпечення, яка включає комп'ютерні програми і дані,

що призначені для розв’язання певного кола задач і зберігаються на машинних носіях. Програмне забезпечення є або даними для використання в інших програмах, або алгоритмом, реалізованим у вигляді послідовності інструкцій для процесора[5, с. 5].

Кінцевий програмний продукт – це також окремий програмний артефакт, а також моделі програмного продукту, дизайн, його специфікації, сертифікати безпеки та захисту, плани управління проектом, посібники користувача, код, програмні компоненти, тощо [47, с. 2777; 79, с. 308].

Таким чином, програмні артефакти, які можуть бути створені штучним інтелектом можна розділити на **декілька груп за видом та призначенням**.

1. Сегменти коду та модулі. Сучасні системи ШІ, такі як нейронні мережі, що генерують код, можуть створювати функціональні блоки коду або навіть цілі модулі. Приклади включають генерацію шаблонного коду, конкретизацію шаблонів або більш складні структури коду, пристосовані до конкретних завдань. Такий функціонал мають наступні інструменти: OpenAI Codex [94], Tabnine [111], GitHub Copilot [58] та Replit Ghostwriter [55], Amazon CodeWhisperer [26].

2. Документація, специфікації, дизайн та посібники користувача. Інструменти автоматизованої генерації документації використовують ШІ для інтерпретації коду і створення вичерпної документації, забезпечуючи узгодженість і звільняючи розробників від цього часто виснажливого завдання. Theneo [117], DeveloperHub [34] пропонують такі можливості.

3. Сценарії тестування та тестові кейси. ШІ може генерувати тестові кейси на основі вимог до програмного забезпечення або шляхом аналізу наявного коду, забезпечуючи майже повне покриття і потенційно виявляючи граничні випадки, які можуть бути пропущені розробниками.

Одними із платформ які пропонують генерування тест-кейсів є Taskade [113] та Testsigma [15].

4. Конфігураційні файли та скрипти налаштування. За допомогою інструментів зі штучним інтелектом оцінка специфіки середовища та вимог до програмного забезпечення призводить до створення інсталяційних скриптів або конфігураційних файлів, які здатні полегшити швидку ініціалізацію розгортання програмного забезпечення.

Нові можливості запропоновані засобами штучного інтелекту **впливають на сферу інженерії програмного забезпечення.**

Роль розробника програмного забезпечення може еволюціонувати у міру того, як ШІ все тісніше переплітається з процесом створення програмного забезпечення. Розробники можуть перейти авторів коду до наглядачів створення коду, забезпечуючи відповідність результатів роботи ШІ цілям проекту та виправляючи будь-які нюанси, які ШІ може пропустити.

Завдяки інструментам ШІ, здатним генерувати код і пов'язані з ним артефакти, *розробка програмного забезпечення може стати більш доступною*. Люди, які не мають глибоких знань у сфері програмування, зможуть розробляти концепції та створювати додатки, використовуючи платформи, керовані ШІ, долаючи розрив між технічними та нетехнічними зацікавленими сторонами.

Поява артефактів, створених штучним інтелектом, може призвести до необхідності *встановлення нових стандартів і передових практик*. Подібно до того, як методології розробки програмного забезпечення еволюціонували від Waterfall до Agile, ми можемо стати свідками народження нових парадигм, зосереджених на розробці за допомогою ШІ.

Відбудеться *зрушення в забезпеченні якості програмного забезпечення*. Традиційне тестування програмного забезпечення покладається на людське розуміння і передбачувану поведінку

користувачів. Оскільки ШІ відіграє значну роль у створенні програмного забезпечення, можуть з’явитися нові методології тестування, які зосередяться на непередбачуваності результатів роботи ШІ та гарантуватимуть, що вони відповідають людським очікуванням.

Таким чином, вторгнення штучного інтелекту у створення програмних артефактів – це не просто технічна еволюція, це зміна самої структури індустрії програмного забезпечення. Ширші наслідки стосуються економіки, ролей, стандартів і доступності. Як і у випадку з будь-якою трансформаційною технологією, поєднання передбачення, адаптації та безперервного навчання буде ключовим для зацікавлених сторін, які долатимуть цей новий рубіж.

1.3. Переваги та ризики використання програмних артефактів, згенерованих штучним інтелектом

Впровадження ШІ у створення програмних артефактів викликало значний інтерес і дискусії в індустрії програмного забезпечення. Хоча штучний інтелект має низку значних переваг, він також несе в собі потенційні ризики, про які зацікавлені сторони повинні бути обізнані. У цьому підрозділі висвітлюються як переваги, так і виклики, пов’язані з результатами застосування штучного інтелекту.

Основними **перевагами** використання артефактів, створених ШІ є:

1. *Підвищення ефективності*. Штучний інтелект може швидко генерувати програмні артефакти, від блоків коду до документації, що може значно скоротити час розробки. Таке прискорення може призвести до швидшого завершення проекту та виведення його на ринок.
2. *Послідовність і стандартизація*. Автоматизовані процеси ШІ гарантують, що згенеровані артефакти підтримують стабільну

якість і відповідають визначеним стандартам, зменшуючи ймовірність помилок, спричинених людиною.

3. *Розширене розв'язання проблем*. Деякі моделі штучного інтелекту можуть ідентифікувати та реалізовувати складні алгоритми або рішення, які можуть зайняти багато часу або бути складними для розробників.
4. *Забезпечення правового комплаєнсу*. Деякі інструменти ШІ можна навчити дотримуватися певних регуляторних або правових стандартів, гарантуючи, що створені програмні компоненти будуть відповідати їм за замовчуванням.
5. *Роз'яснення щодо інтелектуальної власності (ІВ)*. Створення артефактів за допомогою ШІ може призвести до більш чіткого розмежування інтелектуальної власності, особливо якщо результати роботи ШІ вважатимуться нетрадиційним інтелектуальним внеском, що спрощує питання ліцензування та повторного використання.
6. *Доступність і демократизація*. Завдяки інструментам штучного інтелекту люди без глибокого досвіду програмування можуть створювати необхідні програмні компоненти, що робить створення програмного забезпечення більш інклюзивним.

Незважаючи на достатню кількість переваг, які надають засоби штучного інтелекту, використання програмних артефактів, створених ШІ, несе в собі певні ризики:

1. *Надмірна залежність від ШІ*. Надмірна залежність може призвести до того, що розробники недооцінюватимуть свій критичний аналіз. Через це вони потенційно будуть пропускати помилки, допущені ШІ або не враховувати важливі нюанси, про які ШІ-ту невідомо. Крім того,

необізнаність ШІ з контекстом розробки, вимогами до ПЗ або його частини і стеком використовуваних технологій призводить до високої ймовірності генерації неточних, неповних або хибних результатів. Саме тому критична оцінка створених програмних продуктів розробником відіграє надзвичайну роль у створенні якісного програмного забезпечення.

2. *Непередбачувані результати*. ШІ, особливо якщо він покладається на складні моделі, може давати неочікувані або не зовсім зрозумілі результати, що може призвести до потенційних проблем у майбутньому.
3. *Втрата майстерності*. Людський фактор у розробці програмного забезпечення – коли розробники глибоко розуміють і створюють свій код – може зменшитися, що може призвести до потенційної ерозії навичок. Це також може призвести до поганої обізнаності у вихідному коді, що призведе до складності підтримування ПП.
4. *Етичні проблеми та упередженість*. Моделі ШІ можуть створювати програмні артефакти, які несвідомо несуть в собі упередженість, присутню в їхніх навчальних даних. Це може призвести до ненавмисних і потенційно шкідливих упереджень у кінцевих програмних продуктах.
5. *Занепокоєння щодо втрати роботи*. Хоча ШІ може спростити багато аспектів розробки програмного забезпечення, існують побоювання, що він може витіснити певні робочі ролі, що призводить до дебатів про майбутній ландшафт галузі.
6. *Невизначеність прав власності та прав інтелектуальної власності*. У правовому полі різних держав може бути нечітко визначено, кому належать права на артефакти, створені ШІ. Це

може призвести до конфліктів щодо авторських прав, патентів та інших прав інтелектуальної власності [92].

7. *Питання відповідальності*. У випадках, коли програмні артефакти, створені ШІ, виходять з ладу або завдають шкоди, визначення відповідальності може бути складним. Невизначеним залишається питання чи несуть відповідальність розробники ШІ, власники ШІ як програмного продукту, користувачі інструменту ШІ, чи сам ШІ.
8. *Конфіденційність і захист даних*. Моделі штучного інтелекту часто потребують великих обсягів даних для навчання. Створення програмних артефактів на основі потенційно конфіденційних або персональних даних може призвести до правових проблем, пов’язаних з конфіденційністю та захистом таких даних [65].
9. *Невизначеності з ліцензуванням і повторним використанням*. Юридичні ускладнення можуть виникнути, якщо програмні артефакти, створені ШІ, будуть інтегровані в більші програмні системи. Розуміння умов ліцензування та забезпечення того, щоб повторно використані компоненти не порушували жодних умов, може бути непростим завданням [71].

Таким чином, поява штучного інтелекту у створенні програмних артефактів несе в собі поєднання захоплюючих можливостей і складних викликів, особливо в правовій сфері. Оскільки ШІ продовжує розвиватися, проактивні правові рамки та глибоке розуміння тонкощів, пов’язаних з ним, матимуть вирішальне значення для навігації на цій новій арені.

РОЗДІЛ 2. ПРОБЛЕМИ ПРАВА ІНТЕЛЕКТУАЛЬНОЇ ВЛАСНОСТІ НА ПРОГРАМНІ АРТЕФАКТИ ЗГЕНЕРОВАНІ ШТУЧНИМ ІНТЕЛЕКТОМ

2.1. Проблеми авторського права

Програмне забезпечення може бути різним, охоплюючи комп'ютерні програми, веб-сайти, мобільні додатки, дистрибутиви тощо. По суті, кожен з цих форматів складається з програмного коду та елементів індивідуалізації [6, с. 219].

Стрімке поширення штучного інтелекту відкриває нову еру інновацій, але водночас створює безліч викликів для існуючої системи інтелектуальної власності. Зокрема, сфера програмних артефактів, створених штучним інтелектом, створює складні дилеми щодо права власності, авторства та визначення прав.

Існує безліч інструментів на основі ШІ, які автоматично генерують вихідний код програмного забезпечення, наприклад Github Copilot, ChatGPT, Ghostwriter та інші.

Такі помічники для програмування зі штучним інтелектом надають розробникам коду привабливу можливість створювати програмне забезпечення швидше, продуктивніше, ефективніше і потенційно з меншою кількістю помилок.

Водночас, продукти згенеровані, ШІ ставлять нові виклики перед правом інтелектуальної власності (далі, ІВ), оскільки його принципи грунтуються саме на людській творчості та праці, а також породжують

нові правові питання перед спеціалістами з питань ІВ щодо використання інструментів ШІ для розробки програмних продуктів. Наприклад, чи може згенерований код бути захищений авторським правом і патентним правом? Чи можна вважати згенерований код похідною роботою від іншого захищеного авторським правом коду – наприклад, захищеного авторським правом коду, який використовується для навчання ШІ? Чи підпадає використання матеріалів захищених авторським правом для тренування моделей штучного інтелекту під доктрину добросовісного використання (fair use doctrine)?

Основою ІТ-індустрії є **права інтелектуальної власності**. Цей термін охоплює різноманітні **особисті немайнові** та **майнові права** (ч. 2 статті 418 ЦКУ) [9], пов'язані з такими активами, як програмне забезпечення. Кожна форма вираження ІВ є самостійним активом, інтегрованим у ширшу систему власності. Існують правові норми, що передбачають різні способи захисту цих майнових і немайнових прав залежно від їхніх різних типів.

Цивільний кодекс України відносить комп'ютерні програми до об'єктів права ІВ (ч. 1 статті 420 ЦКУ) [9].

Існує чотири **основні форми захисту прав** ІВ, які застосовуються до програмного забезпечення: *патенти*, *авторські права*, *комерційні таємниц*і та *торгові марки*, кожна з яких пропонує окрему форму правового захисту. У той час як патенти, авторські права та комерційні таємниці захищають безпосередньо технологію, торгові марки захищають назви і символи, які допомагають диференціювати продукт на ринку [6, с. 219]. Проте лише дві форми захисту – авторське право і патентування – доцільно розглядати у розрізі використання програмних артефактів, згенерованих штучним інтелектом.

Віднесення програмного забезпечення до певного виду інтелектуальної власності було особливо складним завданням через його

подвійний характер, що створює перешкоди для тих, хто намагається узгодити його з попередньо встановленими правовими категоріями. Тому були спроби віднести її до категорії авторського права, патентів або комерційної таємниці, і навіть запропонувати унікальне право на програмне забезпечення *sui generis* [70].

Складність правової класифікації випливає з багатогранної природи програмного забезпечення. На відміну від окремого об'єкта, ПЗ складається з декількох компонентів, які можуть підпадати під різні категорії захисту ІВ. Визначення ПЗ як послідовності інструкцій, що спрямовують комп'ютер на отримання певного результату, допомагає пролити світло на відповідний тип захисту ІВ. Спочатку ці інструкції сформульовані у вигляді вихідного коду або скриптових рядків на комп'ютерній мові, що, завдяки своїй письмовій формі, логічно позиціонує програмне забезпечення для захисту авторського права на кшталт літературного твору.

Ця точка зору підкріплюється різними міжнародними договорами. Наприклад, *стаття 4* Договору Всесвітньої організації інтелектуальної власності про авторське право (WCT) [2], *стаття 10* Угоди про торговельні аспекти прав інтелектуальної власності (TRIPS Agreement) [7] та *стаття 1* Директиви Ради Європейського Співтовариства 91/250/ЄЕС [30] визначають програмне забезпечення як літературні твори, що мають право на захист авторським правом. Також, відповідно до п. 2 ч.1 статті 433 ЦКУ [9] комп'ютерні програми є об'єктами авторського права.

Проблема жорсткої класифікації програмного забезпечення як літературного твору виникає тоді, коли визнається, що комп'ютерні програми містять елементи, які зазвичай не захищені авторським правом. Програмне забезпечення виходить за рамки простого літературного вираження; функціональність коду не прив'язана до його граматичної структури. Дві програми можуть мати абсолютно різний вихідний код, але

надавати схожий набір інструкцій для досягнення схожих або аналогічних результатів. Цей сценарій підкреслює часто обговорювану *доктрину "ідея/вираження" (idea/expression)*, що займає центральне місце в дебатах про захист програмного забезпечення.

Доктрина "*ідея/вираження"* домінує у США і полягає в тому, що ідеї не захищені авторським правом, захищене лише їх вираження. Тобто ідея або концепція певної програми не охороняється. Крім того, описання певних дій, які мають утилітарне практичне значення та надалі виконуються третьою особою, авторським правом не охороняються [10, с. 69]. Таким чином, ця концепція відокремлює загальну ідею, від її конкретного вираження, припускаючи, що в той час, коли є можливість захистити авторським правом конкретне вираження, неможливо захистити авторським правом загальну ідею.

Хоча очевидно, що копіювання значних частин вихідного коду в іншу програму є порушенням авторських прав, такі випадки відносно рідкісні. Суть питання змістилася в бік захисту нелітературних компонентів, вбудованих у програмне забезпечення.

До випадків копіювання функціональних частин програми і визначення чи це порушує закони про авторське право довго застосовували доктрину "ідея/вираження". Проте ситуація ускладнилася із запровадженням у США громіздкої правової доктрини під назвою "*Абстрагування-фільтрація-порівняння*". Ця доктрина використовується для визначення порушення авторських прав шляхом порівняння різних частин програмного забезпечення, і її використовували і критикували суди у Великій Британії [70].

У Великій Британії ця тема була піднята у судовій справі *Navitaire Inc. v. Easyjet Airline Co. & Bulletproof Technologies Inc. [2004] EWHC 1725 (Ch)*. Суд вирішив обмежити захист авторських прав на нелітературні елементи програмного забезпечення, тобто *функціональні частини*

програмного забезпечення не повинні підпадати під захист авторських прав. Це рішення чітко визначило, що захист авторського права не повинен поширюватися на те, як функціонує програмне забезпечення, а має охоплювати лише специфічний спосіб, у який ці функції виражені в коді [3, с. 9].

Крім того, суд урахував, що особливим аспектом комп'ютерних програм є те, що існує кілька різних способів отримати подібний чи однаковий результат. Отже, "ділова логіка", тобто функціональність, програми не може охоронятися авторським правом. В іншому випадку відбулося б значне необґрунтоване розширення авторських прав [3, с. 9].

Питання авторства щодо програмних артефактів, створених штучним інтелектом, перебуває на передньому плані дебатів навколо прав інтелектуальної власності у сфері програмного забезпечення. Суть питання полягає в традиційному авторському праві, яке ґрунтується на людському авторстві, і в тому, як воно поширюється (або не поширюється) на твори, створені штучним інтелектом.

У деяких країнах, що наслідують ***британську традицію***, зокрема у Великій Британії, Ірландії, Новій Зеландії та Південній Африці, існує механізм, який дозволяє обійти вимогу про авторство людини, визнаючи авторство "комп'ютерних творів" у випадках, коли авторство людини не може бути встановлене. Відповідно до цих режимів, авторство – і, відповідно, авторські права – надаються особі, яка здійснила заходи, необхідні для створення твору [67].

У доповіді ***Європейського парламенту*** наголошується на необхідності вирішення питань авторського права, пов'язаних з творами, створеними ШІ. У ньому підкреслюється, що людська творчість повинна складати основу системи інтелектуальної власності, і ставиться питання про те, якою мірою твори, створені ШІ, можуть бути простежені до людей, які їх створили [41]. Ця позиція також підтверджується практикою Суду

Європейського Союзу (Court of Justice of the European Union). Суд Європейського Союзу також неодноразово заявляв, зокрема у своєму знаковому рішенні у справі *Infopaq (C-5/08 Infopaq International A/S v Danske Dagbaldes Forening)*, що авторське право поширюється лише на оригінальні твори, і що оригінальність повинна відображати "власне інтелектуальне творіння автора". Зазвичай це розуміють як те, що оригінальний твір повинен відображати особистість автора, а це означає, що для існування твору, який охороняється авторським правом, необхідна наявність автора-людини [62].

У Сполучених Штатах існує усталений принцип, згідно з яким захист авторських прав вимагає авторства людини. Бюро авторських прав США (U.S. Copyright Office, USCO) підтвердило, що термін "автор" виключає нелюдей, і в березні 2023 року опублікувало вичерпну заяву про те, що воно не визнає ШІ автором і не прийме реєстрацію авторських прав на твори, створені ШІ. У контексті генеративного ШІ це означає, що "якщо традиційні елементи авторства твору були створені машиною, твір не має людського авторства, і Відомство не зареєструє його" [21; 112].

Бюро авторських прав США також заявив, що твори, не створені автором-людиною, не підлягають охороні авторським правом, що відповідає загальній позиції більшості правових юрисдикцій, які надають авторське право лише на оригінальні твори, створені авторами-людьми [62].

Часто права та умови використання контенту, згенерованого ШІ, згадуються в Умовах використання (Terms of Use) на офіційних сайтах організацій-власників інструменту ШІ. Так, в ***Умовах використання OpenAI***, вказано, що користувач може "надавати вхідні дані до Сервісів ("Вхідні дані") та отримувати вихідні дані, що генеруються та повертаються Сервісами на основі Вхідних даних ("Вихідні дані"). Вхідні дані та Вихідні дані разом є "Контентом". У відносинах між сторонами і в

межах, дозволених чинним законодавством, ви (прим. користувач) є власником усіх Вхідних даних. *За умови дотримання вами цих Умов, OpenAI передає вам всі свої права, права власності та інтереси на Вихідні дані.* Це означає, що ви можете використовувати Контент з будь-якою метою, включаючи комерційні цілі, такі як продаж або публікація, якщо ви дотримуєтеся цих Умов. […] Ви несете відповідальність за Контент, в тому числі за те, щоб він не порушував чинне законодавство або ці Умови [114]."

Пункт 2 Угоди користувача GitHub Copilot визначає, що "GitHub не заявляє жодних прав власності на Рекомендації[1]. Ви (прим. користувач) зберігаєте право власності на свій код [57]."

Інша платформа Replit, яка слугує майданчиком для поширення контенту з відкритим вихідним кодом і яка пропонує інструмент ШІ Repl AI з їхньою LLM Ghostwriter, визначає, що код, який знаходиться в публічних Repls (прим. репозиторіях), автоматично підпадає під MIT open-source ліцензію [45]. Додавання файлу ліцензії до публічного Repl не перешкоджає розповсюдженню Repl за ліцензією MIT. Щоб файл ліцензії набув чинності, Repl слід зробити приватним [75]. Хоча Умови використання платформи явно не вказують право на контент згенерований їх ШІ, доцільно допустити, що код, який знаходиться у репозиторіях користувача належить йому, навіть код згенерований за допомогою ШІ. Відповідно до Replit, їх модель тренується лише на даних, які зберігаються у публічних репозиторіях, у приватних – лише за згоди власника [29].

Amazon також представила інструмент ШІ CodeWhisperer. Відповідно до офіційного сайту продукту, розробник володіє кодом, який він пише, включно з будь-якими пропозиціями, наданими CodeWhisperer.

[1] "Рекомендації" - це код, функції та інші результати, згенеровані GitHub Copilot (GitHub Copilot Product Specific Terms)

Розробник несете відповідальність за код, який він/вона пише, включаючи пропозиції CodeWhisperer, які програміст приймаєте [25].

Отже, хоча традиційне поняття авторства в авторському праві ставиться під сумнів через появу робіт, створених ШІ, правова спільнота дотримується позиції, що ШІ не може визнаватись автором, за деякими винятками у британській традиції. Контент, створений генеративним ШІ, не підлягає захисту авторським правом, незважаючи на те, що власником згенерованого контенту визнається особа, що надала вхідні дані для генерування результату.

Перехід до визнання творів, створених ШІ, може вимагати переоцінки авторського права, щоб воно залишалося актуальним і ефективним у захисті прав авторів і водночас сприяло інноваціям у сфері ШІ та розробці програмного забезпечення.

2.2. Патентоспроможність

У сфері прав інтелектуальної власності патенти слугують найважливішим механізмом захисту новизни та винахідливості, втілених у винаходах. На відміну від авторських прав, які захищають вираження ідей, патенти захищають фактичне втілення або виконання ідей. Ця відмінність, яку часто називають дихотомією “ідея/вираження”, лежить в основі необхідності патентного захисту, особливо у сфері програмного забезпечення та коду, згенерованого штучним інтелектом, де функціональна сутність – це те, що в першу чергу визначає цінність та інновації. У патентному праві не існує дихотомії ідея/вираження. Якщо ідея відповідає умовам патентоспроможності, то їй надається патентна охорона.

Патентоспроможність – це відповідність винаходу умовам надання йому патентного захисту. Це важливий аспект права інтелектуальної власності, який заохочує інновації, надаючи винахідникам

виключні права на їхні винаходи на певний період, зазвичай 20 років. Основними критеріями патентоспроможності є новизна, винахідницький рівень (або неочевидність) та промислова придатність (або корисність) [4, с. 54]. Нижче наведений аналіз того, як патентоспроможність застосовується до програмних артефактів, таких як код, відповідно до різних правових систем.

Угода про торговельні аспекти прав інтелектуальної власності є важливим міжнародним договором, який встановлює мінімальні стандарти патентного захисту, які в деяких юрисдикціях поширюються і на програмне забезпечення. ***Пункт 1, статті 27 Угоди*** визначає, що патенти повинні бути доступними для "будь-яких винаходів, продуктів чи процесів у всіх сферах технології за умови, що вони є новими, включають винахідницький крок і придатні для промислового використання [8]." Однак угода не розглядає питання патентоспроможності програмного забезпечення або коду, створеного штучним інтелектом.

Паризька конвенція про охорону промислової власності забезпечує основу для патентного захисту, але прямо не згадує програмне забезпечення або винаходи, створені штучним інтелектом.

Згідно з ***Європейською патентною конвенцією***, програмне забезпечення як таке не є об'єктом патентування. Однак програмне забезпечення може бути запатентоване, якщо воно забезпечує нове і неочевидне технічне рішення технічної проблеми [43].

ЄС визнає необхідність захисту технічних рішень, створених за допомогою технології ШІ, в рамках правової бази інтелектуальної власності, щоб заохотити інвестиції в ШІ та підвищити правову визначеність для громадян, бізнесу та винахідників. Однак особливості патентування винаходів, створених за допомогою ШІ, залишаються предметом постійних дискусій [42].

Патентне законодавство США дозволяє запатентувати програмне забезпечення, якщо програмне забезпечення, покращує “функціональність комп’ютера”, тобто підвищує швидкість обчислень або зменшує кількість необхідних обчислювальних ресурсів, або виконує обчислювальні завдання у нетрадиційний спосіб. У справі *Alice Corp. v CLS Bank International* Верховний суд США встановив рамки для визначення патентоспроможності програмного забезпечення, зосередившись на тому, чи додає програмне забезпечення трансформаційну винахідницьку концепцію до будь-якої абстрактної ідеї, яку воно може втілювати [16].

Проте питання, щодо програмних артефактів і ПЗ створеного, за допомогою засобів ШІ залишається дискусійним. У США нещодавнє рішення Федерального окружного суду у справі *Thaler v. Vidal* роз’яснило, що ШІ не може бути включений до переліку “винахідників” для цілей отримання патенту. Відомство США з патентів і торгових марок (USPTO) також звернулося до громадськості з проханням прокоментувати закони, що регулюють винахідництво ШІ, що свідчить про еволюційний характер дискусії навколо ШІ і патентного права [86].

Хоча конкретної інформації про позицію країн британської традиції щодо патентування коду, створеного ШІ, немає, ширший дискурс про винахідництво та патентне право у сфері ШІ, ймовірно, вплине на ці юрисдикції.

Традиційне патентне право виходить з припущення, що винахідниками є люди, що створює значну перешкоду, коли мова йде про артефакти, створені ШІ. Розвиток технологій ШІ кидає виклик існуючій патентній системі, що вимагає перегляду міжнародних договорів і національних законодавств з урахуванням винаходів, створених за допомогою ШІ [53; 121]. Необхідність патентування винаходів, створених за допомогою штучного інтелекту, для розвитку науки і технологій визнається у всьому світі. Однак нинішні розрізнені правила в різних

юрисдикціях призводять до відсутності єдності в тому, як до винаходів, створених за допомогою ШІ, ставляться з точки зору патентного права [11].

Таким чином, розвиток технологій штучного інтелекту зумовлює нагальну потребу переглянути і, можливо, змінити існуючу патентну законодавчу базу як на національному, так і на міжнародному рівнях, щоб вона залишилася актуальною та ефективною в просуванні інновацій, забезпечуючи при цьому належний захист винаходів, створених за допомогою штучного інтелекту.

РОЗДІЛ 3. ПРОГРАМНИЙ КОД, СТВОРЕНИЙ ШТУЧНИМ ІНТЕЛЕКТОМ: ПРОБЛЕМИ З ДЖЕРЕЛАМИ ТА ЛІЦЕНЗУВАННЯМ

3.1. Ліцензування в контексті пропрієтарного та open source програмного забезпечення

Поява ШІ призвела до нової парадигми в розробці програмного забезпечення, зокрема завдяки системам ШІ, здатним генерувати програмний код. Глибоке розуміння системи ліцензування, впливу навчальних даних на код, створений ШІ, і правових наслідків цього є ключовим для розробки методології розв'язання правових конфліктів, які можуть виникнути.

Моделі штучного інтелекту, які використовують для генерування програмних артефактів, вимагають специфічних даних для їх навчання – коду, написаного програмістами.

Так, відомий інструмент *GitHub Copilot* працює на основі генеративної моделі штучного інтелекту, розробленої GitHub, OpenAI та Microsoft. Вона була навчена на тексті природною мовою та вихідному коді з загальнодоступних джерел, включаючи код у публічних репозиторіях на GitHub [58].

Інший відомий інструмент *Replit AI*, який пропонує веб-платформа для поширення і написання коду Replit, повертає результати, згенеровані великими мовними моделями, навченими на загальнодоступному коді та оптимізованими за допомогою Replit [28]. Сервіс використовує публічні репозиторії, які називаються repls, створені користувачами Replit для навчання їх моделі ШІ [45].

Генеративний сервіс ШІ від Amazon CodeWhisperer працює на основі фундаментальної моделі, навченої на різних джерелах даних, включаючи Amazon і відкритий код [25].

Таким чином, зрозуміло, що дані, які використовують для тренування моделей штучного інтелекту, – це у своїй більшості проекти з відкритим вихідним кодом (open-source projects).

Проекти з відкритим вихідним кодом часто містять відповідну open-source ліцензію, яка визначає обмеження щодо використання оригіналу, а також права споживачів цього програмного забезпечення.

Ліцензія на програмне забезпечення – це юридичний інструмент, що дозволяє іншим особам, крім власника, використовувати, змінювати та/або розповсюджувати програмне забезпечення.

Коли програмне забезпечення розповсюджується, воно, як правило, супроводжується ліцензією, в якій перераховані права які зберігаються та передаються автором або дистриб'ютором, а також умови використання ПЗ. Саме використання ПЗ умовно передбачає прийняття умов цієї ліцензії [91, с. 1].

Деякі платформи, такі як Replit, закріплюють у своїх Умовах використання правила, щодо ліцензування та ліцензію за замовчуванням, якщо ліцензія не вказана для публічних репозиторіїв [45]. GitHub у свою чергу надає рекомендації щодо включення ліцензії у open-source проект, проте зазначає, що у випадку відсутності ліцензії, загальні принципи та правила авторського права будуть застосовані, а саме: користувач зберігає всі права на свій вихідний код, і ніхто не може відтворювати, поширювати або створювати похідні роботи на основі роботи користувача. Проте Умови використання GitHub також фіксують право кожного користувача переглядати код у публічних репозиторіях і робити так звані fork – копії публічних репозиторіїв – для подальшого особистого використання [76].

Проекти з відкритим програмним кодом (open-source software) завжди протиставляються *пропрієтарному ПЗ* (proprietary software).

Пропрієтарне програмне забезпечення, як правило, належить конкретній особі, яка претендує на всі права інтелектуальної власності на програмне забезпечення, ліцензує ці права за певну плату та ініціює судові позови проти будь-кого, хто використовує ПЗ без ліцензії [106, с. 211].

Зазвичай, **програмне забезпечення з відкритим кодом** створюється в результаті "групових зусиль" (коли автори програмного забезпечення передають свої права інтелектуальної власності спільноті/організації, відповідальній за ПЗ з відкритим кодом); і ліцензується безоплатно (або може бути вільно доступним без ліцензійних обмежень як матеріал, що є суспільним надбанням) [106, с. 211].

Хоча відкрите програмне забезпечення ґрунтується на альтернативному методі розробки, основоположний принцип спільноти відкритого ПЗ виходить за межі інтересів професійних програмістів. Відкритий код пропагує ідею, що коли користувачі отримують доступ до вихідного коду ПЗ, яке вони використовують, кінцеві користувачі отримують величезну свободу створювати або вдосконалювати ПП за допомогою власних обчислювальних інструментів. Коли кінцеві користувачі мають можливість читати, розповсюджувати або змінювати вихідний код свого ПЗ, це не лише розширює базу знань, що є суспільним надбанням, але й покращує життя користувачів завдяки розширенню можливостей комп'ютерних технологій [35].

Сторонами у правових відносинах щодо ліцензування є ліцензіат і ліцензіар. Відповідно до ч. 1 статті 1108 Цивільного Кодексу України, ***ліцензіар*** – це особа, яка має виключне право дозволяти використання об'єкта права інтелектуальної власності іншій особі (***ліцензіату***). Дозвіл на використання об'кта права інтелектуальної власності називається ***ліцензією*** [9].

Пропрієтарне програмне забезпечення ліцензується на умовах, призначених для захисту права власності ліцензіара і обмеження використання ліцензіатом. Такі ліцензії зазвичай забороняють доступ до вихідного коду і мають обмеження на використання, щоб гарантувати, що ліцензіар контролює доступ і використання програми, яка ліцензується. Такі ліцензії також часто обмежують право користувача вносити зміни, переліцензувати або розповсюджувати програмне забезпечення. Бізнес-модель пропрієтарного програмного забезпечення базується на отриманні прибутку за рахунок ліцензійних платежів та супутніх послуг з підтримки програмного забезпечення [106, с. 213].

ПЗ з відкритим вихідним кодом ліцензується за відкритою, "заснованою на спільноті" (community-based) концепцією [106, с. 213].

Основна мета open-source ліцензування – позбавити інших права *ексклюзивно* використовувати твір. Зазвичай, щоб дозволити своїм творам досягти широкої аудиторії, і щоб заробляти на життя створенням творів, автори повинні передати всі або майже всі права, надані авторським правом, тим суб'єктам, які здатні розповсюджувати і, таким чином, експлуатувати цей твір [74, с. 4].

Хоча існує безліч ліцензійних угод/схем для відкритого коду, open-source ліцензії зазвичай включають право ліцензіата мати доступ до вихідного коду і копіювати, модифікувати та розповсюджувати ПЗ з будь-якою метою без сплати винагороди ліцензіару. Ліцензія з відкритим кодом зазвичай не містить обмежень на кількість користувачів і не обмежує способи використання програмного забезпечення ліцензіатом. Однак багато open-source ліцензійних угод вимагають, щоб будь-яке повторне розповсюдження програмного забезпечення (і часто будь-яких похідних творів) розповсюджувалося на тих самих умовах, що й оригінальна ліцензія на відкритий вихідний код. Такі ліцензії називають **copyleft open-source ліцензіями**. Ці вимоги гарантують, що будь-яке розповсюдження

модифікованого коду зберігає аспекти відкритого коду, що містяться в оригінальній ліцензії, щоб нова, покращена версія була також доступна користувачам, як і оригінальний код [106, с. 213].

Ендрю Лорент наводить перелік **принципів open-source ліцензій** :

1. ***Принцип відкритого розповсюдження (open distribution)***: споживач ПЗ з відкритим кодом може вільно розповсюджувати (в обмін на оплату чи ні) копії твору.
2. ***Принцип відкритої модифікації (open modification)***: споживач може вільно модифікувати оригінальний твір і розповсюджувати ці похідні роботи безоплатно чи за плату.
3. ***Принцип обмеження наступництва (generational limitation)***: копії твору, які споживач розповсюджує, чи то оригінальний твір, чи то його/її власний похідний твір, самі мають бути ліцензовані у спосіб, сумісний з оригінальною ліцензією. Ліцензія з відкритим вихідним кодом може вимагати, щоб похідні твори розповсюджувалися на тих самих умовах, на яких ліцензіату було дозволено доступ до твору за оригінальною ліцензією. Це означає, що люди, які отримують копії цих творів, самі повинні мати можливість розповсюджувати оригінал і створювати похідні твори на основі оригіналу, за умови, що вони дозволяють іншим робити те саме. Цей принцип запобігає “закриттю” відкритого коду і спонукає споживачів дотримуватись загальних цінностей і принципів open-source спільноти [35, с. 8 – 11].

Перевагами відкритого ліцензування порівняно з моделями ліцензування пропрієтарного комерційного програмного забезпечення є:

1. *спонукання інновацій* – будь-хто може зробити внесок у проекти з відкритим кодом без жодних обмежень, що пришвидшує розробку і покращує якість продукту;

2. *надійність* – оскільки велика кількість обізнаних спеціалістів задіяні у open-source проекті, покращення впроваджуються швидше. Крім того, контроль змін здійснюється також колективно. Будь-яка пропозиція, щодо внесення змін у проект проходить ретельну перевірку інших контриб'юторів;
3. *довговічність* – компанія може відмовитись від підтримування і розвитку свого комерційного програмного забезпечення. Програма швидко виходить з ужитку, ніхто не має доступу до вихідного коду, крім самої компанії-власниці продукту. З іншого боку, open-source проекти залишаються у відкритому доступі, будь-хто може знайти нове застосування застарілому коду, адаптувати і відродити його [35, с. 6 – 7].

Найпоширеніші open-source ліцензії та їх порівняльна характеристика наведені у **Таблиці 1**. Різні ліцензії проаналізовані за такими основними, проте не вичерпними, *критеріями*:

1. **Права:**
   - 1.1. ***Комерційне використання***: чи можуть ліцензовані матеріали та похідні від них роботи бути використані в комерційних цілях.
   - 1.2. ***Розповсюдження***: чи можуть ліцензійні матеріали бути розповсюджені.
   - 1.3. ***Модифікація***: чи можуть ліцензійні матеріали бути модифіковані.
   - 1.4. ***Приватне використання***: чи можна використовувати та змінювати ліцензійні матеріали в приватному порядку.
   - 1.5. ***Патентне використання***: чи передбачає ліцензія явне надання патентних прав від співавторів та інші.
2. **Обов'язки:**

2.1. ***Повідомлення про ліцензію та авторське право***: чи повинні бути додані копія ліцензії та повідомлення про авторські права до ліцензованого матеріалу.

2.2. ***Зміна стану ліцензованого матеріалу***: чи є додаткові вимоги при модифікації ліцензованого матеріалу та інші.

3. **Обмеження**:

3.1. ***Відповідальність***: чи включає ліцензія обмеження відповідальності.

3.2. ***Гарантії***: чи включає ліцензія гарантії та інші.

Таблиця 1

Порівняльна характеристика open-source ліцензій

Назва ліцензії	Опис
Massachusetts Institute of Technology License або X License (MIT) [81]	**Резюме** Коротка і проста ***дозвільна ліцензія*** з умовами, що вимагають лише збереження авторських прав і повідомлень про ліцензію. Ліцензовані твори, модифікації та більші твори можуть розповсюджуватися на інших умовах і без вихідного коду. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання **Обов'язки:** споживач повинен надати копію ліцензії та повідомлення про авторські права до ліцензованого матеріалу. **Обмеження:** ✗ виключена відповідальність автора та власника авторських прав за будь-які збитки або претензії; ✗ не включає жодних гарантій.
Berkeley Software Distribution Licenses (BSD)	**Резюме** ***Дозвільна ліцензія***, подібна до ліцензії MIT, що надає велику свободу з мінімальними вимогами до використання, головним

[116; 115]	чином щодо збереження повідомлень про авторські права. Ліцензія BSD має кілька варіантів, зокрема версії з 2-м та 3-м пунктами, які часто називають “BSD 2-Clause” та “BSD 3-Clause” відповідно. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання **Обов’язки:** надати копію ліцензії та повідомлення про авторські права на ліцензований матеріал. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій; ✗ BSD дуже схожа на ліцензію MIT, але версія “BSD 3-Clause” містить додатковий пункт, який забороняє використання імені ліцензіара для просування похідних продуктів без дозволу.
Apache License 2.0 [19]	**Резюме** ***Дозвільна ліцензія***, основні умови якої вимагають збереження авторських прав та повідомлень про ліцензію. Контриб’ютори надають чітко виражені патентні права. Ліцензовані твори, модифікації та більші твори можуть поширюватися на інших умовах і без вихідного коду. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання ✓ Патентні права **Обов’язки:** - надати копію ліцензії та повідомлення про авторські права до ліцензованого матеріалу; - зміни, внесені до ліцензійного матеріалу, повинні бути

	задокументовані. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій; ✗ у цій ліцензії явно зазначено, що вона не надає права на торговельну марку.
Mozilla Public License (MPL 2.0) [84]	**Резюме**. ***Слабка копілефт ліцензія***, що дозволяє інтеграцію з пропрієтарним програмним забезпеченням, але за певних умов гарантує, що ліцензоване програмне забезпечення залишається з відкритим вихідним кодом. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання **Обов'язки:** - надати копію ліцензії та повідомлення про авторські права до ліцензованого матеріалу; - зміни, внесені до ліцензійного матеріалу, повинні бути задокументовані; - оприлюднення вихідного коду при розповсюдженні ліцензійного матеріалу; - розповсюдженя скомпільованої форми ПЗ повинно залишатися на умовах MPL, або субліцензувати її на інших умовах. Однак нові умови ліцензії для скомпільованої форми не повинні намагатися обмежувати або змінювати права одержувачів щодо вихідного коду, як це передбачено MPL. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій; ✗ не надає жодних прав на торгові марки, знаки обслуговування або логотипи, за винятком тих, що можуть бути необхідними для дотримання вимог щодо

	повідомлення, зазначених у ліцензії; ✗ містить положення, пов'язані з патентними правами, ці права є обмеженими і надаються на певних умовах.
Common Development and Distribution License (CDDL) [27]	**Резюме** ***Копілефт ліцензія***, яка дозволяє змішувати програми під CDDL з програмами під іншими ліцензіями. Вона вимагає, щоб зміни до вихідного коду були доступні під CDDL, але дозволяє розповсюджувати бінарні файли під іншою ліцензією. CDDL створено для полегшення повторного використання. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання **Обов'язки:** - надати копію ліцензії та повідомлення про авторські права до ліцензованого матеріалу; - зміни, внесені до ліцензійного матеріалу, повинні бути задокументовані і надані на умовах CDDL; - зберігати будь-які повідомлення про ліцензування або будь-який описовий текст, який вказує на будь-якого учасника або початкового розробника; - при розповсюдженні ПЗ у компільованій формі, вихідний код має бути доступним на умовах CDDL. Компільована форма може бути випущена на умовах CDDL або будь-якою сумісною з CDDL ліцензією; - для кожної модифікації, яку споживач здійснює, він повинен ідентифікувати себе як модифікатора, включивши відповідне повідомлення у змінені файли. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій; ✗ заборона видаляти або вносити будь-які зміни до будь-яких повідомлень про авторські права, патенти і торгові марки, що містяться у програмі.
Eclipse Public License v.2.0 (EPL)	**Резюме** ***Копілефт ліцензія***, відома своїми положеннями щодо

[40]	комерційного використання програмного забезпечення, з вимогами зберігати ту ж саму ліцензію при розповсюдженні програмного забезпечення. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання **Обов'язки:** - надати копію ліцензії та повідомлення про авторські права до ліцензованого матеріалу; - зміни, внесені до ліцензійного матеріалу, повинні бути задокументовані; - модифікації повинні випускатися під тією ж ліцензією при розповсюдженні ліцензованого матеріалу. Якщо модифікації поширюються у вигляді об'єктного коду, автор повинен вказати, що вихідний код може бути наданий одержувачу за запитом, а також пояснити, як отримати вихідний код. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій.
GNU General Public License v3.0 (GNU GPLv3) [61]	**Резюме** Дозволи цієї ***строгої копілефт ліцензії*** обумовлені наданням повного вихідного коду ліцензованих творів і модифікацій, які включають більші твори, що використовують ліцензований твір, під тією ж ліцензією. Авторське право та повідомлення про ліцензію мають бути збережені. Автори надають явну згоду на передачу патентних прав. **Права:** ✓ Комерційне використання ✓ Розповсюдження ✓ Модифікація ✓ Приватне використання ✓ Патентні права **Обов'язки:** - надати копію ліцензії та повідомлення про авторські права до

	ліцензованого матеріалу; - зміни, внесені до ліцензійного матеріалу, повинні бути задокументовані; - оприлюднення вихідного коду при розповсюдженні ліцензійного матеріалу; - модифікації повинні випускатися під тією ж ліцензією при розповсюдженні ліцензованого матеріалу. У деяких випадках може бути використана подібна або споріднена ліцензія. **Обмеження:** ✗ обмежена відповідальність автора та власника авторських прав; ✗ не включає жодних гарантій.

Аналіз найпопулярніших ліцензій для open-source проектів показує, що багато з них є копілефт ліцензіями. При використанні такого коду у своїх проектах, програмісти, компанії змушені будуть випускати свої продукти під тією самою ліцензією, що й open-source код, або під будь-якою сумісною ліцензією. Цю інформацію необхідно враховувати при розробці пропрієтарного ПЗ, власники яких хочуть захистити свій продукт і максимально обмежити доступ до вихідного коду.

Ліцензування коду, створеного штучним інтелектом, створює правову дилему. Дехто стверджує, що результат має успадковувати ліцензію на програмне забезпечення ШІ, тоді як інші вважають, що для врахування унікальної природи коду, створеного ШІ, може знадобитися нова система ліцензування.

Інструменти, що використовують генеративний ШІ, такі як GitHub Copilot і ChatGPT, тренуються на даних з відкритим вихідним кодом, частина з яких знаходяться під *сильними копілефт ліцензіями (strong copyleft licenses)* – ліцензії, що вимагають надання вихідного коду для всієї похідної роботи, під тією ж ліцензією[39, с.9], –- такими як GPL або AGPL. Це викликало певні дебати про те, чи слід вважати результати роботи генеративного ШІ *похідною роботою* від коду, на якому він

навчався [49]. Якщо ця гіпотеза буде сприйнята правовою спільнотою, то інженери, які використовують генеративний інструмент ШІ у розробці, змушені будуть дотримуватися умов ліцензій на код, на якому тренувалась модель ШІ.

Але це тільки в тому випадку, якщо генеративний результат ШІ містить вираження, що охороняється авторським правом. Досить короткі фрази, які генеративний ШІ часто продукує в контексті розробки програмного забезпечення, не є об'єктами авторського права, а отже, не підпадають ні під авторське право, ні під будь-яку ліцензію з відкритим кодом [49].

Проте виникає і низка інших питань, якщо похідна природа згенерованого коду буде визнана: *Що буде у випадку, коли такі ліцензії несумісні (остаточний продукт містить сніпети коду, які є похідними від різних проектів під несумісними ліцензіями)? Як дізнатися, під впливом яких тренувальних даних був згенерований код і чи містить він ліцензії, які накладають на користувачів обов'язки і обмеження?*

Офіційний сайт Amazon CodeWhisperer, явно містить попередження, що "CodeWhisperer створює новий код на основі того, що він дізнався з коду, на якому навчався, і контексту, який ви надали у вигляді попереднього коду та коментарів. Хоча CodeWhisperer не призначений для відтворення коду, на якому він навчався, можливо, що в рідкісних випадках він буде генерувати код, який точно відповідає певним фрагментам коду в навчальних даних [25]." Крім того, у випадку, коли CodeWhisperer виявить, що його результат відповідає певним навчальним даним з відкритим кодом, вбудований трекер посилань повідомить користувача про це з посиланням на тип ліцензії (наприклад, MIT або Apache) та URL-адресу проекту з відкритим кодом. Таким чином, розробник може оцінити джерело і визначити, чи використовувати згенерований код у його роботі. Також цей інструмент має налаштування,

які забороняють CodeWhisperer робити рекомендації, які містять посилання на відомий ліцензований відкритий код. Корпоративна підписка (*CodeWhisperer Professional*) дозволяє також зберегти таке обмеження на рівні організації [25].

Github зазначає, що їх інструмент “Copilot Chat здатен генерувати новий код, але робить це імовірнісно. Хоча ймовірність того, що він може створити код, який відповідає коду з навчального набору, низька, рекомендація Copilot Chat може містити деякі фрагменти коду, які збігаються з кодом з навчального набору. Copilot Chat використовує фільтри, які блокують збіги з публічним кодом у репозиторіях GitHub [77].” На жаль, жодних посилань на публічні репозиторії для перевірки джерела, які були включені в тренувальний сет і стали основою для згенерованого результату GitHub не надає. Так само, ChatGPT і ReplitAI не надають посилання на open-source проекти, які могли стати основою для згенерованого коду.

Важливість питання ліцензування підтверджує відкритий нещодавно колективний позов проти GitHub Copilot, Microsoft, та OpenAI (*DOE 1 et al v. GitHub, Inc. et al, No. 4:2022-cv-06823*) [38]. Позивачі звинувачують відповідачів у порушенні прав інтелектуальної власності [36].

Ключові аспекти позову стосуються законності GitHub Copilot та моделі OpenAI Codex, що лежить в його основі. У скарзі стверджується, що ці технології були розроблені з використанням відкритого вихідного коду з GitHub репозиторіїв без належних дозволів або дотримання умов ліцензування, встановлених оригінальними розробниками [37].

Рішенням від травня 2023 року суддя відхилив клопотання захисту про відхилення двох важливих скарг позивачів. Перша претензія, яку суддя підтримав, полягала в тому, що здатність Codex відтворювати точні копії коду є порушенням умов ліцензування програмного забезпечення.

Друга стосувалася порушення статті 1202(b) Закону про авторське право в цифрову епоху (the Digital Millennium Copyright Act), стверджуючи, що Copilot і Codex відтворюють захищений авторським правом код без включення необхідної інформації, такої як автор, назва, власник, терміни та умови. Суд відхилив висунуті звинувачення у змові відповідачів (civil conspiracy). Крім того, суддя відхилив, але з дозволом на внесення змін, вимоги позивачів щодо порушення розділу 1202(a) Закону про авторське право в цифрову епоху, втручання в договір, шахрайства, неправдивого зазначення походження, неправомірного збагачення, недобросовісної конкуренції, порушення Політики конфіденційності та Умов надання послуг GitHub, порушення Каліфорнійського закону про конфіденційність споживачів і недбалості. Суддя зазначив, що цим позовам бракує деталей, необхідних за законом для розгляду справи, але визнав їх достатньо правдоподібними, щоб розглянути змінену скаргу, яка надає більше ясності щодо передбачуваної шкоди, спричиненої Copilot та Codex [37].

Ухвала, винесена в травні 2023 року, являє собою неоднозначний результат: суд підтримав ключові вимоги позивачів, відхиливши інші, і дозволив позивачам внести зміни до певних частин їхньої скарги для більшої ясності та деталізації. Справа продовжує розвиватися, і ці рішення є важливим кроком у судовому процесі.

Позов проти Copilot на GitHub є частиною ширшої тенденції суперечок щодо авторських прав, пов’язаних з технологіями генеративного ШІ. Та сама команда юристів, яка представляла позивачів у справі Copilot, подала позов до суду в Каліфорнії проти Midjourney, Stability AI та DeviantArt (Andersen v. Stability AI Ltd. et al, No. 3:23-cv-00201) у січні 2023 року. Ці компанії звинувачують у тому, що вони навчали свої AI-генератори на мільярдах зображень з інтернету без згоди власників авторських прав. Аналогічно, Getty Images ініціювала судовий позов у Великобританії та США проти Stability AI, стверджуючи, що

компанія несанкціоновано використала понад 12 мільйонів фотографій Getty для навчання моделей ШІ. Якщо будь-який із судових позовів проти включення матеріалів, захищених авторським правом, у результати роботи генеративного ШІ або в процес їхньої розробки буде виграно, це може мати серйозний вплив на сферу генеративного ШІ. Успіх цих систем ШІ залежить від широкого і різноманітного збору даних для забезпечення точних і неупереджених результатів [69].

Таким чином, враховуючи описані вище ризики використання програмних артефактів, згенерованих ШІ, розробники повинні використовувати результати генеративного ШІ з обережністю, оскільки що згенеровані артефакти можуть підпадати під обмеження, які містять ліцензії open-source проектів, які були включені в тренувальні дані для моделей штучного інтелекту.

3.2. Проблема несумісності ліцензій та методи вирішення

Сучасне програмне забезпечення зазвичай будується на основі існуючих компонентів, які вказуються як залежності (модулі, пакети) та отримуються через менеджери пакетів або реєстри, такі як NPM для JavaScript, RubyGems для Ruby, PyPI для Python та інші. Ці залежності, як прямі, так і непрямі, можуть створювати складні мережі, які не є повністю прозорими. У таких мережах часто трапляються ситуації, коли ліцензії на програмне забезпечення різних компонентів суперечать одна одній.

Емпіричне дослідження 1,846 GitHub показує, що 72,91% досліджених проектів страждають від ліцензійної несумісності [125, с.1-2]. У березні 2020 року повідомлялося, що 577 148 програмних проектів, які створені за допомогою Ruby on Rails, стикаються з ліцензійними проблемами [95, с.1]

Несумісність ліцензій на програмне забезпечення виникає, коли існують суперечливі умови в декількох ліцензіях на відкрите програмне

забезпечення в межах одного проекту. Це трапляється, коли умови однієї ліцензії ПЗ з відкритим вихідним кодом суперечать умовам іншої, що унеможливлює одночасне дотримання обох ліцензій. Наприклад, ліцензія одного компонента може забороняти субліцензування, в той час як основна ліцензія проекту це дозволяє. Або одна ліцензія може вимагати контакту з автором, а інша – забороняти будь-які контакти. Такі протиріччя створюють юридичні ризики і можуть призвести до порушення законів про авторське право [125, с.1].

Хоча дві ліцензії можуть здаватися сумісними, програмісти повинні переконатися, що вони дійсно сумісні. На перший погляд нешкідливі умови в одній або обох ліцензіях можуть зробити їх несумісними одна з одною. Розповсюдження або модифікація програм, включаючи код під несумісною ліцензією, призведе до порушення авторських прав [74, с.159].

Для того, щоб ***дослідити, чи має конкретний проект проблему несумісності ліцензій***, автор(-и) проекту повинні:

1. визначити, які залежності має проект, а також версії кожної із залежностей. Зазвичай ця інформація міститься у особливому файлі в root директорії. Він генерується автоматично менеджером пакетів для відповідної мови програмування;
2. Визначити ліцензію кожної залежності. Ця інформація часто міститься у метаданих модулю або у файлах типу LICENSE або COPYING у папці, що містить файли модуля;
3. ідентифікувати сніпети (програмні артефакти), які були згенеровані інструментами ШІ та визначити, чи не є згенерований код точною копією даних, на яких модель ШІ тренувалась. У випадку, якщо згенерований код співпадає з кодом одного з open-source проектів, визначити, яку ліцензію містить даний проект;
4. уважно ознайомитися з умовами відповідних ліцензій;

5. отримати (створити, знайти) таблицю сумісності ліцензій або проконсультуйтеся з юристами, щоб визначити, чи є потенційні конфлікти між ліцензіями. Існують також онлайн інструменти, такі як mend.io [93], Joinup, Black Duck від Synopsis та ін., які допомагають визначити, чи сумісні найвідоміші open-source ліцензії між собою. Наприклад, Joinup [72] – платформа від Європейської комісії – містить безкоштовний інструмент, який дозволяє проаналізувати сумісність різних популярних ліцензій. Платформа пропонує обрати під якою ліцензією випускається основний проект і ідентифікувати ліцензії залежностей, а далі виконати аналіз на сумісність цих ліцензій;
6. задокументувати висновки і переконатися, що всі ліцензійні зобов'язання дотримані (наприклад, вказано авторство або наданий доступ до вихідного коду).

Кроки для ***розв'язання конфлікту*** ліцензій:

1. *безпосереднє розв'язання несумісності ліцензій* залежностей та/або ліцензії основного проекту:
   1.1. заміна несумісного модулю на альтернативний, який має сумісну ліцензію.
   1.2. виключити конфліктуючу залежність і створити власний код, для вирішення задачі, яку виконував модуль.
   1.3. звернутися до ліцензіарів, щоб обговорити несумісність і попросити про виняток або альтернативне ліцензування.
   1.4. розглянути можливість повторного ліцензування проекту на умовах, сумісних із залежностями, якщо це можливо.
2. *консультація з експертом права* з питань ліцензування програмного забезпечення, щоб переглянути план усунення

несумісності та проконсультуватися щодо складних питань ліцензування (наприклад, наявність нестандартної, кастомної ліцензії);

3. *впровадження обраного рішення*, оновлення проектної документації та інформування усіх членів команди щодо вимог ліцензування;
4. *встановити процедури для постійного моніторингу сумісності ліцензій*, особливо при оновленні або додаванні нових залежностей.

Таким чином, цій главі ми заглибилися в нюанси ліцензування програмного забезпечення, окресливши ролі ліцензіарів і ліцензіатів та протиставивши рамки open-source ліцензій і пропрієтарних ліцензій. Значну увагу було приділено новим складнощам, що виникають при створенні коду системами ШІ, зокрема правовим проблемам, які виникають, коли ШІ відтворює код ідентичний тому, який міститься в їхніх навчальних даних, що може підпадати під дію open-source ліцензій. Ми проаналізували і систематизували найпоширеніші open-source ліцензії з відкритим кодом, такі як MIT, BSD, Apache License 2.0, MPL, CDDL, EPL і GNU GPL 3.0, виділивши їхні окремі положення та потенціал для конфліктів. Нарешті, ми заглибилися в методологію виявлення та усунення ліцензійної несумісності, підкресливши необхідність пильного управління ліцензуванням програмного забезпечення в проектах, які інтегрують код, згенерований ШІ, щоб забезпечити дотримання правових норм і підтримати цілісність принципів відкритого коду.

РОЗДІЛ 4. МЕТОДОЛОГІЯ ДОТРИМАННЯ ПРАВОВИХ ВИМОГ ТА ВИРІШЕННЯ ПРАВОВИХ КОНФЛІКТІВ ПРИ ВИКОРИСТАННІ АРТЕФАКТІВ СТВОРЕНИХ ШТУЧНИМ ІНТЕЛЕКТОМ

4.1. Огляд та загальна структура методології

Розробка методології дотримання правових вимог і вирішення правових конфліктів у контексті артефактів, згенерованих ШІ вимагає багатогранного підходу, що враховує різні правові сфери, включаючи інтелектуальну власність, контракти і закони про захист прав споживачів. Ось пропонована структура, яку можна адаптувати до різних національних законодавств.

Методологія, яку можна адаптувати для використання у різних національних системах права, має наступну **структуру**:

1. правові дослідження та аналіз чинного законодавства;
2. оцінка ризиків;
3. ведення документації та відстеження;
4. виявлення потенційних правових проблем та система комплаєнсу;
5. розробка механізму вирішення спорів;
6. забезпечення навчання та освіти;
7. впровадження системи моніторингу та стратегії вдосконалення.

Методологія включає кроки та заходи, які можуть виконуватися різними суб'єктами**:**

1. **інженери та юридичні спеціалісти** – виконання методології для запобігання виникненню правових проблем в результаті використання генераторів програмних артефактів під час розробки ПЗ та/або дотримання правових норм чи договірних зобов'язань;
2. **компанії-власники ПЗ** – заходи, які повинні організувати та впровадити на вищому рівні організації для забезпечення розробки пропрієтарного ПЗ в правових межах, уникаючи

порушення прав третіх осіб та для забезпечення інтересів власника пропрієтарного ПЗ.

4.2. Основний зміст методології

4.2.1. Правові дослідження та аналіз чинного законодавства

Огляд ключових юрисдикцій. Незважаючи на те, що метою цієї дисертації є створення юрисдикційно-незалежного фреймворку, для оцінки кожного конкретного випадку використання програмного артефакту, згенерованого ШІ, розуміння правового ландшафту в ключових юрисдикціях може стати джерелом інформації про найкращі світові практики. Для цього необхідно провести порівняльний аналіз правових стандартів, пов’язаних зі штучним інтелектом, у різних юрисдикціях, а також існуючі міжнародні норми та стандарти. Це передбачає визначення країн з передовою практикою або значними правовими прецедентами, які можуть вплинути на глобальні норми та загальноприйняті позиції, щодо правового комплаєнсу та вирішенню конфліктів, пов’язаних зі штучним інтелектом. На основі результатів викладених у вищенаведених розділах, передовими країнами, на практику яких варто звернути увагу, є США, Великобританія та судова практика Європейського Союзу.

Оновлення законодавства. Постійний моніторинг останніх змін у законодавстві, судовій практиці та регуляторних інструкціях, пов’язаних зі штучним інтелектом у різних юрисдикціях, дозволить передбачити зміни, які можуть вплинути на комплаєнс. Підписка на юридичні бази даних, інформаційні бюлетені або створення партнерств з юридичними фірмами, що спеціалізуються на технологічному праві допоможуть впровадити надійну стратегію моніторингу змін.

Вивчення правових документів та судової практики щодо конкретних інструментів ШІ. Для повного розуміння прав та обов’язків

споживача, а також наслідків використання конкретного інструменту-генератора програмних артефактів, необхідно бути обізнаним із Умовами використання або Ліцензійною угодою відповідного інструменту ШІ. Крім того, потрібно моніторити судову практику, відкриті судові справи проти компанії власника, продукту ШІ, який споживач використовує у своїй роботі. На сьогодні, сфера ШІ знаходиться у так званій “сірій зоні”, коли регулятори не встигли прийняти відповідну законодавчу базу, саме тому судові рішення та постанови можуть бути корисними для розуміння тенденцій розвитку правової сфери. Наприклад, вищезгадані позови проти Github Copilot, Open AI, Microsoft, Midjourney, Stability AI, DeviantArt у судах США та Великобританії та ухвали суду у цих справах є цінним джерелом знань для юристів, розробників ШІ та користувачів. Ці справи ілюструють еволюцію правового ландшафту, що оточує технологію ШІ. Вивчаючи результати та аргументи в цих справах, можна отримати уявлення про те, як суди тлумачать і застосовують закони в контексті ШІ. Ці знання мають вирішальне значення не лише для розуміння чинних правових стандартів, а й для прогнозування майбутніх змін у регулюванні.

4.2.2. Оцінка ризиків

Оцінка артефактів штучного інтелекту. Необхідним є впровадження процесу оцінки характеру коду або контенту, створеного ШІ, наприклад, розробка критеріїв для оцінки коду, згенерованого ШІ, на предмет потенційних правових ризиків. Сюди входить визначення джерела і характеру навчальних даних, параметрів моделі ШІ та результатів для виявлення будь-яких потенційних порушень інтелектуальної власності.

Аудит залежностей. Необхідно провести ретельний аудит усіх залежностей у ПЗ, щоб забезпечити відповідність різним ліцензійним вимогам, щоб переконатися, що їхні ліцензії сумісні з передбачуваним

використанням. Це стосується не лише прямих залежностей, а й перехідних залежностей, які можуть мати різні умови ліцензування.

4.2.3. Ведення документація та відстеження

Ведення записів. Створіть систему для детального документування кожного етапу процесу генерації ПА, штучним інтелектом, включаючи вхідні дані, параметрами генерації, процес генерації, інструмент ШІ, який було використано, дату генерації, дату включення програмного артефакту у проект та проведені перевірки згадані у наступному пункті методики. Наявність цієї інформації може мати вирішальне значення для доведення правомірності та добросовісності використання ШІ при розробці ПЗ у випадку судових спорів.

Контроль версій. Використовувати надійні системи контролю версій для управління змінами та ведення історії артефактів, створених ШІ.

4.2.4. Виявлення потенційних правових проблем та система комплаєнсу

Ідентифікація потенційних юридичних конфліктів. Розробити інструмент або контрольний список для швидкого виявлення потенційних правових конфліктів, які можуть виникнути через використання артефактів, створених ШІ, зосередившись на питаннях інтелектуальної власності, потенційних порушеннях угод або правах споживачів.

Контрольний список може включати наступні аспекти для оцінки.

1. Оцінка прав інтелектуальної власності

1.1. *Аналіз авторства*:

1.1.1. Визначити, чи може код, створений ШІ, мати автора.

1.1.2. Визначити автора. Інформація, хто володіє контентом, створеним ШІ, як було вказано у підрозділі про проблеми авторського права, міститься у Умовах використання сервісів генеративного ШІ. Досліджені Умови розглянутих генераторів коду виявили тенденцію, що власником згенерованого контенту є користувачі інструменту ШІ, які надали вхідні дані. Проте не виключено, що інші інструменти генеративного ШІ містять інші правила і визначають власником створеного контенту саму організацію, що пропонує інструмент ШІ користувачам. Крім того, необхідно мати на увазі, що з розвитком регулювання сфери ШІ, правила, щодо визначення власника можуть змінитися у майбутньому. Наприклад контент, згенерований ШІ може бути визнаним суспільним надбанням або ж ШІ може бути визнаним автором згенерованого контенту.

1.1.3. Задокументувати підстави для такого визначення.

1.2. *Авторські права*:

1.2.1. Встановити, кому належать авторські права на програмні артефакти, створені ШІ.

1.2.2. Перевірити, чи містить код матеріали, захищені авторським правом, і характер цих матеріалів (наприклад, з відкритим вихідним кодом, пропрієтарні).

1.2.3. Визначити необхідність атрибуції ШІ. Умови використання сервісів ШІ або ж нормативні правові акти, інструкції у різних юрисдикціях для

суб’ктів різних сфер економіки, що будуть прийняті у майбутньому можуть накладати обов’язок зазначати, що певний контент був згенерований ШІ.

1.2.4. Визначити формат атрибуції. Зазвичай, довільної форми зазначення інструменту ШІ, який згенерував певний контент повинно бути достатньо. Проте деякі сервіси у своїх Умовах можуть визначити формат атрибуції [22].

1.3. *Оригінальність і суттєва схожість*:

1.3.1. Оцінити, чи є згенерований ШІ код оригінальним, чи він значною мірою копіює існуючий код.

1.3.2. Впровадити використання програмних інструменти для порівняння створеного коду з існуючими кодовими базами, щоб виявити схожість. Наприклад, налаштувати генератор ШІ таким чином, щоб він надавав посилання на проекти з відкритим кодом у випадку, коли згенерований результат співпадає з кодом такого open-source проекту. так, вищезгаданий Amazon CodeWhisperer має відповідний функціонал.

1.4. *Похідні роботи*:

1.4.1. Проаналізувати, чи є код, створений за допомогою ШІ, похідним твором.

1.4.2. Визначити, чи були отримані відповідні ліцензії на будь-яку використану оригінальну роботу.

2. Відповідність ліцензійним вимогам

2.1. *Сумісність ліцензій*:

2.1.1. Переглянути всі ліцензії, пов’язані із ПА, згенерованими ШІ і їх залежностями, виявлених у кроках пункту 1.3 та 1.4..

2.1.2. Забезпечити сумісність сумісність ліцензій. Впровадити процес, який гарантує, що весь код, створений ШІ, відповідає ліцензійним вимогам програмного забезпечення, в яке він буде інтегрований. Це може включати автоматизовані інструменти, які можуть виявляти ліцензійні конфлікти або несумісність. Детальна інформація про несумісність ліцензій та методика вирішення конфліктів ліцензій зазначена у підрозділі 3.2.

2.1.3. Для проектів з відкритим вихідним кодом створити керівні принципи для розробників при включенні коду, згенерованого ШІ, у проект, гарантуючи, що внесок не порушує основних принципів відповідних ліцензій з відкритим вихідним кодом.

2.2. *Дотримання ліцензійних вимог*:

2.2.1. Вимоги до авторства. Перевірити, чи є в ліцензії вимоги щодо зазначення авторства та чи були вони дотримані.

2.2.2. Визначити будь-які обмеження, що накладаються ліцензією (наприклад, некомерційне використання, положення про спільне використання) та обов’язки, що накладаються на користувача ліцензійного ПЗ.

2.2.3. Забезпечити дотримання відповідних умов.

3. Договірні зобов’язання

3.1. *Користувацькі угоди*:

3.1.1. Переконатися, що ШІ та його згенеровані артефакти не суперечать умовам користувацьких угод або умовам надання послуг.

3.1.2. Переконатися, що всі договірні обов'язки щодо роботи ШІ і його використання при розробці ПЗ, дотримані.

3.2. *Угоди з третіми сторонами*: Підтвердити дотримання угод з третіми особами, таких як контракти з постачальниками даних, умови використання API тощо.

4. Законодавство про захист прав споживачів

4.1. *Розкриття інформації*:

4.1.1. Перевірити, чи існує законодавча вимога розкривати інформацію про використання ШІпри розробці ПЗ користувачам або споживачам.

4.1.2. Переконатися, що маркетингові матеріали точно відображають роль ШІ в розробці продукту.

4.2. *Відповідальність за продукцію чи надані послуги*:

4.2.1. Оцінити потенційну відповідальність, якщо згенерований ШІ код вийде з ладу і спричинить шкоду або збитки.

4.2.2. Визначити адекватність, всебічність та достатність попереджень і застережень, пов'язаних із ПА, згенерованими ШІ .

5. Захист даних та конфіденційність

5.1. *Легальність джерела даних*:

5.1.1. Переконатися, що дані, які використовуються для навчання ШІ, були отримані законним шляхом і відповідають законам про захист даних.

5.1.2. Переконатися, що персональні дані, якщо такі є, оброблялися відповідно до національних законів про конфіденційність даних та міжнародних регламентів таких як Загальний регламент про захист даних (General Data Protection Regulation) [52] тощо.

6. Етичні міркування

6.1. Взяти до уваги етичні міркування в процес оцінки, щоб переконатися, що використання ШІ в розробці програмного забезпечення відповідає суспільним цінностям і не призводить до несправедливих або шкідливих результатів.

6.2. *Упередженість і справедливість*:

6.2.1. З'ясувати, чи проводилася раніше оцінка коду, створеного ШІ на предмет неупередженості та справедливості.

6.2.2. Оцінка та аналіз:

6.2.2.1. Визначити ключові показники ефективності. Оцінити, що означає справедливість для розробленого продукту, яка може включати такі фактори, як рівні можливості, відсутність упереджень або демографічний паритет.

6.2.2.2. Визначити чіткі цілі для аналізу, наприклад, виявлення потенційних упереджень у процесах прийняття рішень або забезпечення рівного ставлення до різних груп користувачів.

6.2.2.3. Переглянути набори даних, які використовуються для навчання ШІ. Звернути увагу на репрезентативність, історичні упередження та аномалії.

6.2.2.4. Оцінити модель ШІ, на основі якого працює використовуваний при розробці ПЗ інструмент, на предмет прозорості та інтерпретованості [108].

6.2.3. Тестування програмного продукту для порівняння результатів рішень ШІ для різних груп у контрольованих умовах.

6.2.4. Запровадити заходи для вирішення будь-яких виявлених етичних проблем.

Запропонований список призначений для використання як динамічний інструмент, який регулярно оновлюється, щоб відображати зміни в законодавстві та практиці. Він вимагає участі юристів, розробників і фахівців з комплаєнсу. Мета контрольного списку – не лише виявити потенційні конфлікти, а й створити основу для їх вирішення, мінімізуючи юридичні ризики, пов’язані з артефактами, згенерованими штучним інтелектом.

4.2.5. Розробка механізму вирішення спорів

Розробити механізм вирішення спорів, які можуть виникнути через використання ПА, створених ШІ. Розроблені процедури можуть включати як внутрішні механізми врегулювання як переговори, медіація, так і зовнішні, такі як арбітраж або судовий розгляд, і повинні бути достатньо гнучкими, щоб адаптуватися до різних юрисдикцій.

Розробити план дій у разі подання судового позову, включно зі стратегіями початкового реагування та потенційними варіантами врегулювання.

4.2.6. Забезпечення навчання та освіти

Навчання розробників. Організовувати регулярні тренінги для розробників або самонавчання з правових аспектів, пов'язаних з кодом, створеним штучним інтелектом. Це може включати розуміння закону про авторське право, відмінностей у ліцензуванні та правових наслідків включення коду, створеного ШІ.

Навчання юридичної команди. Переконатися, що in house юристи та/або аутсорс юридичні консультанти в курсі останніх досягнень у галузі технологій ШІ і розуміють технічні аспекти, які можуть мати юридичні наслідки. Впевнитись, що згадані спеціалісти також мають чіткі юридичні стратегії, досвід з розв'язання правових конфліктів пов'язаних з розробкою ПЗ.

4.2.7. Впровадження системи моніторингу та стратегії вдосконалення

Цикл зворотного зв'язку. Створити механізми для отримання зворотного зв'язку від розробників, користувачів та юристів з метою постійного вдосконалення системи комплаєнсу. Це може включати регулярні опитування, зустрічі або використання спеціального інструменту зворотного зв'язку.

Перегляд політики. Скласти графік перегляду політик і процедур, щоб вони відповідали останнім правовим і технологічним змінам.

4.2.8. Етапи впровадження

1. Розробити комплексний контрольний список, що включає вищезгадані елементи, адаптований під конкретні інтереси та вимоги суб’єкта.
2. Навчити команду проводити первинну оцінку артефактів, згенерованих ШІ.
3. Залучити експертів у галузі права для вдосконалення системи та вирішення складних питань.
4. Протестувати створений фреймворк на невеликому проекті та вдосконалювати його на основі отриманого досвіду.
5. Впровадити систему компла’нсу в усіх командах розробників, створивши відповідні структури підтримки.
6. Постійний моніторинг та адаптація фреймворку на основі правових та технологічних змін.

Таким чином, дотримуючись цієї концепції, розробники і компанії можуть забезпечити дотримання прав інтелектуальної власності, дотримання ліцензійних вимог і захист власних законних інтересів під час використання коду, створеного ШІ, як у відкритому, так і в пропрієтарному програмному середовищі.

РОЗДІЛ 5. ПРОЕКТУВАННЯ, РОЗРОБКА ТА ТЕСТУВАННЯ КОМПЛАЄНС ПЛАТФОРМИ

5.1. Мета розробки платформи

Поява ШІ у сфері розробки програмного забезпечення призвела до виникнення безлічі правових ускладнень, що потребують чіткого розуміння та навігаційних інструментів для дотримання вимог законодавства та найкращих практик. Web-додаток “AI LawGuide” [14] розроблений з метою задоволення цієї критичної потреби, забезпечуючи комплексну інтерактивну платформу, яка спрощує тонкощі правових та етичних міркувань для коду, створеного штучним інтелектом.

Основними цілями платформи є:

1. *Усунення прогалин у знаннях*. AI Law Guide покликаний заповнити прогалину між технологіями ШІ, що стрімко розвиваються, та правовими рамками, які їх регулюють. Оскільки такі інструменти ШІ, як ChatGPT, GitHub Copilot і Amazon CodeWhisperer, дедалі більше стають невід’ємною частиною розробки програмного забезпечення, потреба в доступному розумінні правових наслідків стає першочерговою. Ця платформа слугує каналом для цих знань, роблячи їх доступними для широкого кола користувачів, включаючи розробників, юристів та власників продуктів.
2. *Персоналізовані юридичні рекомендації*. Визнаючи різноманітність використання ШІ в різних юрисдикціях і сферах застосування, платформа пропонує персоналізовані юридичні рекомендації. Користувачі можуть ввести конкретні критерії, пов’язані з їхнім випадком використання - наприклад, тип використовуваного інструменту ШІ, характер програмного забезпечення (пропрієтарне або з відкритим вихідним кодом),

юрисдикцію та характер питання, яке цікавить користувача. Такий індивідуальний підхід гарантує релевантні та точні поради, підвищуючи таким чином дотримання правових норм і знижуючи ризик ненавмисних порушень.

3. *Розширення можливостей користувачів через інформування*. Надаючи актуальну інформацію про правові зміни, найкращі практики та новини у сфері штучного інтелекту, програма дає користувачам можливість приймати обґрунтовані рішення. Він демістифікує складний юридичний жаргон і подає інформацію в доступному форматі, дозволяючи користувачам проактивно управляти юридичними ризиками, пов’язаними з кодом, створеним за допомогою ШІ.
4. *Сприяння правовому та етичному використанню ШІ*. AI Law Guide – це не лише юридичний інструмент, але й етичний компас. Він сприяє розробці та використанню ШІ у спосіб, що відповідає законодавству та є етично обґрунтованим. Завдяки рекомендаціям для аналізу неупередженості та справедливості, платформа виступає за відповідальне використання ШІ, підкреслюючи важливість розробки технологій, які є рівноправними та справедливими.
5. *Сприяння глобальному дотриманню правових норм*. Завдяки підходу, заснованому на діагностиці юрисдикції, AI Law Guide враховує глобальний характер розробки програмного забезпечення. Він розрахований на широку аудиторію, оскільки охоплює різні міжнародні закони та правила, що робить його цінним інструментом як для глобальних компаній, так і для індивідуальних розробників.
6. *Постійна еволюція*. Визнаючи динамічний характер як технології ШІ, так і нормативно-правової бази, платформа

розроблена з урахуванням їх постійної еволюції. Платформа регулярно оновлюється, щоб відображати останні юридичні прецеденти, технологічні досягнення та відгуки користувачів, забезпечуючи його актуальність та ефективність у постійно мінливому середовищі.

Таким чином, розробка AI Law Guide є відповіддю на нагальну потребу в ясності та визначеності правових аспектів застосування штучного інтелекту при розробці програмного забезпечення. Він є свідченням потенціалу технологій у спрощенні складних правових ландшафтів, сприяючи формуванню більш поінформованої, законослухняної та етичної спільноти розробників штучного інтелекту.

5.2. Загальний огляд платформи

AI Law Guide Platform – це новаторське веб-рішення, призначене для вирішення складних правових проблем, що виникають у зв’язку з використанням ШІ при розробці ПЗ. Комплексна платформа слугує важливим ресурсом для розробників, юристів та власників продуктів, надаючи індивідуальні рекомендації та актуальну інформацію про юридичні нюанси коду, створеного за допомогою ШІ. У цьому підрозділі описано ключові особливості та функціональні можливості AI Law Guide, що ілюструє його роль як важливого інструменту для навігації в правовому полі застосування ШІ.

Аутентифікація. Користувачі мають можливість створити власний аккаунт і увійти у додаток за допомогою електронної пошти та паролю. Створити профіль користувачі можуть на сторінці реєстрації, а увійти на сторінці логінізації відповідно. Крім того, користувачі мають альтернативний варіант аутентифікації – за допомогою Google OAuth 2.0. При натисканні відповідної кнопки на сайті, користувача буде переадресований на сторінку авторизації за допомогою Google профілю.

Після успішної аутентифікації, користувача буде переадресовано у додаток на головну сторінку платформи.

У випадку, коли користувач обрав Google OAuth для реєстрації, він має змогу додатково створити пароль на сторінці профілю. Таким чином користувач матиме змогу використовувати як локальну авторизацію за допомогою паролю та електронної пошти, так і соціальний провайдер авторизації Google OAuth.

У випадку, коли користувач зареєструвався за допомогою електронної пошти та паролю, а наступного разу увійшов у додаток за допомогою Google OAuth і електронна ношта у базі даних платформи співпадає з електронною поштою профілю Google, система автоматично з'єднає соціальний профіль Google з профілем користувача у системі платформи.

Авторизація. Користувачі поділено на дві основні категорії на основі ролей. На сьогодні платформа включає звичайних користувачів, які мають роль BASIC, та адмін користувачів з ADMIN роллю відповідно. Сторінки для реєстрації профілю та входу у додаток розроблені виключно для звичайних користувачів. Такі користувачі мають доступ до основного функціоналу, доступ до свого профілю та збережених даних. Звичайні користувачі не мають доступу до даних інших користувачів та не мають змоги модифікувати основний контент платформи. Адмін користувачі мають змогу переглядати список користувачів, видаляти користувачів, а також модифікувати, оновлювати контент платформи.

Основний функціонал платформи. Платформа складається з таких основних сторінок:

1. *Dashboard*. Основна сторінка платформи, яка поділена на три основні частини:
    1.1. *Новини*. Ця частина призначена для відображення основних новин у сфері технологій та ШІ. Користувач

може обрати специфічну категорію для відображення відповідник новин. Сайт пропонує такі основні категорії: All, AI, ChatGPT, OpenAI, Github Copilot, Programming.

1.2. *Закладки*. Користувачі можуть додати відповідні новини у закладки. Останні чотири закладки будуть відображені у окремій секції цієї сторінки.

1.3. *Юридична допомога*. На основній сторінці користувачі також можуть написати звернення для отримання юридичної консультації від спеціаліста. Після введення особистих даних, опису основної суті питання, користувачі надсилають запит, після чого очікують зворотного зв’язку від спеціаліста.

2. *Compliance Tool*. Сторінка, яка реалізує основну функції платформи – автоматична генерація рекомендацій для користувачів з використання ПА, згенерованих ШІ.

2.1. Сторінка включає панель критеріїв, які є визначальними для формування рекомендацій. Користувач має змогу обрати інструмент ШІ, який використовується для генерування програмних артефактів – ChatGPT, Github Copilot, Amazon Codewisperrer, Replit AI; тип ПЗ, який знаходиться у розробці – пропрієтарне чи open source; юрисдикцію (на сьогодні, лише США, Великобританія і Україна доступні); тип проблеми, яка цікавить найбільше – авторське право та право власності, ліцензування, упередженість тощо.

2.2. Користувач надсилає запит на сервер, який повертає згенеровані рекомендації. Рекомендації згруповані за певним типом питання, на які необхідно звернути увагу, compliance checklist, який користувач може скопіювати,

а також список корисних посилань для подальшого самостійного дослідження.

2.3. Користувач має змогу зберегти згенеровані рекомендації зазначивши ім’я та опис. Після збереження користувач автоматично переадресований на сторінку огляду збережених рекомендацій. Список збережених рекомендацій доступний на окремій сторінці.

3. *Закладки*. Сторінка керування збереженими новинами, де користувач має змогу переглянути список закладок, перейти за посиланням збереженої новини, а також видалити непотрібні закладки.
4. *Збережені рекомендації*. Сторінка, яка відображає список збережених рекомендацій у вигляді таблиці, яка містить основну інформацію – назва, опис, дата збереження, обрані критерії для генерації рекомендацій.
5. *Профіль користувача*. Платформа вимагає лише ті особисті дані, які необхідні та достатні для функціонування платформи і надання доступу до функціоналу окремим користувачам – електронну пошту. На сторінці профілю, користувач може переглянути дані, які зберігаються про нього у базі даних платформи. Крім того, на цій сторінці користувач має змогу змінити пароль, а також створити пароль у випадку, коли для створення профілю була використана Google OAuth аутентифікація.

Платформа спроектована таким чином, щоб у майбутньому розширити функціонал додавши сторінку звернень за консультаціями та перегляд відповіді спеціаліста, сторінку зареєстрованих юридичних консультантів та можливість звернутися до обраного консультанта, форум

користувачів для обговорення гострих питань, які виникають на перетині технологій та права.

5.3. Вибір інструментів і технологій для проектування та розробки платформи

Під час розробки AI Law Guide було ретельно відібрано різноманітні технології та інструменти для оптимізації функціональності, зручності користування та масштабованості. Кожен компонент технологічного стека був обраний з огляду на його специфічні можливості, відповідність цілям проекту та галузевим стандартам.

5.3.1. Планування, дизайн архітектури та UI/UX дизайн

1. Платформу *Lucidchart* [78] було використано для створення ERD-діаграм, діаграми розгортання платформи, діаграми послідовності та діаграми компонентів. Цей інструмент був обраний за його інтуїтивно зрозумілий інтерфейс і надійні функції, які спрощують складне моделювання даних і візуалізацію процесів.

2. *Figma* [48] обрано для розробки UI/UX дизайну завдяки високій точності проектування, зрозумілості на зручності юзер інтерфейсу, які мають вирішальне значення для створення привабливого та зручного дизайну додатку.

5.3.2. Інструменти розробки

1. Редактору коду *Visual Studio Code* [119] віддається перевага за широку підтримку різних мов програмування, інтеграцію з безліччю розширень та комплексні інструменти налагодження.

2. *Prisma Studio* [98] – це потужний інструмент керування базами даних, який надає візуальний інтерфейс для взаємодії з БД. Він пропонує більш інтуїтивний спосіб перегляду та редагування даних у порівнянні з

традиційними інтерфейсами БД, що робить управління ними більш доступним. Він спрощує створення, читання, оновлення та видалення даних. Зручний інтерфейс дозволяє швидко вносити зміни та робити запити, що сприяє швидкій розробці та тестуванню взаємодії з БД.

3. *React DevTools* [101] – розширення для браузера, яке надає візуальний інтерфейс для перевірки та налагодження дерев React-компонентів. Воно дозволяє переглядати ієрархію React-компонентів, що відображаються на сторінці, разом з їхніми поточними пропсами та станом.

4. *Redux DevTools* [102] – це важливий інструмент для керування станом програми в Redux. Він дозволяє відстежувати зміни стану в часі, надаючи історичний погляд на стан. Цей інструмент надає інформацію про відправлені дії (actions) та зміни стану, дозволяючи проаналізувати, як дії впливають на стан програми. Цей аналіз має вирішальне значення для забезпечення того, щоб логіка управління глобальним станом додатку функціонувала належним чином.

5. *Postman* [96] використовувалася для налагодження та тестування ендпоінтів створеного REST API. Платформа надає зручний інтерфейс та широкий функціонал для надсилання запитів на сервер.

6. *Git* [56] було обрано для контролю версій. На відміну від централізованих систем контролю версій, Git є розподіленим. Це означає, що кожен учасник має повну історію кодової бази, що дозволяє виконувати роботу та коміти в автономному режимі та об’єднувати їх пізніше. Такий децентралізований підхід підвищує гнучкість і стійкість до втрати даних.

7. *Github* [59] слугує центральним місцем для зберігання коду. Він особливо популярний для проектів з відкритим кодом, пропонуючи Wiki та трекер проблем, які полегшують отримання більш детальної документації та зворотного зв’язку. GitHub було обрано безпосередньо для

зберігання коду платформи через його інтеграцію з Netlify та Render.com для безперервного розгортання (Continuos Deployment). Ця інтеграція автоматизує процес розгортання кожного разу, коли зміни вносяться до обраної гілки в репозиторії. Крім того, GitHub пропонує автоматичну перевірку для кожного pull request, що є важливою функцією для підтримки якості коду та гарантування того, що новий код не містить помилок або регресій. Цей автоматизований процес допомагає підтримувати цілісність і надійність кодової бази.

5.3.3. Front-end стек

1. *React*. Ключовою особливістю React є його компонентна архітектура, яка дозволяє створювати багаторазові та підтримувані компоненти інтерфейсу. Цей модульний підхід сприяє ефективній розробці, особливо у великих додатках, де компоненти можуть бути розроблені незалежно, а потім легко інтегровані в додаток.

Крім того, React має велику спільноту розробників та багату екосистему інструментів та бібліотек, таких як Redux для управління станом. Ця розгалужена мережа підтримки пропонує безліч ресурсів, навчальних посібників та сторонніх пакетів, які можуть бути корисними у вирішенні проблем розробки.

2. *TypeScript* – мова програмування, доповнення до JavaScript, що вводить статичну типізацію, яка дозволяє виявляти помилки під час компіляції, а не під час виконання. Таке раннє виявлення помилок і дефектів робить процес розробки більш ефективним і зменшує ймовірність помилок.

Надійна система типізації TypeScript призводить до більш чистого, передбачуваного коду, який легше налагоджувати та підтримувати. Вона забезпечує чітку домовленість про те, які типи даних використовуються,

зменшуючи неоднозначність у кодових базах, особливо у великих проектах.

Подібно до React, TypeScript також користується потужною підтримкою спільноти та екосистеми. Він широко прийнятий в індустрії, забезпечуючи розробникам широкий спектр ресурсів та інструментів.

3. *Redux, Redux-Thunk, Redux Persist* – тріада, яка сприяє ефективному управлінню станами у всьому додатку, з Redux-Thunk, що обробляє асинхронні дії, та Redux Persist, що забезпечує сталість стану додатку.

4. *Material UI Kit* надає повний набір попередньо розроблених компонентів, що дозволяє створювати послідовний та сучасний дизайн користувацького інтерфейсу. Використання готових компонентів відповідає одному з основних принципів розробки програмного забезпечення – перевикористання коду, використання готових модулів – що пришвидшує час розробки.

5. *Vite Bundler* пропонує швидкий час та забезпечує ефективність збірки, що має вирішальне значення для додатків, орієнтованих на продуктивність.

Vite відомий як сучасний фронтенд-інструмент збірки, розроблений для покращення досвіду розробки. Однією з його основних переваг є використання нативних модулів ES (native ESM), що дозволяє йому завантажувати код через HTTP, минаючи необхідність бандлінгу під час розробки. Такий підхід дозволяє значно скоротити час запуску та оновлення порівняно з традиційними пакувальниками, такими як Webpack. Негайний цикл зворотного зв’язку, що забезпечується функцією гарячої заміни модулів (HMR) [123], значно покращує досвід розробника, роблячи ітерації над змінами коду простішими та ефективнішими. Така швидкість та ефективність особливо корисні для складних, керованих

даними додатків, таких як AI Law Guide, де очікуються часті зміни та оновлення інтерфейсу.

Крім того, Vite підтримує можливості, які оптимізують роботу програми, такі як lazy loading, tree shaking за замовчуванням [46]. Це відповідає меті створення програми, орієнтованої на продуктивність. Крім того, проста конфігурація Vite, стандартна підтримка TypeScript і сумісність з різними фреймворками роблять його адаптивним і перспективним вибором.

5.3.4. Back-End стек

1. *Node.js та Express.js.* Вони складають основу серверного середовища, обрані за масштабованість, продуктивність та широку підтримку спільноти.

Node.js – це потужне та універсальне середовище виконання JavaScript, яке має високу масштабованість та ефективність, що робить його популярним вибором для сучасних веб-додатків. Його неблокуюча архітектура, керована подіями, дозволяє обробляти кілька паралельних запитів, що є важливою особливістю для платформи з великим обсягом даних [90]. Така масштабованість необхідна для того, щоб пристосуватися до різних рівнів трафіку і складних взаємодій з даними без шкоди для продуктивності. Крім того, використання JavaScript як на стороні клієнта, так і на стороні сервера спрощує розробку, забезпечуючи більш уніфікований та ефективний процес кодування. Така єдність мови в стеку не тільки спрощує процес розробки, але й зменшує когнітивне навантаження на розробників, сприяючи створенню більш згуртованого та продуктивного середовища розробки.

Express.js, мінімальний і гнучкий фреймворк для веб-додатків Node.js. Він доповнює Node.js, надаючи надійний набір функцій для веб- і мобільних додатків. Він спрощує завдання створення веб-серверів,

оскільки його модулі інтуїтивно зрозуміло вирішують всі завдання – від маршрутизації до обробки помилок [44]. Ця модульність дозволяє структурувати платформу AI Law Guide у чистий і зручний спосіб, що полегшує оновлення та масштабування. Широка підтримка фреймворку спільнотою та велика кількість плагінів і розширень додатково гарантують, що платформа може відповідати новим веб-стандартам і практикам, що розвиваються, зміцнюючи її надійність і майбутню життєздатність у швидкозмінній сфері технологій.

2. *Prisma ORM* – інструмент об'єктно-реляційного відображення, який значно спрощує робочі процеси з базами даних у додатках Node.js. Його основна перевага полягає в здатності забезпечувати надійну безпеку типів, яка легко інтегрується з TypeScript, мовою, що використовується для розробки платформи [99]. Ця інтеграція гарантує, що моделі даних у додатку узгоджуються зі схемою БД, зменшуючи ймовірність помилок під час виконання, підвищуючи якість коду. Функція безпеки типів Prisma забезпечує більш надійний і зручний в обслуговуванні код, що є життєво важливим для такої складних платформ, які створюється з перспективою розширення функціоналу і еволюції.

Крім того, Prisma ORM пропонує інтуїтивно зрозумілий і зручний інтерфейс запитів, що спрощує складні операції з базами даних. Здатність ORM генерувати адаптовані запити на основі моделей програми забезпечує оптимальну продуктивність і запобігає поширеним помилкам, пов'язаним із взаємодією з базами даних, таким як проблеми з N+1 запитами [68].

Додатково, система міграції Prisma допомагає розвивати схему бази даних контрольованим і передбачуваним чином, що має вирішальне значення для підтримки цілісності та масштабованості платформи. Вичерпна документація та потужна підтримка спільноти ще більше

підвищують її привабливість, гарантуючи, що розробники мають ресурси та рекомендації, необхідні для ефективного використання її можливостей.

3. *JWT* та *bcrypt*. JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді JSON-об'єкта. Ця інформація може бути перевірена і їй можна довіряти, оскільки вона підписана цифровим підписом. JWT можуть бути підписані за допомогою секретного ключа (за допомогою алгоритму HMAC) або пари публічних/приватних ключів за допомогою RSA або ECDSA [73]. У контексті AI LawGuide, JWT слугує ефективним методом безпечної аутентифікації користувачів на основі токенів. Коли користувач входить в систему, система генерує JWT, який потім надсилається клієнту. Цей токен використовується в наступних запитах до сервера, що дозволяє серверу підтвердити особу користувача. Цей метод є безстатусним, тобто серверу не потрібно зберігати записи про кожну сесію користувача, тим самим зменшуючи навантаження на сервер і покращуючи масштабованість. Крім того, JWT можна легко інтегрувати з мікросервісами, що робить їх ідеальним вибором для сучасних розподілених додатків. Можливість кодування ролей і дозволів всередині токена також забезпечує тонкий контроль доступу, підвищуючи безпеку програми.

bcrypt, з іншого боку, використовується для безпечного хешування паролів. У будь-якому додатку, де користувачі повинні реєструватися та аутентифікуватися, безпечне зберігання паролів є надзвичайно важливим. bcrypt надає потужний спосіб хешування паролів перед тим, як зберігати їх у БД. Його ключовою особливістю є адаптивність до зростаючих обчислювальних потужностей: bcrypt може включати "робочий фактор" (work factor) або "сіль" (salt) - додатковий фрагмент даних, який додається до пароля перед хешуванням. Це робить процес хешування повільнішим, тим самим значно зменшуючи ймовірність атак грубої сили (brute-force

attacks). Крім того, bcrypt стійкий до атак за допомогою райдужної таблиці [100; 122] завдяки своєму хешуванню з використанням солі. У контексті AI Law Guide використання bcrypt означає, що навіть якщо база даних буде скомпрометована, зловмисники не зможуть легко отримати паролі користувачів, що додає додатковий рівень безпеки до даних користувачів.

4. *PostgreSQL*. Потужна реляційна база даних з відкритим вихідним кодом, відома своєю надійністю, функціональною стійкістю та продуктивністю, що робить її чудовим вибором для складних додатків. Однією з ключових переваг PostgreSQL є потужна підтримка розширених типів даних і складних запитів. Це дозволяє ефективно обробляти та запитувати складні та різноманітні типи даних, характерні для додатків юридичних технологій, такі як структуровані дані, текстові документи та метадані. Здатність PostgreSQL ефективно виконувати складні запити має важливе значення для надання швидких і точних результатів пошуку та функцій аналізу даних в AI Law Guide.

Крім того, відповідність PostgreSQL властивостям ACID (атомарність, узгодженість, ізоляція, довговічність) [20] гарантує надійну обробку транзакцій, що є критично важливим для підтримки цілісності та узгодженості даних, особливо в юридичних додатках, де точність даних має першочергове значення.

Крім того, PostgreSQL пропонує надійні функції безпеки, включаючи надійні механізми контролю доступу та підтримку SSL, що забезпечує безпечне зберігання даних і передачу інформації [109]. Масштабованість бази даних і здатність обробляти великі обсяги даних також роблять її перспективним вибором, здатним задовольнити зростаючі потреби в даних і базу користувачів AI Law Guide.

5.3.5. Інші сервіси та API

1. *Google OAuth 2.0* надає надійний і безпечний метод аутентифікації користувачів за допомогою облікових записів Google.

OAuth 2.0 підвищує безпеку, дозволяючи додаткам отримувати доступ до захищених ресурсів без передачі конфіденційних облікових даних, таких як імена користувачів та паролі, стороннім клієнтам. Замість цього доступ надається через обмін токенами, що знижує ризик крадіжки облікових даних або викриття [118].

Автентифікація за допомогою OAuth 2.0 покращує користувацький досвід. Користувачі можуть використовувати свої існуючі облікові записи для автентифікації та авторизації сторонніх додатків, зменшуючи перешкоди та сприяючи кращому впровадженню, сприйняттю нової платформи.

Крім того, як широко прийнятий галузевий стандарт, OAuth 2.0 пропонує широку підтримку на різних платформах, фреймворках і мовах програмування, полегшуючи інтеграцію різних додатків і сервісів в масштабованому вигляді.

2. *NewsAPI* [89]. Ефективне рішення для отримання поточних новин з різних джерел, пов’язаних зі штучним інтелектом, надаючи вичерпну та актуальну інформацію.

3. *GoDaddy* обрано для придбання домену завдяки надійності та широкому спектру послуг.

5.3.6. Розгортання та хостинг

1. *Netlify*. Netlify є популярним вибором для фронтенд-хостингу та безперервного розгортання завдяки кільком ключовим особливостям:

1. Netlify інтегрується безпосередньо з репозиторіями Git, забезпечуючи безперервне розгортання. Це означає, що будь-який новий комміт, автоматично запускає процес збірки та

розгортання на Netlify, синхронізуючи розгортання з останньою версією коду без необхідності складної конфігурації [31].

2. Безпека є важливим аспектом будь-якого веб-додатку, і Netlify підвищує її, пропонуючи безкоштовні SSL-сертифікати через Let’s Encrypt. Це спрощує процес налаштування HTTPS для сайтів, гарантуючи, що дані, які передаються між сервером та користувачами, зашифровані [66].
3. Netlify надає послуги з управління DNS, що може бути додатковою перевагою для більш ефективного управління конфігураціями, пов’язаними з доменами [87].

2. *Render.com* обраний для хостингу серверної частини та баз даних, особливо завдяки можливостям масштабування. Платформа має наступні переваги:

1. Render.com підтримує широкий спектр додатків і сервісів, включаючи веб-додатки, API та бази даних PostgreSQL. Його гнучкість дозволяє ефективно розгортати та керувати різноманітними внутрішніми сервісами [103].
2. Розгортання бази даних PostgreSQL на Render.com є простим і швидким. Використання SSD-накопичувачів для баз даних забезпечує вищу продуктивність і більшу надійність у порівнянні з традиційними HDD. Крім того, Render.com забезпечує щоденне резервне копіювання баз даних, що є важливим для забезпечення цілісності даних та їх відновлення [32].
3. Render.com обслуговує контент через глобальну мережу доставки контенту (CDN), що забезпечує швидке завантаження та захист від DDoS-атак [33]. У поєднанні з безкоштовним TLS сертифікатами [50], це налаштування

забезпечує як оптимізацію продуктивності, так і безпеку для внутрішніх служб програми.

4. Хмарна платформа Render.com розроблена таким чином, щоб бути масштабованою, що дозволяє регулювати ресурси в міру зростання попиту. Така масштабованість у поєднанні з оплатою за використання означає, що розробник можете ефективно управляти витратами, гарантуючи, що платформа може впоратися зі зростаючими навантаженнями без шкоди для продуктивності [107].

Отже, Netlify та Render.com забезпечують надійну, безпечну та масштабовану інфраструктуру для платформи AI Law Guide. Безперервне розгортання Netlify, безкоштовні можливості управління SSL та DNS добре підходять для потреб зовнішнього хостингу, тоді як гнучкі хостингові рішення Render.com, ефективне управління базами даних та можливості масштабування роблять його ідеальним вибором для потреб хостингу.

Кожна технологія та інструмент були обрані для створення цілісної, ефективної та масштабованої програми, здатної задовольнити складні потреби платформи AI Law Guide. Таке поєднання передових технологій гарантує, що платформа не тільки надійна і безпечна, але й готова до майбутніх розробок і вдосконалень у сфері штучного інтелекту та юридичних технологій, що стрімко розвиваються.

5.4. Архітектурні рішення та імплементація

5.4.1. Архітектура розгортання

Архітектура платформи розроблена для забезпечення високої доступності, відмовостійкості та безперервного **розгортання**, як показано на представленій *UML діаграмі розгортання* (*див. Додаток 1, Діаграма 1*).

Діаграма інкапсулює конфігурацію розгортання програмної системи, показуючи розподіл артефактів по вузлах.

Розгортання фронтенду

Netlify Application Server розміщує front-end частину платформи, фактично, production артефакти, згенеровані інструментом Vite після процесу бандлігу вихідного коду React додатку – файли у форматі HTML, CSS, JS та медіа файли, які додаток використовує. Розгортання на Netlify дозволяє використовувати глобальну мережу доставки контенту (CDN) для швидкої доставки контенту та автоматичної конфігурації SSL, що підвищує безпеку шляхом шифрування даних під час передачі.

ПК/ноутбук – браузер. Користувач взаємодіє з додатком через браузер на своєму ПК або ноутбуці. Пакет React SPA доставляється через HTTPS, що забезпечує безпечний та зашифрований зв’язок.

Розгортання бекенду

На *Render.com Application Server* знаходиться Node.js додаток, який слугує бекендом для платформи, імплементації REST API архітектури. Цей сервер взаємодіє з фронтендом через HTTPS.

На *Render.com Database Server* розміщена база даних PostgreSQL, яка зберігає дані платформи, керована Prisma ORM. Взаємодія з базою даних відбувається через TCP/IP, що забезпечує надійну передачу даних.

Continuous deployment

GitHub Server діє як центральний репозиторій для кодової бази, розділений на front-end та back-end репозиторії відповідно. Безперервне розгортання полегшується завдяки GitHub інтеграції з хостинговими провайдерами, які автоматично розгортають останню версію коду на відповідні сервери після оновлення коду на головній гілці репозиторію.

Передача управління DNS від GoDaddy до Netlify

Передача управління DNS від GoDaddy до Netlify була стратегічним рішенням для консолідації процесів управління доменами та розгортання. Служби управління DNS від Netlify пропонують бездоганну інтеграцію з хостинговим середовищем, що дозволяє керувати як розгортанням, так і налаштуваннями DNS в одному місці. Ця інтеграція спрощує процес налаштування користувацьких доменів і SSL-сертифікатів, тим самим зменшуючи складність і ймовірність помилок у конфігурації. Крім того, використання Netlify для управління DNS дозволяє використовувати автоматичне оновлення SSL, що гарантує, що SSL-сертифікати домену завжди актуальні без необхідності втручання розробника.

Централізуючи управління DNS за допомогою Netlify, ми також отримуємо вигоду від його надійної інфраструктури, яка забезпечує швидке оновлення DNS та покращену продуктивність. Це гарантує, що будь-які зміни у розгортанні або налаштуваннях домену поширюються швидко і надійно.

Можливості масштабування Netlify

Інфраструктура Netlify розроблена для автоматичного масштабування для обробки навантажень будь-якого розміру. Основний сервіс, Netlify Core, дозволяє розгортати сайти/додатки та керувати ними в глобальній мережі CDN, виділяючи масштабованість як ключову перевагу. Це доповнюється такими функціями, як розширений контроль кешу та безсерверні функції, які сприяють швидкому випуску production релізів та кращій продуктивності. Платформа Netlify складається з модульних компонентів, що гарантує, що при масштабуванні платформи

продуктивність залишається високою з майже миттєвим часом завантаження для кожного відвідувача [88].

Можливості масштабування Render.com

Render.com надає як мануальне, так і автоматичне масштабування. Ручне масштабування дозволяє вказати постійну кількість екземплярів, які сервер буде використовувати, тоді як автоматичне масштабування регулює кількість екземплярів сервісу на основі завантаження процесора та пам’яті. Починаючи з 2023 року, автомасштабування в Render доступне в планах Team, Organization та Enterprise, надаючи сервіс, який масштабується на основі фактичного використання, а не наданих завищених розрахунків. Стратегія масштабування Render передбачає розрахунок, який множить поточну кількість екземплярів на співвідношення поточного та цільового використання, збільшуючи або зменшуючи масштабування за необхідності, з різними часовими вікнами для збільшення та зменшення, щоб гарантувати, що служби можуть швидко реагувати на сплески трафіку та уникати частих коливань [107].

Використовуючи можливості масштабування Netlify і Render.com, платформа AI Law Guide може забезпечити ефективне задоволення мінливого попиту, підтримуючи продуктивність і зручність роботи користувачів без необхідності масштабного управління інфраструктурою. Така масштабованість має вирішальне значення для обробки потенційного збільшення користувацького трафіку і навантаження на дані, гарантуючи, що платформа залишатиметься оперативною і надійною в міру зростання використання.

5.4.2. Взаємодія компонентів системи

Взаємодія різних компонентів системи представлені на *Діаграмі 2 (див. Додаток 2)*. Архітектура платформи ретельно розроблена для

обробки складних робочих процесів, оптимізації взаємодії з користувачем і підтримки високих стандартів безпеки та цілісності даних. Аналітичний огляд взаємодії різних компонентів представлений нижче.

Управління станом front-end додатку

Фронтенд використовує комбінацію локального управління станами React та глобального управління станами Redux для створення ефективного користувацького інтерфейсу. Стан React ідеально підходить для управління станами інтерфейсу, які не потребують спільного використання між кількома компонентами. Наприклад, значеннями полів вводу та локальними станами форм зазвичай керують всередині React-компонентів, щоб мінімізувати непотрібний рендеринг та зберегти швидкість роботи додатку.

Паттерн Flux – модель, яка забезпечує односпрямований потік даних, що спрощує управління станами у складних додатках [85] – надихнув авторів бібліотеки Redux [97]. Він централізує стан програми, роблячи його передбачуваним і простим в управлінні. Сховище Redux є єдиним джерелом істини, гарантуючи, що оновлення стану є явними і відстежуваними. Центральне сховище зберігає дані, які необхідні багатьом компонентам додатку, наприклад, дані користувача та інформація щодо аутентифікації. Завдяки Redux Thunk як проміжному компоненту, платформа може обробляти асинхронні дії, такі як виклики API за допомогою axios, які є критично важливими для отримання або публікації даних без блокування інтерфейсу користувача. Redux Persist гарантує, що основні частини глобального стану, такі як дані сеансу користувача, зберігаються в усіх сеансах браузера, використовуючи Local Storage Browser API, покращуючи взаємодію з користувачем, усуваючи необхідність повторної вибірки даних після кожного перезавантаження.

Створення юзер інтерфейсу front-end частини

У той час як React надає інструментарій для створення індивідуальних компонентів додатку, на сьогодні найбільш поширеною практикою є використання готових рішень – бібліотек компонентів – для фронтенд частини, наприклад Bootstrap, Material UI, Chakra UI, Blueprint UI, Vue Material, Vuetify, Angular Material. Перевагою використання таких бібліотек є пришвидшення розробки, узгоджений і послідовний зовнішній вигляд та функціонал додатку, використання вже протестованих компонентів, усунення необхідності підтримувати код компонентів. Проте повністю усунути необхідність створення індивідуальних компонентів, які додаток потребує, звісно, бібліотеки не взмозі.

Маршрутизація в React SPA

Маршрутизація в платформі здійснюється за допомогою бібліотеки *react-router-dom v6*, яка забезпечує декларативний спосіб управління навігацією та рендерингом компонентів на основі шляху до URL-адреси. Навігація в додатку структурована навколо компонента Routes, який діє як контейнер для компонентів Route, кожен з яких відображає певний шлях до компонента. Це призводить до ієрархічної, легкої в управлінні конфігурації маршрутів, яка узгоджується з вкладеним характером інтерфейсу користувача, дозволяючи розробникам створювати маршрути за шаблоном, подібним до структури компонентів.

Використання компонентів Link і NavLink усуває вбудовану обробку посилань браузером, забезпечуючи безперешкодну інтеграцію зі станом програми. Це гарантує миттєву навігацію, оскільки сторінка не перезавантажується; натомість компоненти, пов’язані з новим маршрутом, рендерингуються реактивно. Хук useNavigate ще більше збагачує можливості навігації, дозволяючи програмні перенаправлення, які

особливо корисні в таких сценаріях, як заповнення форм або після певних дій користувача, таких як успішна автентифікація або вихід з системи.

У цьому контексті модель react-router-dom інкапсулює сучасний досвід навігації в односторінкових додатках, забезпечуючи надійний і підтримуваний підхід до маршрутизації, що робить значний внесок у чуйний та інтуїтивно зрозумілий характер платформи AI Law Guide. Структура маршрутів платформи розроблена таким чином, щоб відображати логічний потік додатку, причому кожен маршрут веде до окремого набору функцій, забезпечуючи користувачам чітку і просту подорож по додатку.

Використання Node.js фреймворку Express.js

Express.js, що є основою внутрішнього інтерфейсу, пропонує мінімалістичний, але потужний набір функцій для створення веб-додатків та інтерфейсів API. Його цінують за гнучкість і швидкість, забезпечуючи легкий шар, побудований на основі *Node.js*, який керує серверною логікою та маршрутами, а також інтегрує проміжне програмне забезпечення, таке як CORS, для забезпечення безпеки.

Back-end сервіс не тільки обробляє автентифікацію та безпеку користувачів, але також відповідає за отримання даних, їх обробку та управління відповідями допоміжних API. Використання патернів async/await у маршрутах Express.js забезпечує неблокування операцій вводу/виводу, що має вирішальне значення для продуктивності платформи.

Використання Prisma Client

Prisma Client, що виконує роль ORM, покращує взаємодію додатку з базою даних PostgreSQL. Він спрощує роботу з базою даних за допомогою автоматично згенерованого конструктора запитів, що зменшує

ризик помилок під час виконання та полегшує внесення змін до схеми бази даних.

PostgreSQL

Вибір *PostgreSQL* в якості системи баз даних є стратегічним завдяки її розширеним можливостям, таким як складні запити, зовнішні ключі, тригери, розріз даних (views), цілісність транзакцій і керування паралельним доступом за допомогою багатоверсійності (multiversion concurrency control). Вона забезпечує надійність, необхідну для складних взаємозв'язків даних, притаманних юридичним технологічним платформам.

Компоненти, що забезпечують безпеку даних

Бібліотека *jsonwebtoken* використовується для безпечної передачі інформації між сторонами у вигляді JSON-об'єктів, jsonwebtoken має вирішальне значення для створення та перевірки токенів у потоках автентифікації користувачів.

Надійна бібліотека для хешування паролів, *bcrypt* є важливим компонентом для захисту облікових даних користувачів. Опис методу шифрування цієї бібліотеки було описано вище в огляді стеку.

Потік автентифікації з Google OAuth 2.0

Потік автентифікації є критично важливим аспектом безпеки платформи та управління користувачами. Він починається з того, що front-end ініціює потік OAuth 2.0, де користувачеві пропонується вибрати обліковий запис в інтерфейсі Google. Після вибору Google перенаправляє назад до front-end додатку з кодом у параметрах запиту URL-адреси. Цей код потім обробляється front-end додатком і надсилається до back-end сервісу.

Back-end сервіс, використовуючи фреймворк Express.js в одному із обробників запитів, робить захищений запит до серверів Google, щоб обмінятися кодом з інформацією про профіль користувача. Після успішної аутентифікації з Google, він використовує *jsonwebtoken* для генерації JWT, які використовуються для створення безпечного сеансу для користувача. Цей токен надсилається назад до фронтенду, завершуючи цикл автентифікації. Якщо користувач є новим, Prisma Client сприяє створенню нового запису користувача в базі даних PostgreSQL. Якщо ні, він знаходить існуючого користувача, гарантуючи, що пароль і конфіденційна інформація надійно зашифровані за допомогою bcrypt.

5.4.3. Структура бази даних та взаємодія її сутностей

Схема SQL бази даних представлена у *Діаграмі 3 (див. Додаток 3)* “сутність-зв’язок” (Entity Relationship diagram) і являє собою структуру, яка підтримує складність і функціональність, що вимагаються додатком.

Дизайн, орієнтований на користувача

Центральним елементом ERD є сутність User, яка має зв’язки як з Bookmark, так і з Guide, що вказує на те, що система розроблена з урахуванням потреб користувачів. Можливість для користувачів зберігати закладки і рекомендації свідчить про те, що платформа ставить на перше місце персоналізовану організацію контенту і простоту доступу до інформації.

Гнучка автентифікація

Зв’язок сутності User з LocalAuth і SocialAuth дозволяє використовувати універсальні механізми автентифікації. Ця гнучкість має вирішальне значення для підвищення доступності користувачів і забезпечення безпечних варіантів входу, як за допомогою традиційних

комбінацій електронної пошти-пароля або через постачальників соціальних акаунтів, таких як Google, Facebook і Twitter, як показано в переліку SOCIAL_AUTH_PROVIDER. На початковому етапі лише автентифікація за допомогою Google була імплементована, проте дизайн системи проектувався з перспективою розширення провайдерів для автентифікації. Дана архітектура забезпечить безперешкодне додавання нових методів автентифікації у майбутньому.

Структурування контенту

Сутність Guide пов’язана із сутністю Criterion, яка в свою чергу пов’язана з CriterionContentItem та CriterionChecklistItem. Такий багаторівневий підхід до організації контенту дозволяє забезпечити розширене структурування рекомендацій, що дає змогу складати їх з різних критеріїв і підпунктів. Це свідчить, що метою платформи є надання не лише інформативного, але й добре організованого та деталізованого контенту, що покращує здатність користувача орієнтуватися та розуміти складну правову інформацію.

Модульність та розширюваність

Використання типу enum для CRITERION_TYPE та SOCIAL_AUTH_PROVIDER вказує на те, що система розроблена для модульності та розширюваності. У міру розвитку правових технологій нові типи критеріїв або постачальників соціальної автентифікації можуть бути легко інтегровані в платформу без необхідності внесення значних структурних змін до бази даних.

Таким чином, ERD відображає надійну, орієнтовану на користувача та гнучку систему, здатну адаптуватися до динамічних потреб юридичної галузі, забезпечуючи при цьому безпечний та персоналізований користувацький досвід.

5.4.4. Потік створення рекомендацій для користувача

На *Діаграмі 4 (див. Додаток 4)* показано процес створення та збереження рекомендацій на платформі AI Law Guide, висвітлено взаємодію між користувачем, фронтенд сервісом, бекенд-сервісом і базою даних.

Коли користувач ініціює процес створення ґайду, він починає з вибору критеріїв у front-end сервісі. Потім вибрані критерії надсилаються на сервер, який робить запит до бази даних, щоб отримати відповідну інформацію для обраних критеріїв. Після отримання списку даних, вони надсилаються назад до клієнта, де користувач може ознайомитись із змістом рекомендацій. Після того, як рекомендації було згенеровано, користувач може ініціювати збереження рекомендацій. Тоді front-end сервіс надсилає запит на збереження рекомендацій до back-end сервісу. Back-end обробляє цей запит і надсилає запит до БД для створення нової сутності Guide, пов’язуючи її з сутністю User. Після успішного створення БД надсилає відповідь back-end сервісу, який, у свою чергу, повідомляє про це front-end сервіс. Нарешті, front-end сервіс перенаправляє користувача до новоствореного довідника, завершуючи процес створення.

Отже, архітектурний дизайн та реалізація платформи AI Law Guide демонструють добре продуманий підхід, що поєднує сучасні веб-технології для створення масштабованого, безпечного та орієнтованого на користувача додатку. Front-end архітектура використовує екосистему React у поєднанні з Redux для управління станом, щоб забезпечити чуйний та інтуїтивно зрозумілий користувацький інтерфейс. Back-end частина, що працює на Node.js та Express.js, у поєднанні з надійною базою даних PostgreSQL в поєднанні з Prisma ORM, забезпечує ефективну обробку даних та автентифікацію користувачів за допомогою Google OAuth. Крім того, використання Netlify та Render.com для розгортання забезпечує

можливості автоматичного масштабування, що дозволяє платформі безперешкодно справлятися з різними навантаженнями. Разом ці компоненти утворюють цілісну систему, яка не лише відповідає сучасним вимогам юридичних технологій, але й готова до майбутнього розширення та розвитку.

ВИСНОВКИ

Технологія штучного інтелекту стрімко розвинулась за останні декілька років. Поштовхом до стрімкого акцептування інструментів ШІ стало створення організацією OpenAI великої мовної моделі, на основі якої працює популярний ChatGPT. Без перебільшення можна стверджувати, що інструменти ШІ на сьогодні використовуються практично у кожній сфері. Не винятком стала й галузь інженерії програмного забезпечення. З одного боку, учасники процесу розробки ПЗ отримали значні ***переваги*** завдяки зручності у використанні та легкодоступності інструментів ШІ: ефективність розробки підвищилась; пришвидшились темпи виконання задач; можливість використання інструментів ШІ на різних етапах розробки; сприяння доступності і демократизації сфери. З іншого боку, використання цієї технології створює певні ***ризики***: спричиняє надмірну залежність від інструментів ШІ, а тому й атрофію технічних навичок спеціалістів; потенційно можлива генерація непередбачуваних, хибних або упереджених результатів; правова невизначеність у сфері використання ШІ тощо.

Дослідження правових та етичних аспектів використання програмних артефактів, створених ШІ, виявило кілька важливих правових проблем. Найважливішою з них є ***проблема авторства та авторських прав*** у сфері ШІ. Національне законодавство різних країн, в тому числі й України, а також міжнародні конвенції присвячені праву інтелектуальної власності, захищають програми, а тому і програмні артефакти, як літературні твори в режимі авторського права. Основною передумовою надання правового захисту є творча діяльність, креативність людини, оригінальність твору. Антропоцентристський підхід авторського права

пов’язує ці оціночні поняття як притаманні діяльності або результатам діяльності фізичної особи. У випадку генерації контенту штучним інтелектом відсутня як творча складова і оригінальність роботи, так і суб’єкт захисту: ШІ не визнається автором у контексті авторського права; результати згенеровані ШІ не мають ознак оригінальності, а процес створення бракує творчості і креативності. Результати роботи ШІ є патернами і закономірностями виявленими моделлю ШІ у тренувальних даних. Єдиним винятком є країни британської традиції, де захист в режимі авторського права може бути наданий роботам, створеним за допомогою комп’ютера. Такий захист надається особі, яка забезпечила заходи, необхідні для створення твору, зазвичай – користувачу, у деяких випадках – програмісту. Проте такий піддається критиці деякими науковцями у сфері права.

Частина науковців пропонують визнати роботи згенеровані штучним інтелектом суспільним надбанням, інші – надати обмежений або альтернативний захист таким роботам, проте передумови та елементи такого захисту залишаються невизначеними. Обидва підходи обумовлені побоювання, щодо наслідків надання захисту в режимі авторського права роботам, створеним ШІ – підрив основ авторського права і принципів, на яких воно побудоване, надмірне економічне стимулювання створення таких творів, концентрація авторських прав у руках невеликої кількості компаній і, нарешті, інструменталізації поняття твору та надмірної монетизації доступу до творів.

У деяких національних юрисдикціях, програми можуть претендувати на ***патентний захист*** при дотриманні певних умов. Надання такого захисту для винаходів, створених з допомогою ШІ, не обумовлене суб’єктивними поняттями, які фігурують у авторському праві – єдиною вимогою для отримання захисту є дотримання умов патентоспроможності: винахід повинен бути новим, він має

винахідницький рівень і є промислово придатним. Системи штучного інтелекту більшістю науковців розглядається як інструмент, засіб, що допомогли створити винахід, а винахідником вважається користувач ШІ. Доречно зазначити, що існують і альтернативні підходи, де “авторство” у контексті патентного права пропонують надати ШІ, а патентні права – розробнику ШІ, або користувачу системи і розробнику спільно.

Незважаючи на відсутність обмежень у вигляді оцінки креативності і творчої складової притаманних авторському праву, використання інструментів ШІ при створенні винаходу все ж таки ускладнює процес експертизи для визначення елементів патентоздатності, особливо оцінку винахідницького рівня. Оскільки процедура оцінки винаходу у такому випадку ускладнена елементом залучення інструментів ШІ, передбачається, що патентні бюро різних країн згодом розроблять відповідні інструкції, що регулюватимуть процедуру експертизи та/або визначатимуть особу винахідника і власника патентних прав у таких випадках. Альтернативним рішенням може бути поява нового міжнародного фреймворку, що стане правовою базою, яку національні законодавці візьмуть за основу.

Найбільшою проблемою, яка може виникнути при використанні інструментів ШІ на різних етапах розробки програмного забезпечення, залишається ***порушення ліцензійних вимог***. Великі мовні моделі ШІ вимагають специфічних навчальних даних для того, щоб генерувати програмні артефакти для повторного використання. Організації, що займаються розробкою ШІ, використовують дані, що знаходяться у загальному доступі – зазвичай, проекти з відкритим вихідним кодом, що містять різні дозвільні і обмежувальні копілефт open-source ліцензії. Генерація системами ШІ результатів, що є копіями даних, на яких вони були треновані, вже стало загальновідомим фактом. Інтеграція таких згенерованих копій, у випадку коли вони походять з проектів, що містять

певні ліцензії створюють ризик порушення ліцензійних умов таких проектів. Ситуація ускладнюється тим, що часто інструменти ШІ, що використовуються для генерації програмних артефактів не надають індикаторів, що результат є копією певного джерела. Визначення походження залишається неможливим. Прикладами інструментів, що створюють такі ризики є ChatGPT, Github Copilot, Replit AI. На противагу цим технологічним рішенням, Amazon CodeWhisperer пропонує функціонал, що допомагає відстежити джерело, таким чином, надаючи розробнику можливість проаналізувати результат і походження коду, щоб прийняти виважене рішення щодо використання згенерованого артефакту.

Особливу обережність варто проявляти у випадку використання рекомендацій ШІ у *пропрієтарному програмному* забезпеченні. Небезпеку для такого типу ПЗ становлять копілефт ліцензії різного рівня строгості GNU, CDDL, MPL, EPL, які були проаналізовані у роботі. Основною рисою копілефт ліцензій є вимогу розповсюджувати похідні роботи під тією ж ліцензією, що й оригінальна робота. Ліцензії такого типу часто вимагають оприлюднення вихідного коду і в поєднанні з попередньо згаданим обмеженням вони суттєво перешкоджають інтересам власників пропрієтарного ПЗ, цілями яких є обмеження доступу до програмного продукту і його захист від копіювання, використання та розповсюдження.

Використання згенерованих ШІ результатів без достатньої обережності і обізнаності призведе до порушень прав інтелектуальної власності авторів проектів, включених в масив тренувальних даних. У випадку визначення походження коду і включення його до основного проекту, розробники повинні впевнитись, що вони дотримуються умов ліцензування кожного проекту походження уникаючи *конфлікту ліцензій* – коли умови ліцензій залежностей або інтегрованого ПА, створеного ШІ несумісні з основним проектом або між собою. Розроблена у роботі *методика розв'язання конфлікту ліцензій* містить 4 основні кроки:

безпосереднє розв’язання несумісності ліцензій з чотирма можливими напрямками вирішення; консультація з правовими експертами; впровадження, документація і супровід обраного рішення; впровадження процедур постійного моніторингу.

Основна методологія запропона у роботі є юрисдикційно незалежною і розглядає згадані основні та додаткові правові, технічні та етичні проблеми, пов’язані із застосуванням штучного інтелекту в розробці програмного забезпечення. Вона орієнтована як на учасників розробки ПЗ, правових експертів, такі і на компанії, що є власниками програмних продуктів. Методологія є адаптивною і гнучкою, тому може бути пристосована до кожного окремого випадку в залежності від юрисдикції, типу проекту, що розробляється чи суб’єкта, що використовує методологію.

Методологія передбачає правовий аналіз чинної системи авторського права та патентування, вивчення норм ліцензування та їхньої застосовності до контенту, створеного ШІ, розробку стратегій дотримання законодавства для пом’якшення ліцензійних конфліктів, уникнення використання результатів, що сприяють упередженості алгоритмів ПЗ, а також безперервний процес оцінки для адаптації до мінливих правових та етичних стандартів.

Комплексна і багаторівнева методологія складається з *семи основних етапів*, які в свою чергу включають додаткові кроки рекомендовані для виконання: 1) правові дослідження та аналіз чинного законодавства; 2) оцінка ризиків; 3) ведення документації та відстеження; 4) виявлення потенційних правових проблем та система комплаєнсу; 5) розробка механізму вирішення спорів; 6) забезпечення навчання та освіти; 7) впровадження системи моніторингу та стратегії вдосконалення. Вона спирається на ретельний аналіз правового простору, комплексну оцінку інструменту ШІ (аспектів імплементації, тренувальних даних, алгоритмів,

нормативно-правових актів, що регулюють використання такого сервісу), аналіз проекту, що створюється, тестування коду проекту з інтегрованими артефактами ШІ, постійний моніторинг змін і постійну освіту користувача методології.

Для реалізації методології у практичній сфері було розроблено ***комплексну веб-платформу AI Law Guide***. Платформа виконує подвійну функцію: по-перше, вона слугує інформативним посібником для розробників, юристів і власників продуктів, які орієнтуються в складному ландшафті у сфері ШІ. По-друге, вона слугує інструментом для впровадження комплексної методології вирішення правових конфліктів у цій сфері. Сервіс зі зручним і зрозумілим дизайном, орієнтований на користувача. Він пропонує персоналізовані рекомендації на основі конкретних критеріїв, обраних користувачем таких як тип використовуваного інструменту ШІ (ChatGPT, Replit AI, Github Copilot, Amazon CodeWhisperer), характер програмного забезпечення (пропрієтарне, open-source), юрисдикцію (США, Україна, Великобританія, загальна) та основне правове питання.

Front-end частина розроблена з використанням TypeScript, React та Redux у поєднанні з Material UI для створення користувацького інтерфейсу. Back-end частина створена за допомогою Node.js, Express.js, TypeScript, Prisma ORM і PostgreSQL. Такий вибір стеку гарантує надійність, підтримуваність, стабільність та масштабованість додатку.

Розгортання та інтеграція були критично важливими факторами при розробці платформи. Використовуючи GitHub для зберігання коду клієнтської і серверної частини, GitHub також використовує інтеграцію з Netlify для хостингу front-end частини та Render.com для хостингу back-end та бази даних. Така модель підвищує ефективність розробки та забезпечує узгодженість практик розгортання. Інтеграція GitHub з Netlify і Render.com дозволяє автоматизувати розгортання при кожному додаванні

нового коду до основних гілок репозиторіїв, що відповідає сучасним практикам DevOps. Така модель розгортання не тільки спрощує процес, але й гарантує, що платформа залишається актуальною та функціональною, безперешкодно відображаючи останні зміни та оновлення.

Платформа AI Law Guide втілює в собі складну інтеграцію юридичної експертизи та передових технологій, пропонуючи унікальний інструмент у сфері ШІ та дотримання правових норм. Завдяки зручному дизайну, широкому набору функцій і стратегічному розгортанню платформа має всі можливості для того, щоб суттєво вплинути на підхід до вирішення правових конфліктів на перетині сфер ШІ, права і програмної інженерії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аронов А., Дзюбенко А. Підхід до створення студентської фабрики програм. Проблеми програмування. 2011. № 3. С. 87-93. URL: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/50974/10_s_87_93.pdf (дата звернення: 21.09.2023)
2. Договір Всесвітньої організації інтелектуальної власності про авторське право від 20.12.1996. Станом на 17.10.2023. URL: https://zakon.rada.gov.ua/laws/show/995_770 (дата звернення 15.10.2023)
3. Зеров К. Захист авторського права на комп'ютерні програми. Теорія і практика інтелектуальної власності. 2020. № 6. С. 5-14. URL: https://doi.org/10.33731/62020.233854 (дата звернення 16.10.2023)
4. Інтелектуальна власність та патентознавство : підручник / Н. О. Білоусова, Н. В. Гаврушкевич, М. А. Данильченко та ін. Київ : Політехніка, 2021. 374 с. URL: https://ela.kpi.ua/bitstream/123456789/44252/1/Intelektualna_vlasnist_2021.pdf (дата звернення 15.10.2023)
5. Коновалов В.С., Радоуцький К.Є. Сучасні принципи і методи проектування програмного забезпечення. Конспект лекцій з дисципліни "Системне програмування". Харків: УкрДАЗТ, 2015 - Ч. 2. - 109 с.
6. М.В. Пономаренко. Інтелектуальна власність у сфері програмного забезпечення. Проблеми права інтелектуальної власності. Часопис Київського університету права. 2021. № 1. С. 218 – 221. URL: https://doi.org/10.36695/2219-5521.1.2021.42 (дата звернення 14.10.2023)
7. Угода про торговельні аспекти прав інтелектуальної власності від 15.04.1994. Станом на 06.12.2005. URL: https://zakon.rada.gov.ua/laws/show/981_018 (дата звернення 15.10.2023)
8. Угода про торговельні аспекти прав інтелектуальної власності від 15.04.1994. Станом на 06.12.2005.URL: https://zakon.rada.gov.ua/laws/show/981_018 (дата звернення 15.10.2023)
9. Цивільний Кодекс України: Закон України від 16.01.2003. № 435-IV. Станом на 05.10.2023. URL: https://zakon.rada.gov.ua/laws/show/435-15#n2235 (дата звернення 14.10.2023)

10. Ярмак А.О. Категорія «формат» в праві інтелектуальної власності (аналіз вітчизняного і зарубіжного досвіду): дис... д-ра. філос.: 081:08. Харків, 2021. 208 с. URL: https://naukanew.nlu.edu.ua/nauka/download/diss/yarmak/d_yarmak.pdf (дата звернення 15.10.2023)
11. Abbott R. Allow patents on AI-generated inventions — for the good of science. 2023. Nature. URL: https://www.nature.com/articles/d41586-023-02598-2 (date of access: 17.10.2023)
12. About. OpenAI. URL: https://openai.com/about (date of access: 21.09.2023)
13. AI Across Google: PaLM 2. Google AI. URL: https://ai.google/discover/palm2 (date of access: 21.09.2023)
14. AI LawGuide. URL: https://ailawguides.com (date of access: 20.10.2023)
15. AI-powered test automation. TestSigma. URL: https://testsigma.com/ai-driven-test-automation (date of access: 20.10.2023)
16. Alice Corp. v. CLS Bank Int'l, 573 U.S. 208 (2014). June 19, 2014. №13-298. Justia US Supreme Court. URL: https://supreme.justia.com/cases/federal/us/573/208 (date of access: 17.10.2023)
17. Anticipating the Next Generative AI Breakthrough and Exploring Uncharted Territory. Cryptorank. URL: https://cryptorank.io/ru/news/feed/4d175-anticipating-next-generative-ai-breakthrough (date of access: 21.09.2023)
18. Anyoha R. The History of Artificial Intelligence. Science in the News. URL: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence (date of access: 21.09.2023)
19. Apache License Version 2.0. Apache Org URL: https://www.apache.org/licenses/LICENSE-2.0.txt (date of access: 20.10.2023)
20. Appendix M. Glossary. PostgresSQL. URL: https://www.postgresql.org/docs/16/glossary.html (date of access: 20.10.2023)
21. Artificial Intelligence and Copyright. A Notice by the Copyright Office, Library of Congress on 08/30/2023. No 2023-18624. P. 59942-59949 URL: https://www.federalregister.gov/documents/2023/08/30/2023-18624/artificial-intelligence-and-copyright (дата звернення 17.10.2023)
22. Baca M., Suzuki G. J., Sulakian S. Copyrights, Professional Perspective - Navigating Legal Risks with AI-Generated Content. Bloomberg Law. URL:

https://www.bloomberglaw.com/external/document/X955T1BO000000/copyrights-professional-perspective-navigating-legal-risks-with- (date of access: 20.10.2023)
23. Bopche A. Analyzing The Scope Of Copyright Protection For AI Generated Works: Juxtaposing The Advantages And Disadvantages Of Providing Authorship Rights To A Non-Living Entity From A Juristic And Philosophical Viewpoint. Indian Journal of Law and Legal Research. Vol. 5, No 4. URL: https://3fdef50c-add3-4615-a675-a91741bcb5c0.usrfiles.com/ugd/3fdef5_f12f550d82884ad69b553f4cf7ca0862.pdf (date of access: 21.09.2023)
24. Chen J., Yoshida N., Takada H. An investigation of licensing of datasets for machine learning based on the GQM model. 2023. URL: https://doi.org/10.48550/arXiv.2303.13735 (date of access: 20.10.2023)
25. CodeWhisperer FAQs. AWS. URL: https://aws.amazon.com/codewhisperer/faqs (date of access:20.10.2023)
26. CodeWhisperer. AWS. URL: https://aws.amazon.com/codewhisperer (date of access: 20.10.2023)
27. Common Development and Distribution License 1.1. SPDX. URL: https://spdx.org/licenses/CDDL-1.1.html (date of access: 20.10.2023)
28. Copilot vs Replit AI. Replit Docs. URL: https://docs.replit.com/power-ups/replitai/copilot-vs-replitai (date of access: 20.10.2023)
29. Copyright Law in the Age of AI. Replit Blog. URL: https://blog.replit.com/copyright-law-in-the-age-of-ai (date of access: 20.10.2023)
30. Council Directive on the legal protection of computer programs. No 91/250/EEC of 14.05.1991. As of 19.11.1993. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31991L0250 (date of access: 15.10.2023)
31. Create from GitHub. Netlify Docs. URL: https://docs.netlify.com/create/features/cloud-project/import (date of access: 20.10.2023)
32. Databases. Render Docs. URL: https://render.com/docs/databases (date of access: 20.10.2023)
33. DDoS Protection. Render Docs. URL: https://render.com/docs/ddos-protection (date of access: 20.10.2023)
34. DeveloperHub. URL: https://developerhub.io (date of access: 21.09.2023)

35. Dixon R. Open Source Software Law. London: Artech House, 2004. 287 p.
36. DOE 1 et al v. GitHub, Inc. et al. Case Summary. UniCourt.URL: https://unicourt.com/case/pc-db5-doe-1-et-al-v-github-inc-et-al-1330346 (date of access: 20.10.2023)
37. DOE 1 et al v. GitHub, Inc. et al. Court order in Case 4:22-cv-06823-JST. United States District Court Northern District of California. URL: https://storage.courtlistener.com/recap/gov.uscourts.cand.403220/gov.uscourts.cand.403220.95.0.pdf (date of access: 20.10.2023)
38. DOE 1 et al v. GitHub, Inc. et al. Justia. URL: https://dockets.justia.com/docket/california/candce/4:2022cv06823/403220 (date of access: 20.10.2023)
39. D'Antoni M. Rossi M. A. Copyright vs. Copyleft Licencing and Software Development. 2007. 23 p. URL: https://www.researchgate.net/publication/23696501_Copyright_vs_Copyleft_Licencing_and_Software_Development (date of access: 22.10.2023)
40. Eclipse Public License - v 2.0. Eclipse Org. URL: https://www.eclipse.org/org/documents/epl-2.0/EPL-2.0.txt (date of access: 20.10.2023)
41. European parliament Report on intellectual property rights for the development of artificial intelligence technologies. 2020. No. A9-0176/2020. European Parliament. URL: https://www.europarl.europa.eu/doceo/document/A-9-2020-0176_EN.html (date of access 17.10.2023)
42. European Parliament resolution on intellectual property rights for the development of artificial intelligence technologies of 20.10.2020 . No A9-0176/2020. URL: https://www.europarl.europa.eu/doceo/document/TA-9-2020-0277_EN.html (date of access: 17.10.2023)
43. European Patent Convention. URL: https://www.epo.org/en/legal/epc/2020/index.html (date of access: 15.10.2023)
44. Express.js. URL: https://expressjs.com (date of access: 20.10.2023)
45. FAQ. Replit Docs. URL: https://docs.replit.com/power-ups/replitai/faq (date of access: 20.10.2023)
46. Features. Vite. URL: https://vitejs.dev/guide/features.html (date of access: 20.10.2023)
47. Fernández M., D., Böhm W., Vogelsang A. et al. Artefacts in software engineering: a fundamental positioning. Softw Syst Model. 2019. No 18. C.

2777–2786. URL: https://doi.org/10.1007/s10270-019-00714-3 (date of access: 11.10.2023)
48. Figma. URL: https://www.figma.com (date of access: 20.10.2023)
49. FOSSA Editorial Team. Generative AI and Software Development: Copyright Law and License Compliance. 2023. FOSSA. https://fossa.com/blog/generative-ai-and-software-development-copyright-law-and-license-compliance (date of access: 22.10.2023)
50. Fully Managed TLS Certificates. Render Docs. URL: https://render.com/docs/tls (date of access: 20.10.2023)
51. Gatto J. G. Solving Open Source Problems with AI Code Generators – Legal Issues and Solutions, Part 1. 2023. Sheppard, Mullin, Richter & Hampton LLP logo. URL: https://www.lawoftheledger.com/wp-content/uploads/sites/15/2023/04/AI-Code-Generators-Article_Part-1-0423.pdf (date of access: 20.10.2023)
52. General Data Protection Regulation: Regulation (EU) 2016/679 of 14.04.2016. URL: https://gdpr-info.eu (date of access: 20.10.2023)
53. George A., Walsh T. Artificial intelligence is breaking patent law. 2022. Nature. URL: https://www.nature.com/articles/d41586-022-01391-x (date of access: 17.10.2023)
54. Ghahramani Z. Introducing PaLM 2. Google Keyword. URL: https://blog.google/technology/ai/google-palm-2-ai-large-language-model (date of access: 21.09.2023)
55. Ghostwriter. Replit. URL: https://replit.com/site/ghostwriter (date of access: 21.09.2023)
56. Git. URL: https://git-scm.com (date of access: 20.10.2023)
57. GitHub Copilot Product Specific Terms. GitHub. URL: https://github.com/customer-terms/github-copilot-product-specific-terms (date of access: 22.10.2023)
58. GitHub Copilot. GitHub.URL: https://github.com/features/copilot (date of access: 21.09.2023)
59. GitHub. URL: https://github.com (date of access: 20.10.2023)
60. Glaubitz A. How should liability be attributed for harms caused by biases in Artificial Intelligence? 2021. URL: https://politicalscience.yale.edu/sites/default/files/glaubitz_alina.pdf (date of access: 20.10.2023)
61. GNU General Public Licence Version 3 of 29.06.2007. GNU Org. URL: https://gnu.org/licenses/gpl-3.0.txt (date of access: 20.10.2023)

62. Guadamuz A. Artificial intelligence and copyright. WIPO Magazine. 2017. URL: https://www.wipo.int/wipo_magazine/en/2017/05/article_0003.html (date of access 17.10.2023)
63. Haenlein M., Kaplan A., A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California Management Review. Vol 61, No 4. P. 5-14. URL: https://www.researchgate.net/publication/334539401_A_Brief_History_of_Artificial_Intelligence_On_the_Past_Present_and_Future_of_Artificial_Intelligence (date of access: 21.09.2023)
64. Haripriya S., Manikandan, L. C. A Study on Artificial Intelligence Technologies and its Applications. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2020. Vol. 6, No 4. P. 336–344. URL: https://ijsrcseit.com/paper/CSEIT206455.pdf (date of access: 21.09.2023)
65. Hiter S. AI and Privacy Issues: What You Need to Know. Eweek. URL: https://www.eweek.com/artificial-intelligence/ai-privacy-issues (date of access: 20.10.2023)
66. HTTPS (SSL). Netlify Docs. URL: https://docs.netlify.com/domains-https/https-ssl/#app (date of access: 20.10.2023)
67. Hugenholtz P.B., Quintais J.P. Copyright and Artificial Creation: Does EU Copyright Law Protect AI-Assisted Output?. IIC. 2021. No 52, P. 1190–1216. URL: https://doi.org/10.1007/s40319-021-01115-0 (date of access: 17.10.2023)
68. Ignjatovic A. Understanding and fixing N+1 query. Medium. 2020.URL: https://medium.com/doctolib/understanding-and-fixing-n-1-query-30623109fe89 (date of access: 20.10.2023)
69. Ilan D., Cohen B. J.. Court Allows Three of Plaintiffs' Claims to Survive Motion to Dismiss in Lawsuit That Could Significantly Impact the World of Generative AI. Cleary's IP & Technology Insights. URL: https://www.clearyiptechinsights.com/2023/06/court-allows-three-of-plaintiffs-claims-to-survive-motion-to-dismiss-in-lawsuit-that-could-significantly-impact-the-world-of-generative-ai/ (date of access: 20.10.2023)
70. IP and Software. 2008. WIPO. URL: https://www.wipo.int/wipo_magazine/en/2008/06/article_0006.html (date of access: 08.10.2023)

71. Jandhyala S., Kim J., Bhattacharyya A. Copyrights, Professional Perspective - IP Issues With AI Code Generators. 2023. Bloomberg Law. URL: https://www.natlawreview.com/article/solving-open-source-problems-ai-code-generators-legal-issues-and-solutions-part-1 (date of access: 20.10.2023)
72. Joinup Licensing Assistant. European Commission. URL: https://joinup.ec.europa.eu/collection/eupl/solution/joinup-licensing-assistant/jla-compatibility-checker (date of access: 20.10.2023)
73. JSON Web Token. URL: https://jwt.io/introduction (date of access: 20.10.2023)
74. Laurent A. M. St. Understanding Open Source and Free Software Licensing. Sebastopol: O'Reilly Media, 2008. 208 p.
75. Legal and Security Info. Replit Docs. URL: https://docs.replit.com/legal-and-security-info/licensing-info (date of access: 20.10.2023)
76. Licensing a repository. GitHub Docs. URL: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository (дата звернення 20.10.2023)
77. Limitations of GitHub Copilot Chat. GitHub Docs.URL: https://docs.github.com/en/copilot/github-copilot-chat/about-github-copilot-chat#limitations-of-github-copilot-chat (date of access:20.10.2023)
78. Lucidchart. URL: https://www.lucidchart.com (date of access: 20.10.2023)
79. Mehmet A., Marcelloni F., Tekinerdogan B, et al. Active Software Artifacts. Lecture Notes in Computer Science. 1998. Vol 1357. P. 307–310. URL: https://doi.org/10.1007/3-540-69687-3_62 (date of access: 14.10.2023)
80. Mijwil M. History of Artificial Intelligence. 2015. URL: https://www.researchgate.net/publication/322234922_History_of_Artificial_Intelligence (date of access: 21.10.2023)
81. MIT License. Massachusetts Institute of Technology. URL: https://www.mit.edu/~amini/LICENSE.md (date of access: 20.10.2023)
82. Mittal A. Prompt Engineering. Essential Guide to Prompt Engineering in ChatGPT. Unite.ai. URL: https://www.unite.ai/prompt-engineering-in-chatgpt/ (дата звернення:: 21.09.2023)
83. Moulaison-Sandy H. L. What Is a Person? Emerging Interpretations of AI Authorship and Attribution. Proceedings of the Association for Information Science and Technology. Vol. 60 No 1. 2023. P. 279 - 290. URL: https://doi.org/10.1002/pra2.788 (date of access: 20.09.2023)

84. Mozilla Public License Version 2.0. Mozilla Org URL: https://www.mozilla.org/en-US/MPL/2.0/ (date of access: 20.10.2023)
85. Munasinghe S.. Flux and Redux. 2018. Medium. URL: https://medium.com/@sidathasiri/flux-and-redux-f6c9560997d7 (date of access: 20.10.2023)
86. Nemec D. R. Rann L. M. AI and Patent Law: Balancing Innovation and Inventorship. 2023. Skadden Insights. URL: https://www.skadden.com/insights/publications/2023/04/quarterly-insights/ai-and-patent-law (date of access: 17.10.2023)
87. Netlify DNS. Netlify Docs. URL: https://docs.netlify.com/domains-https/netlify-dns/#app (date of access: 20.10.2023)
88. Netlify platform overview. Netlify Docs. URL: https://docs.netlify.com/platform/overview/ (date of access: 20.10.2023)
89. NewsAPI. URL: https://newsapi.org/ (date of access: 20.10.2023)
90. Node.js. URL: https://nodejs.org/en/about (date of access: 20.10.2023)
91. Nordlander E., Loreto D. et al. Software Licenses: Taxonomy and Analysis. 2004. URL: https://dspace.mit.edu/bitstream/handle/1721.1/34962/6-901Fall2003/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-901Fall2003/5DA43C1A-3307-48BF-B23F-B4CE8261E682/0/Final_Project_Oliner.pdf (date of access: 20.10.2023)
92. O'Rourke J. S. IV, Tawse N. Artificial Intelligence and Intellectual Property: Who Owns Property Created by an Algorithm or a Robot? Journal of Organizational Behavior Education. 2020. No 13. P. 29–48. URL: https://www.academia.edu/45212331/Artificial_Intelligence_and_Intellectual_Property_Who_Owns_Property_Created_by_an_Algorithm_or_a_Robot (date of access: 20.10.2023)
93. Open Source License Compliance. Mend.io. URL: https://www.mend.io/open-source-license-compliance/ (date of access: 20.10.2023)
94. OpenAI Codex. OpenAI. URL: https://openai.com/blog/openai-codex (date of access: 21.09.2023)
95. Pfeiffer H.License Incompatibilities in Software Ecosystems. 2022. URL: https://www.researchgate.net/publication/359000255_License_Incompatibilities_in_Software_Ecosystems (date of access: 20.10.2023)
96. Postman. URL: https://www.postman.com (date of access: 20.10.2023)
97. Prior Art. Redux. URL: https://redux.js.org/understanding/history-and-design/prior-art (date of access: 20.10.2023)

98. Prisma Studio. URL: https://www.prisma.io/studio (date of access: 20.10.2023)
99. Prisma.io. URL: https://www.prisma.io (date of access: 20.10.2023)
100. Rainbow Table Attack. Beyond Identity. URL: https://www.beyondidentity.com/glossary/rainbow-table-attack (date of access: 20.10.2023)
101. React Developer Tools. React. URL: https://react.dev/learn/react-developer-tools (date of access: 20.10.2023)
102. Redux Devtools. GitHub. URL: https://github.com/reduxjs/redux-devtools (date of access: 20.10.2023)
103. Render. URL: https://render.com/ (date of access: 20.10.2023)
104. Research Handbook on Intellectual Property and Artificial Intelligence / ed. by R. Abbott, Cheltelham: Edward Elgar Publishing Limited, 2022. 498 p.
105. Saleh Z. Artificial Intelligence Definition, Ethics and Standards. 2019. URL: https://www.researchgate.net/publication/332548325_Artificial_Intelligence_Definition_Ethics_and_Standards (date of access: 21.09.2023)
106. Sandeen S. K.. Intellectual property deskbook for the business lawyer: a transactions-based guide to intellectual property law. Chicago: American Bar Association, Section of Business Law, Intellectual Property Committee, 2007. 251 p.
107. Scaling. Render Docs. URL: https://render.com/docs/scaling (date of access: 20.10.2023)
108. Schneider M., Grinsell J., Russell T., et al. Identifying Indicators of Bias in Data Analysis Using Proportionality and Separability Metrics. International Conference on Social Computing, Behavioral-Cultural Modeling, & Prediction and Behavior Representation in Modeling and Simulation. 12.07.2019. URL: https://www.researchgate.net/publication/334084817_Identifying_Indicators_of_Bias_in_Data_Analysis_Using_Proportionality_and_Separability_Metrics (date of access: 20.10.2023)
109. Secure TCP/IP Connections with SSL. PostgresSQL. URL: https://www.postgresql.org/docs/current/ssl-tcp.html (date of access: 20.10.2023)
110. Smith C. et al. The History of Artificial Intelligence. History of Computing. University of Washington. 2006. 27 p. URL: https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf (date of access: 21.09.2023)

111. Tabnine. URL: https://www.tabnine.com (date of access: 21.09.2023)
112. Tan D. Generative AI and Authorship in Copyright Law. National University of Singapore. The Centre for Technology, Robotics, Artificial Intelligence & the Law. 2023. URL: https://law.nus.edu.sg/trail/generative_ai_and_authorship_in_copyright_law (date of access 17.10.2023)
113. Taskade. URL: https://www.taskade.com/ (date of access: 21.09.2023)
114. Terms of use. OpenAI. URL: https://openai.com/policies/terms-of-use (date of access: 22.10.2023)
115. The 2-Clause BSD License. Open Source Initiative. URL: https://opensource.org/license/bsd-2-clause (date of access: 20.10.2023)
116. The 3-Clause BSD License. Open Source Initiative. URL: https://opensource.org/license/bsd-3-clause (date of access: 20.10.2023)
117. Theneo. URL: https://www.theneo.io (date of access: 21.09.2023)
118. Using OAuth 2.0 to Access Google APIs. Google for Developers. URL: https://developers.google.com/identity/protocols/oauth2?hl=en (date of access: 20.10.2023)
119. Visual Studio Code. URL: https://code.visualstudio.com (date of access: 20.10.2023)
120. Wang H. Authorship of Artificial Intelligence-Generated Works and Possible System Improvement in China. Beijing Law Review. 2023. Vol.14, No 2. P. 901-912. URL: https://doi.org/10.4236/blr.2023.142049 (date of access: 30.11.2023)
121. Wessendorf N. Patenting AI-generated inventions – is patent law acting the luddite? 2023. TaylorWessing. URL: https://www.taylorwessing.com/en/insights-and-events/insights/2023/09/patenting-ai-generated-inventions (date of access: 17.10.2023)
122. What is bcrypt and how does it work? Nord VPN. URL: https://nordvpn.com/blog/what-is-bcrypt/ (date of access: 20.10.2023)
123. Why Vite. Vite. URL: https://vitejs.dev/guide/why.html (date of access: 20.10.2023)
124. Xie J. An explanation of the relationship between artificial intelligence and human beings from the perspective of consciousness. Cultures of Science. 2021. Vol. 4, No 3, P. 124–134. URL: https://doi.org/10.1177/20966083211056376 (date of access: 21.09.2023)

125. Xu S., Gao Y., Fan L., et al. LiResolver: License Incompatibility Resolution for Open Source Software. ISSTA 2023: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. 2023. P. 652–663 URL: https://doi.org/10.48550/arXiv.2306.14675 (date of access: 20.10.2023)

126. Yalalov D. Bing Search Now Powered by the Next-Generation OpenAI Language Model Prometheus. Metaverse Post. URL: https://mpost.io/bing-search-now-powered-by-the-next-generation-openai-language-model-prometheus (date of access: 21.09.2023)

ДОДАТОК 1. Діаграма 1: Deployment Diagram

ДОДАТОК 2. Діаграма 2: Components Diagram

ДОДАТОК 3. Діаграма 3: Entity Relationship Diagram

ДОДАТОК 4. Діаграма 4: Guide Creation Sequence Diagram