Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Генерація Інтерфейсу через зображення макету

(тема)

Виконав:
здобувач другого року навчання,
групи СШІм-23-2
Ростислав Углик
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник доц. Вадим Шергін
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Углику Ростиславу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Генерація Інтерфейсу через зображення макету

затверджена наказом університету від 21 квітня 20 25 р. № 295Ст

2. Термін подання студентом роботи до екзаменаційної комісії 5 червня 20 25 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел та відомих наукових проектів, Python, TensorFlow, PyTorch, OpenCV, Keras, PIL (Pillow), растрові зображення графічних макетів, SVG-файли, програмний код у форматах HTML, CSS, JSX, JSON, структуровані представлення елементів інтерфейсу.

4. Перелік питань, що потрібно опрацювати в роботі
1) Основи генерації зображень з використанням штучного інтелекту
2) Огляд сучасних моделей та інструментів для генерації UI з зображень
3) Моделювання системи генерації зображень графічних макетів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	21.04.2025	виконано
2	Аналіз предметної галузі		виконано
3	Огляд існуючих методів автоматизації генерації	25.04.2025	виконано
	UI		виконано
4	Аналіз існуючих рішень ШІ, що генерують	28.04.2025	виконано
	інтерфейси на основі зображення макету		виконано
5	Дослідження форматів вхідних та вихідних	30.04.2025	виконано
	даних у системах генерації інтерфейсів		виконано
6	Моделювання структури системи	02.05.2025	виконано
	генерації UI на основі зображення		виконано
7	Аналіз викликів та обмежень	09.05.2025	виконано
8	Написання пояснювальної записки	13.05.2025	виконано
9	Перевірка на академічний плагіат	26.05.2025	виконано
10	Нормоконтроль	27.05.2025	виконано
11	Підготовка презентації та доповіді	28.05.2025	виконано
12	Попередній захист	29.05.2025	виконано
13	Рецензування	30.05.2025	виконано
14	Захист перед ЕК	05.06.2025	

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Вадим Шергін
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 72 с., 1 дод., 21 джерело.

ГЕНЕРАЦІЯ КОДУ, ГЛИБИННЕ НАВЧАННЯ, ІНТЕРФЕЙС КОРИСТУВАЧА, КОМП’ЮТЕРНИЙ ЗІР, ОБРОБКА ЗОБРАЖЕНЬ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єкт дослідження – процес автоматичної генерації коду з графічних макетів за допомогою методів штучного інтелекту.

Предмет дослідження – моделі глибинного навчання та алгоритми, що застосовуються для розпізнавання візуальних елементів інтерфейсу й генерації HTML-коду.

Мета роботи – створення інтелектуальної системи, здатної генерувати інтерфейс користувача на основі зображення макету.

Методи дослідження – аналіз існуючих рішень і технологій в галузі комп’ютерного зору та генерації коду, проєктування архітектури системи на базі глибоких нейронних мереж, реалізація програмного прототипу, тестування на прикладах макетів.

Під час виконання кваліфікаційної роботи проведено теоретичний аналіз літературних джерел і наукових публікацій щодо використання нейронних мереж для аналізу та обробки зображень, а також генерації текстової розмітки. Проаналізовано сучасні підходи до генерації інтерфейсного коду за допомогою моделей типу CNN, RNN та трансформерів. Виділено недоліки наявних рішень, зокрема обмежену точність, відсутність адаптивності та закритість архітектур. У результаті дослідження запропоновано новий підхід до автоматичної побудови HTML-інтерфейсу, що базується на інтеграції методів комп’ютерного зору, розпізнавання елементів макету та генерації структурованого коду.

ABSTRACT

Master’s thesis contains: 72 pp., 1 ann., 21 references.

ARTIFICIAL INTELLIGENCE, CODE GENERATION, COMPUTER VISION, DEEP LEARNING, IMAGE PROCESSING, USER INTERFACE.

The object of research is the process of automatic code generation from graphic layouts using artificial intelligence methods.

The subject of research is deep learning models and algorithms used to recognize visual elements of the interface and generate HTML code.

The purpose of the study is to create an intelligent system capable of generating a user interface based on a layout image.

Research methods: analysis of existing solutions and technologies in the field of computer vision and code generation, design of the system architecture based on deep neural networks, implementation of a software prototype, testing on layout examples.

During the development of the master’s thesis a theoretical analysis of literary sources and scientific publications on the use of neural networks for image analysis and processing, as well as the generation of text markup, was conducted. Modern approaches to generating interface code using models such as CNN, RNN, and transformers are analyzed. The shortcomings of existing solutions are highlighted, including limited accuracy, lack of adaptability, and closed architectures. As a result of the study, a new approach to the automatic construction of an HTML interface based on the integration of computer vision methods, layout element recognition, and structured code generation is proposed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – штучний інтелект;

CNN – Convolutional Neural Network – згорткова нейронна мережа;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

CV – Computer Vision – комп'ютерне бачення;

GUI – Graphical User Interface – графічний інтерфейс користувача;

HTML – HyperText Markup Language – мова розмітки гіпертексту;

JS – JavaScript – мова програмування, що використовується для надання динамічної поведінки елементам вебінтерфейсу;

JSX – синтаксис, що дозволяє писати HTML-подібний код у JavaScript, використовується у React;

Low-code середовище – платформи, що дозволяють розробляти програмне забезпечення з мінімальним або нульовим програмуванням;

OCR – Optical Character Recognition – оптичне розпізнавання символів;

React, Vue, Angular – популярні JavaScript-фреймворки для розробки вебінтерфейсів з використанням компонентного підходу;

RNN – Recurrent Neural Network – рекурентна нейронна мережа;

SPA – Single Page Application – односторінковий додаток; вебдодаток, що завантажується як єдина сторінка з динамічним оновленням контенту;

UX/UI – User Experience / User Interface – досвід користувача / інтерфейс користувача; поняття, що охоплюють зручність, доступність, логіку та естетику інтерфейсу;

ViT – Vision Transformer – трансформер для комп'ютерного зору;

WCAG – Web Content Accessibility Guidelines – міжнародні стандарти вебдоступності;

YOLO – You Only Look Once – модель для швидкої детекції об'єктів на зображеннях у реальному часі.

ВСТУП

У нинішньому цифровому світі розробка інтерфейсу є невід'ємною складовою при створенні програмного забезпечення. Інтерфейси виступають посередником між користувачем та функціональністю застосунку, тому їх якість, зручність та відповідність дизайнерським рішенням відіграють ключову роль у загальному сприйнятті продукту. Створення інтерфейсу зазвичай починається з побудови графічного макету, який згодом має бути втілений за допомогою коду. Проте цей процес часто є трудомістким, рутинним і потребує значних витрат часу та зусиль як від дизайнерів, так і від розробників. У зв'язку з цим виникає потреба в автоматизації процесу перетворення макету в готовий код.

Сучасні досягнення в галузі штучного інтелекту, зокрема комп'ютерного зору, глибинного навчання та трансформерних архітектур, відкривають нові можливості для вирішення цієї задачі. Автоматична генерація інтерфейсного коду за зображенням макету здатна значно прискорити розробку, знизити ймовірність помилок та забезпечити більш ефективну взаємодію між дизайном і реалізацією.

Актуальність обраної теми обумовлена зростаючими вимогами до швидкості створення якісного програмного продукту, а також активним розвитком технологій, здатних автоматизувати рутинні етапи розробки. Проблематика полягає в тому, що більшість існуючих рішень або недостатньо точні, або мають закриту архітектуру, що обмежує їх адаптацію під конкретні потреби. Саме тому дослідження, спрямоване на побудову системи генерації інтерфейсу з зображення за допомогою ШІ, є актуальним та перспективним.

Оскільки графічний інтерфейс є підґрунтям взаємодії користувача з програмним забезпеченням, для його якісної генерації потрібно враховувати принципи побудови сучасних GUI – структуру, функціональність,

стандарти та візуальну гармонію. Це дає змогу забезпечити як естетичну привабливість, так і зручність використання.

Важливу роль у розв'язанні поставленої задачі відіграє комп’ютерний зір, що дозволяє аналізувати графічні макети. Нейромережі, особливо згорткові (CNN) і трансформери, ефективно розпізнають елементи інтерфейсу, визначають їхнє положення, розміри та типи. Це є базою для подальшого перетворення зображення в структурований код.

Процес генерації коду вимагає не лише виявлення елементів, а й формування правильної HTML/CSS-структури. Такі системи мають забезпечувати чистий, зрозумілий код, який можна легко масштабувати або інтегрувати у більші проєкти. Також важливим є оптимізація й адаптивність результату.

На сьогодні вже існує низка рішень, які частково автоматизують цей процес. Проте більшість із них мають обмеження у точності, стабільності або відкритості архітектури. Тому важливо досліджувати нові підходи – від правил, заснованих на логіці, до data-driven моделей, що навчаються на великій кількості прикладів.

У межах цієї роботи проведено аналіз наявних методів, їхньої ефективності та можливостей вдосконалення. Особливу увагу приділено підвищенню точності розпізнавання, універсальності рішення та можливості масштабування під різні задачі.

1 ОСНОВИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Поняття графічного інтерфейсу користувача

1.1.1 Визначення та роль GUI

Графічний інтерфейс користувача – це візуальне середовище, що дозволяє людині взаємодіяти з комп'ютером, додатком або вебресурсом через зображення, піктограми, кнопки, меню та інші візуальні елементи, а не через текстові команди. Основна мета GUI полягає в тому, щоб забезпечити інтуїтивний, зручний та доступний спосіб комунікації з цифровими системами. Саме через GUI реалізується принцип користувацького досвіду, тобто UX (User Experience), котрий вказує на загальну ефективність та комфорт використання певного цифрового продукту [4, с. 110].

На сучасному етапі розвитку технологій, особливо в умовах активного використання штучного інтелекту, GUI перестає бути тільки інструментом навігації або доступу до функцій. Він трансформується в самостійний продукт цифрової творчості, що може бути згенерований за допомогою штучного інтелекту на основі простих запитів або візуальних макетів. Сучасні AI-системи здатні не лише аналізувати дані, але й автоматично створювати повноцінні інтерфейси на основі заданих критеріїв або навіть простих скетчів. Наприклад, зображення кнопки чи панелі, створене від руки, може бути інтерпретоване системою як структурний блок майбутнього GUI [11].

GUI виконує роль посередника між алгоритмічно складними процесами, що відбуваються у системі, та користувачем, який не має технічної освіти. У цьому аспекті важливою є інтелектуальна адаптивність інтерфейсу – можливість змінюватися відповідно до сценарію

використання, поведінки користувача, його звичок та очікувань [14]. Залучення ШІ до створення GUI означає, що ці інтерфейси більше не є статичними – вони набувають рис динамічних, контекстно-чутливих середовищ.

Особливої уваги заслуговують концепції генерації інтерфейсу за допомогою зображення. Цей підхід дозволяє автоматизувати процес створення макетів, значно скорочуючи час між розробкою концепції та реалізацією. Така методика вже активно застосовується в AI-сервісах, де на основі малюнка або завантаженого JPEG-файлу створюється базова структура GUI, яку можна адаптувати чи розширювати вручну [20].

Розвиток таких інструментів, як AutoDraw, що здатні зчитувати та інтерпретувати примітивні ескізи, вказує на важливість еволюції GUI в контексті генеративного дизайну. У таких системах інтерфейс – це вже не просто результат роботи дизайнера чи програміста, а спільне творіння людини й алгоритму [13].

Загалом, у цифрову епоху GUI є не лише «обкладинкою» для програмного забезпечення, а повноцінною частиною комунікації між людиною та штучним інтелектом. Роль GUI поступово зростає у напрямах автоматизації, інклюзії, персоналізації та крос-платформенності, де інтерфейси вже не мають єдиної форми, а змінюються в реальному часі в залежності від потреб і поведінки користувача [19].

1.1.2 Компоненти сучасного інтерфейсу

Сучасний графічний інтерфейс користувача складається з чітко окреслених компонентів, кожен з яких виконує певну функцію в межах системи взаємодії. Серед головних можливо виокремити: візуальні елементи, функціональні блоки, логічні структури, інтерактивні модулі та анімації. Візуальні елементи – це кнопки, поля введення, перемикачі, іконки, які безпосередньо сприймаються користувачем через екран.

Функціональні блоки визначають логіку поведінки програми при взаємодії з тими чи іншими елементами. Логічні структури забезпечують ієрархію: головне меню, підменю, секції, вкладки [4, с. 122].

Застосування AI у генерації GUI призводить до зміни розуміння компонентності. Раніше інтерфейс формувався вручну, блок за блоком, з дотриманням певних патернів та гайдлайнів. Сьогодні ж ШІ здатен миттєво генерувати десятки варіантів розміщення компонентів, пристосовуючи їх до типу пристрою, резолюції екрана або стилістики заданого бренду. Наприклад, у системі Scribble Diffusion користувач створює грубий скетч, а ШІ на його основі пропонує структуровану композицію елементів, які потенційно можуть бути адаптовані під реальний інтерфейс [18].

Окрім базових блоків, до сучасного GUI належать і допоміжні компоненти: підказки, повідомлення, навігаційні панелі, елементи прогресу, анімаційні ефекти. Всі ці частини разом формують цілісний користувацький досвід. Штучний інтелект дає змогу формувати цю систему не лише з позиції функціональності, але й з урахуванням контексту – наприклад, відображаючи ті елементи, які найбільше потрібні конкретному користувачу в певний момент часу [15].

Також важливим є поняття шаблонізації. Генеративні інструменти на основі зображення дозволяють створювати UI-компоненти, що відповідають певним стилістичним і логічним шаблонам. Наприклад, система Drawww.app приймає вхідне зображення і формує з нього групу елементів, з якої можна вибрати необхідні компоненти для GUI. Це значно скорочує етапи проєктування [16].

ШІ також дозволяє передбачати поведінку користувача й заздалегідь змінювати компоненти інтерфейсу. Наприклад, якщо користувач зазвичай натискає певну кнопку першою – ШІ може візуально її виділити або перемістити ближче до зони уваги. Така адаптивність дає змогу зробити інтерфейси більш «живими» та еволюційними [15].

1.1.3 Стандарти та тенденції розробки інтерфейсів

Сучасна розробка графічних інтерфейсів користувача (GUI) стрімко змінюється під впливом нових технологій, зокрема штучного інтелекту, інструментів автоматизації та потреб користувачів у швидких й адаптивних рішеннях. У центрі цього процесу стоїть пошук ефективних стандартів, що дозволяють забезпечити як візуальну послідовність, так і функціональну зручність. Із появою генеративного підходу до побудови інтерфейсів, коли зображення макета перетворюється на код, виникла потреба переосмислення багатьох дизайнерських та технічних принципів [11].

Однією з головних тенденцій є уніфікація стилістичних шаблонів та впровадження принципів UX/UI-дизайну на рівні системних стандартів. Відповідно до сучасних підходів, які висвітлюються у профільних дослідженнях, проектування GUI має базуватися на повторюваних компонентах, що можуть бути легко масштабовані й адаптовані до різних платформ [4]. Це спрощує автоматичну генерацію інтерфейсу, зменшує ризик помилок і підвищує продуктивність розробки.

Особливу увагу приділяють також модульності структури. Сучасні інструменти, як-от AutoDraw або Sketch2Code, орієнтовані на розпізнавання окремих структурних елементів інтерфейсу та їх автоматизоване кодування в HTML/CSS [13]. Це означає, що макет розглядається не як суцільне зображення, а як набір функціональних блоків, кожен із яких має певну семантику й взаємодію. Відповідно, виникає потреба у чітких технічних та візуальних стандартах для таких блоків – кнопок, форм, заголовків, меню тощо.

Ще однією важливою тенденцією є зростання ролі принципів доступності (accessibility). Інтерфейс, згенерований ШІ, має не лише відповідати візуальним очікуванням, а й забезпечувати зручність для користувачів з різними потребами. Це передбачає врахування контрастності, масштабованості, логіки навігації та семантики

елементів [3]. Тому в сучасних рекомендаціях з UX-дизайну підкреслюється важливість інтеграції стандартів WCAG (Web Content Accessibility Guidelines) у процес генерації коду.

У контексті ШІ-генерації GUI також зростає потреба у злагодженій взаємодії між дизайном і кодом. Новітні системи, наприклад, на базі трансформерів, здатні інтерпретувати зображення як вихідні дані для створення функціонального HTML/CSS/JS-коду. Це стимулює розробку єдиних форматів передавання макетів, таких як JSON або XML, де описуються позиції, стилі та функції кожного елемента [12]. Такий підхід дозволяє автоматизувати процес генерації інтерфейсів і зменшити залежність від ручного втручання дизайнерів і розробників.

Крім технічних стандартів, тенденції охоплюють і правові аспекти. Адже автоматична генерація UI, що використовує сторонні зображення чи шаблони, може спричинити питання щодо авторського права. Згідно з аналітичними матеріалами, при використанні стороннього контенту або при створенні нових інтерфейсів системами ШІ варто враховувати як національне законодавство, так і міжнародні норми стосовно інтелектуальної власності [5].

Не менш важливим аспектом є адаптивність. Сучасний інтерфейс має ефективно функціонувати на різних пристроях та екранах. Це формує тренд на застосування систем дизайну, які містять набір адаптивних шаблонів, з урахуванням популярних фреймворків, як-от Bootstrap, Material Design або Tailwind CSS [20]. Інтеграція таких систем у ШІ-моделі дозволяє скоротити час генерації та підвищити якість результату.

Таким чином, у процесі розробки інтерфейсів за допомогою штучного інтелекту виникає потреба не лише в розширенні технічного арсеналу, але й у глибокому розумінні стандартів, етичних норм і тенденцій розвитку UX/UI. Застосування чітких принципів проектування дозволяє забезпечити високу якість кінцевого продукту та сприяє подальшій автоматизації процесів [19].

1.2 Основи штучного інтелекту у комп’ютерному баченні

1.2.1 Поняття комп’ютерного зору

Комп’ютерний зір (computer vision, CV) – це галузь штучного інтелекту, що дозволяє комп’ютерам інтерпретувати й аналізувати зображення або відео з метою здобуття інформації про фізичний світ. Головне завдання полягає у перетворенні вхідних візуальних даних у структуровану інформацію, яку машина здатна опрацювати для подальшого ухвалення рішень [2, с. 340].

У межах дизайну інтерфейсів комп’ютерний зір виконує важливу функцію: він дозволяє алгоритмам розпізнавати компоненти графічного інтерфейсу (кнопки, поля, текст, заголовки) на зображенні макета. Це відкриває можливість для автоматичної генерації коду на основі візуальної структури, що помітно прискорює процес створення вебсторінок і мобільних додатків [12].

З технічного погляду, системи комп’ютерного зору працюють на основі нейронних мереж, зокрема згорткових (Convolutional Neural Networks, CNN), які здатні розпізнавати шаблони, контури та деталі на зображеннях. Ці мережі проходять навчання на великих наборах даних, що містять позначені приклади візуальних елементів. Таким чином формується модель, здатна класифікувати об’єкти, визначати їхнє положення й інтерпретувати контекст [12].

Поява трансформерних архітектур, таких як Vision Transformer (ViT), ще більше розширила можливості комп’ютерного зору. Ці моделі забезпечують вищу точність при обробці зображень, зокрема у завданнях, де необхідно враховувати просторові взаємозв’язки між елементами інтерфейсу [19].

Таким чином, комп’ютерний зір відіграє ключову роль у процесі перетворення графічного макета в функціональний інтерфейс. Воно поєднує

технології глибинного навчання, комп’ютерної графіки та обробки зображень, забезпечуючи точність і масштабованість процесу генерації [4].

1.2.2 Роль нейронних межей у розпізнаванні візуальної інформації

Розробка комп’ютерного зору відбувається через удосконалення архітектур штучного інтелекту. Найбільш розповсюдженими є згорткові нейронні мережі (CNN), які стали базовою технологією для задач класифікації та сегментації зображень. Вони відмінно підходять для виокремлення елементів інтерфейсу з графічного макета завдяки своїй здатності виявляти просторові патерни на різних рівнях абстракції [2, с.340].

Поряд із CNN, широкого розповсюдження набули гібридні архітектури, що поєднують згорткові шари з механізмами уваги (attention mechanisms), зокрема моделі на базі трансформерів. Однією з таких є Vision Transformer (ViT), котра розбиває зображення на патчі (невеликі блоки пікселів) та обробляє їх за допомогою самоуваги. Це забезпечує глибше розуміння взаємозв’язків між об’єктами на зображенні, що особливо важливо при генерації інтерфейсів з високим ступенем деталізації [19].

Ще одним важливим напрямом є сегментація зображення – розподіл його на окремі області, що відповідають конкретним елементам. Для цього застосовуються моделі, такі як U-Net чи Mask R-CNN. У контексті генерації інтерфейсів вони дозволяють точно виділити межі кнопок, полів, заголовків та інших компонентів для їхнього подальшого кодування [12].

Існують також спеціалізовані моделі, призначені саме для аналізу графічних макетів. Наприклад, моделі Sketch2Code або Pix2Code реалізують підхід «зображення → код», розпізнаючи компоненти інтерфейсу на зображенні та перетворюючи їх в HTML/CSS-структури. Ці моделі об'єднують можливості CNN, LSTM (довготривала короткочасна пам’ять) та трансформерів для генерації структурованого коду [13].

Таким чином, сучасні архітектури штучного інтелекту не лише розпізнають візуальні об’єкти, а й здатні інтерпретувати логіку розташування елементів та відтворювати її у вигляді машинозчитуваного коду [11].

1.2.3 Типи нейромереж, що використовуються в задачах CV

Ефективне генерування інтерфейсів зображень вимагає точного розпізнавання кожного візуального складника. Для цього застосовується низка способів комп’ютерного бачення, які дозволяють класифікувати, локалізувати й ідентифікувати елементи GUI на зображенні [13].

Першим етапом зазвичай є детекція об’єктів – процес виявлення компонентів інтерфейсу на зображенні та визначення їхніх координат. Для цього використовуються моделі, як-от YOLO (You Only Look Once) або Faster R-CNN, котрі здатні в режимі реального часу знаходити кнопки, поля введення, чекбокси тощо [4, с. 152].

Наступним етапом є класифікація елементів, що дає змогу системі визначити тип кожного об’єкта. Наприклад, прямокутний блок з написом може бути класифікований як кнопка, а вузьке поле з підписом – як форма введення. Для цього застосовуються згорткові мережі, які навчаються на прикладах мічених інтерфейсів [13].

Ще один важливий підхід – оптичне розпізнавання символів (OCR), що дозволяє визначити текстові складники інтерфейсу. Цей етап є критичним, адже текст відіграє ключову роль у визначенні призначення елементів. Для OCR використовуються такі системи, як Tesseract або спеціалізовані трансформерні моделі [3, с. 87].

Також використовується семантична сегментація, яка дозволяє системі розрізняти межі між компонентами, навіть якщо вони візуально подібні або частково перекриваються. Це дає змогу створювати чисту структуру DOM без помилок дублювання чи втрати елементів [12].

Інтеграція згаданих способів у комплексну систему дає змогу досягати високого рівня точності при генеруванні інтерфейсу зі зображення. На основі розпізнаних елементів формується структура майбутнього коду – з ієрархією, стилями та атрибутами, відповідно до стандартів HTML/CSS [11].

Таким чином, способи розпізнавання складників GUI є фундаментальними для реалізації підходу автоматичної побудови інтерфейсів на основі зображення. Їхня ефективність безпосередньо впливає на якість, функціональність та адаптивність згенерованого результату [19].

1.3 Методи аналізу зображень

1.3.1 Класичні алгоритми

До появи глибинного навчання аналіз зображень здійснювався за допомогою класичних алгоритмів комп’ютерного зору, які спираються на евристики, фільтри та математичні перетворення. Ці методи хоча й поступаються нейромережевим підходам у точності, проте й досі залишаються актуальними завдяки своїй простоті, швидкості й прозорості виконання [4, с. 56].

Серед базових класичних алгоритмів варто виділити детектори контурів (Canny, Sobel, Laplacian), які дають змогу визначити межі об’єктів на зображенні. У контексті інтерфейсів це дозволяє виділити структури кнопок, рамок і текстових полів [2, с. 179]. Наприклад, алгоритм Canny здійснює багатоступеневу обробку, враховуючи згладжування, знаходження градієнтів та пригнічення шумів, що забезпечує чітке виявлення лінійних структур [3, с. 51].

Іншим важливим методом є виділення ознак, зокрема таких дескрипторів, як SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features) та ORB (Oriented FAST and Rotated BRIEF). Ці алгоритми

дають змогу виявляти ключові точки й порівнювати їх між різними зображеннями, що може бути корисним при зіставленні шаблонів інтерфейсів або елементів з бази [4, с. 47].

Також використовуються методи кластеризації, як-от алгоритм k-середніх (k-means), для групування пікселів за кольором або просторовим розташуванням. Це дозволяє автоматично виділити області, що можуть відповідати елементам GUI [19].

Класичні алгоритми є ефективними в умовах, коли вхідні зображення мають невелику складність або потребують попередньої обробки перед подачею у нейронні мережі. Їх також часто поєднують із сучасними методами для підвищення точності розпізнавання, зокрема на етапах препроцесингу [11].

Отже, класичні методи залишаються актуальним інструментом в арсеналі розробників систем генерації інтерфейсів, особливо на етапах попереднього аналізу зображення та фільтрації.

1.3.2 Глибоке навчання

Глибоке навчання (deep learning) є рушійною силою сучасного аналізу зображень, зокрема у сфері розпізнавання елементів інтерфейсу для подальшої генерації коду. Воно ґрунтується на багатошарових нейронних мережах, які автоматично виявляють ознаки, структури й семантичні взаємозв’язки у вхідному зображенні [2, с. 350].

Найпоширенішою архітектурою в цьому контексті є згорткові нейронні мережі (CNN), які чудово підходять для задач візуальної класифікації, локалізації та сегментації. CNN здатні розпізнавати як прості, так і складні елементи інтерфейсу, використовуючи шари згорток, підвибірки та активації [12].

Розширенням цих підходів є трансформерні архітектури, зокрема Vision Transformer (ViT), які оперують не лише локальними, а й

глобальними взаємозв’язками між піксельними блоками. Це дозволяє значно покращити точність розпізнавання складних композицій інтерфейсу [19].

Глибоке навчання дозволяє створювати інтелектуальні моделі, що не потребують ручного опису ознак, а навчаються самостійно на великих наборах даних. Наприклад, системи типу Pix2Code або GUI2Code здатні розпізнавати структуру інтерфейсу й одразу генерувати HTML або інші формати розмітки [13].

Сучасні підходи також передбачають застосування багатозадачного навчання (multi-task learning), що дозволяє моделі водночас класифікувати елементи, виявляти їхні межі та визначати ієрархічну структуру сторінки. Це значно підвищує ефективність у практичних задачах генерації [11].

Глибоке навчання є ключовим чинником, що забезпечує точність, гнучкість та адаптивність при аналізі інтерфейсів. Його використання є обов’язковою умовою для побудови сучасних систем генерації на основі зображення.

1.3.3 Сегментація та класифікація елементів інтерфейсу

Сегментація та класифікація є завершальними, але критично важливими етапами в процесі аналізу зображень інтерфейсів. Їхнє завдання полягає у точному виділенні меж кожного елемента та встановленні його функціонального призначення [3, с. 34].

Сегментація передбачає розділення зображення на області, що відповідають окремим компонентам інтерфейсу: кнопкам, полям, заголовкам, іконкам тощо. У сучасних підходах застосовуються методи семантичної та інстанс-сегментації. Перша дозволяє класифікувати пікселі за приналежністю до певного класу (наприклад, усі кнопки), а друга – розділяє об’єкти навіть одного типу, що дозволяє уникати дублювання [4].

Для сегментації найчастіше використовуються такі архітектури, як U-Net, DeepLabV3+ та Mask R-CNN, які забезпечують високу точність при відтворенні форми й розміру об’єктів. Наприклад, Mask R-CNN дозволяє одночасно виявляти межі, класифікувати об’єкти та створювати маски сегментації [12].

Класифікація елементів – це процес, у якому кожному виявленому об’єкту надається відповідний клас. Це дозволяє системі розрізняти, наприклад, кнопку від поля введення чи текстовий блок від заголовка. Для цього застосовуються CNN або гібридні архітектури з компонентами уваги [5].

Ключову роль у класифікації відіграє семантичний аналіз тексту, отриманого через OCR. Наприклад, якщо текстовий блок містить слово «Submit», система з великою ймовірністю віднесе його до кнопки підтвердження. Семантичний контекст значно покращує точність класифікації [11].

Таким чином, поєднання точного сегментування та адекватної класифікації дозволяє не лише реконструювати структуру інтерфейсу, а й правильно інтерпретувати його логіку для подальшого кодування. Це є основою для побудови інтерфейсів, що відповідають стандартам зручності та функціональності [13].

1.4 Підходи до генерації коду інтерфейсів

1.4.1 Генерація HTML/CSS/JS

Генерація HTML, CSS та JavaScript-коду є основним етапом перетворення графічного макету в інтерактивний вебінтерфейс. Сучасні підходи до автоматизації цього процесу ґрунтуються на поєднанні комп’ютерного зору, методів сегментації зображень та моделювання структури документів. Зображення макету, надане як вхідне, аналізується

системою на предмет виявлення ключових структурних блоків: заголовків, текстових полів, кнопок, форм, іконок, контейнерів тощо. На основі результатів цього аналізу система формує відповідні HTML-елементи з їхніми властивостями та стилями CSS [11].

У процесі генерації HTML використовується ієрархічна структура елементів, де кожному блоку зображення відповідає HTML-тег (наприклад, , <header>, <nav>, <section>). Водночас CSS-код визначає зовнішній вигляд цих елементів: кольори, шрифти, відступи, розташування. Для правильного формування CSS-стилів необхідним є точне розпізнавання координат, розмірів, типів шрифтів та графічних властивостей елементів [12].

JavaScript генерується рідше, оскільки більшість систем поки орієнтовані на статичну верстку. Проте розробки у цьому напрямі тривають, особливо щодо створення інтерактивної логіки – наприклад, клікабельності кнопок або поведінки форм [20]. Деякі системи вже вміють формувати прості скрипти для обробки подій на основі шаблонів інтерфейсів. Універсальність JavaScript дозволяє створювати адаптивні елементи інтерфейсу, що реагують на дії користувача, проте точність такого коду ще поступається результатам роботи людських розробників [14].

Варто також зважати на різницю між класичними HTML/CSS/JS-фреймворками та сучасними підходами на основі компонентів (наприклад, React або Vue). У більшості випадків автоматизована генерація поки орієнтована на класичну структуру коду, хоча з’являються спроби створення готових React-компонентів із шаблонів [15].

Загалом, генерація HTML/CSS/JS-коду на основі графічного макету вже демонструє високий рівень розвитку, проте її застосування обмежується переважно однотипними, структурно простими макетами. Високий ступінь деталізації в макетах або нестандартні елементи часто призводять до помилок у семантиці HTML або стилях CSS, що потребує додаткового втручання розробника [20]. Проте подальший розвиток моделей глибокого

навчання та трансформерів обіцяє зменшити ці обмеження та наблизити генерацію до рівня професійної верстки [14].

1.4.2 Генерація компонентів

Генерація компонентів передбачає створення логічно відокремлених елементів інтерфейсу, які мають внутрішню структуру, стиль та, за потреби, функціональність. Такий підхід особливо важливий у сучасних фреймворках (React, Vue, Angular), де весь інтерфейс складається з незалежних, повторно використовуваних компонентів. Штучний інтелект у цій сфері покликаний не просто згенерувати HTML/CSS, а виокремити на макеті логічні одиниці – картки, меню, списки, модальні вікна, форми – та об’єднати їх у семантичні модулі [11].

Процес починається з аналізу макету або зображення, в якому моделі виявляють межі та функціональне призначення блоків. Завдяки глибокому навчанню на великій кількості зразків інтерфейсів, нейромережі здатні ідентифікувати не лише візуальні ознаки, а й функціональну сутність об’єктів (наприклад, форму входу або картку товару). Після цього система формує код компонента, часто в синтаксисі JSX або з використанням шаблонізаторів [20].

Важливо, що окремі компоненти можуть включати внутрішню логіку, тому штучний інтелект має враховувати сценарії використання. Наприклад, форма зворотного зв’язку може містити перевірку полів, валідацію, повідомлення про помилки. Деякі інструменти вже здатні формувати таку поведінку автоматично, базуючись на візуальних підказках або інструкціях користувача [12].

Перевага компонентного підходу – в масштабованості. Якщо інтерфейс змінюється або доповнюється, компоненти можна переосмислити незалежно один від одного. Це також дозволяє розділити відповідальність між дизайнерами, розробниками та системою генерації. Проте слабким

місцем генерації компонентів залишається інтеграція у великі проєкти зі складною логікою, де недостатньо просто згенерувати зовнішній вигляд – потрібно врахувати дані, події, взаємозв'язки між компонентами [20].

І хоча системи автоматичної генерації ще не замінюють повноцінну ручну розробку, вони вже активно використовуються для створення чорнових прототипів, прискорюючи роботу розробників і дозволяючи зосередитися на складній бізнес-логіці [11].

1.4.3 Постобробка: очищення, оптимізація, інтеграція в проєкт

Інтерактивність є ключовим аспектом сучасних інтерфейсів, отже, генерація коду, що враховує динамічну поведінку елементів, є наступним щаблем складності після формування статичних макетів. Системи, що спеціалізуються на такій генерації, повинні аналізувати не лише візуальну структуру, а й сценарії взаємодії користувача: натискання кнопок, наведення, відкриття модальних вікон, зміни стану елементів [20].

Генерація інтерактивного коду потребує поєднання front-end логіки з UI-поведінкою. Наприклад, кнопка «Показати більше» має не лише вигляд, а й функціонал: прихований контент має з'явитися після натискання. Для цього система мусить сформувати JavaScript або код у відповідному фреймворку (React, Vue), враховуючи стан елементів та їхню зміну. У сучасних моделях для цього використовуються попередньо навчені трансформери, які здатні враховувати контекст і формувати умовні інструкції на кшталт onClick, hover, toggle тощо [14].

Більш просунуті підходи містять виявлення шаблонів поведінки, які були представлені на вхідних зображеннях або відео демонстраціях. Наприклад, система може навчитися асоціювати певну структуру елементів із поведінкою, характерною для випадаючого меню або каруселі. Деякі інструменти вже реалізують генерацію логіки на основі зразків – користувач

малює умовний елемент інтерфейсу, позначає його поведінку, а ШІ генерує відповідний код [16].

Проте значною проблемою є відсутність явних вказівок у макетах щодо сценаріїв взаємодії. У таких випадках система або обмежується шаблонною логікою, або потребує уточнення від користувача. Через це на практиці інтерактивні елементи часто потребують ручного доопрацювання. Окрім того, автоматизоване створення складних анімацій, переходів або логіки зворотного зв'язку поки залишається поза межами повної автоматизації [14].

Попри ці обмеження, напрям інтерактивної генерації активно розвивається, зокрема в межах гібридних систем, де штучний інтелект взаємодіє з користувачем через промпти, вказівки або діалоги. Це дає змогу досягати компромісу між автоматизацією і контролем, дозволяючи формувати адаптивні, живі інтерфейси з мінімальними зусиллями [20].

1.5 Актуальність дослідження задачі генерації інтерфейсу

1.5.1 Попит на автоматизацію розробки

Збільшення складності веб та мобільних додатків у поєднанні з високими сподіваннями юзерів створює перманентний тиск на команди розробників щодо пришвидшення циклів створення продуктів. В цьому контексті автоматизація процесу розробки інтерфейсів набуває стратегічного значення. Одним з ключових факторів, що підсилює потребу на автоматизацію, є вимога до швидкого створення прототипів, повторного використання рішень та зменшення ймовірності людських помилок на етапах верстання або верстки складних елементів. Автоматизовані підходи, що використовують машинне навчання та генеративні моделі, дозволяють спростити та прискорити етапи, які раніше потребували ручної праці та значних ресурсів.

Впровадження генерації коду інтерфейсів на основі графічних макетів з використанням ШІ дозволяє збільшити продуктивність розробки, зменшуючи кількість рутинних дій. Наприклад, моделі, здатні зчитувати графіку, розпізнавати її структурні елементи та перетворювати їх на код, значно скорочують потребу у втручанні програмістів у базову верстку. Це особливо важливо для невеликих команд або стартапів, які працюють в режимі обмежених ресурсів і не можуть дозволити собі повноцінну команду UI/UX-дизайнерів, фронтендерів і тестувальників.

Крім того, глобальна цифрова трансформація, що триває, зумовлює попит на великі обсяги кастомізованих інтерфейсів для бізнес-процесів, що вимагає відповідних інструментів масштабування. Застосування автоматизованих інструментів дозволяє уніфікувати процеси, уникати дублювання роботи та гарантувати узгодженість стилю і логіки інтерфейсів в межах екосистеми компанії. Таким чином, ШІ-генерація інтерфейсів слугує не лише засобом прискорення розробки, а й ключовим інструментом діджиталізації в цілому [14].

1.5.2 Скорочення часу прототипування

Процес створення інтерфейсів традиційно передбачає багатоступеневе прототипування, що включає створення ескізів, візуалізацію у графічних редакторах, узгодження з командою та поступову реалізацію у вигляді коду. Цей шлях, хоч і забезпечує високу якість результату, потребує значних часових витрат, особливо у швидкоплинному середовищі стартапів чи гнучких методологій розробки на кшталт Agile. У відповідь на цю проблему дедалі більше компаній звертаються до рішень, що дозволяють скоротити час на початкові етапи проектування завдяки генерації інтерфейсів за допомогою ШІ [15].

Генеративні моделі здатні на основі найпростіших вхідних даних – начерку, текстового опису або базового шаблону – сформувати інтерфейс у

вигляді коду або готового компонента. Це зменшує залежність від проміжних етапів і дозволяє миттєво отримувати функціональний прототип, який можна протестувати та швидко скоригувати. Такі можливості особливо важливі на ранніх етапах, коли продукт ще формується, а ключовим завданням є перевірка гіпотез. Швидке прототипування дозволяє уникнути витрат на повноцінну реалізацію інтерфейсів, які можуть не виправдати сподівань [20].

Крім того, використання генерації інтерфейсів сприяє спрощенню комунікації між дизайнером, замовником і розробником. Замість описових задач або багатогодинних обговорень, ШІ дозволяє автоматично створити візуалізацію та технічну реалізацію задуму, що зменшує кількість ітерацій. Це прискорює ухвалення рішень, знижує вірогідність непорозумінь і дозволяє раніше перейти до користувацького тестування. З огляду на це, автоматизація прототипування з використанням ШІ є ключовим чинником прискорення життєвого циклу продукту та збільшення його конкурентоспроможності [15].

1.5.3 Практичне застосування

Розвиток штучного інтелекту в царині генерації інтерфейсів вже не обмежується теоретичними моделями чи експериментальними прототипами – численні приклади доводять його практичну користь у дійсному виробничому середовищі. Зокрема, автоматизовані системи генерації коду інтерфейсів впроваджуються у дизайн-платформи, IDE та low-code середовища, де вони значно полегшують працю як дизайнерів, так і розробників. Приміром, популярні інструменти, що аналізують макети з Figma або Adobe XD і генерують HTML/CSS код, демонструють, наскільки дієвим є перетворення графічного зображення у діючий інтерфейс за лічені хвилини [20].

Завдяки генеративному підходу зменшується бар’єр входження для тих, хто не має технічного бекґраунду, але прагне створювати цифрові продукти. Підприємці, маркетологи та інші нефахові користувачі можуть скористатися платформами, які на основі шаблонів, начерків або навіть текстових інструкцій створюють повноцінні вебсторінки або мобільні інтерфейси. Це сприяє демократизації технологій і відкриває нові обрії для креативної діяльності [15].

Крім того, великі компанії застосовують ці підходи у масштабованих проєктах, де потрібно забезпечити послідовність стилю, логіки взаємодії та функціональності на сотнях підсистем або сервісів. Генерація інтерфейсів на основі стандартизованих правил дає змогу автоматично створювати узгоджені й адаптивні компоненти, що зменшує ризики помилок і пришвидшує підтримку продуктів у майбутньому [20].

1.6 Принципи UX/UI дизайну як основа структурування макетів

1.6.1 Основи UX/UI

Основи UX/UI-дизайну відіграють визначальну роль у структурі інтерфейсів, що генеруються на основі зображень макетів. UX (User Experience) зосереджується на забезпеченні зручності застосування продуктом, у той час як UI (User Interface) відповідає за візуальне оформлення та інтерактивність елементів. При автоматизованій генерації інтерфейсів за зображенням макету моделі ШІ повинні враховувати типові сценарії взаємодії користувача з інтерфейсом. Це передбачає правильне виявлення логічної структури сторінки, таких як заголовки, кнопки, форми, списки та блоки контенту, а також визначення їхньої ієрархії, важливості та зв’язків [4, с. 138].

Інтерфейс повинен відповідати очікуванням користувача, що досягається завдяки впровадженню принципів інтуїтивності,

передбачуваності та послідовності. Наприклад, кнопка із закликом до дії (CTA) має бути візуально виділена й розташована там, де користувач очікує її побачити – зазвичай у нижній частині секції або після ключового пояснення. При цьому важливо уникати візуального шуму та надмірного декоративного оформлення, які ускладнюють сприйняття й можуть дезорієнтувати користувача. Для моделей ШІ це означає потребу правильної інтерпретації контрастності, розміру елементів і кольорової гами [10].

Особливу увагу слід приділяти розмітці простору та вирівнюванню, які визначають ритм і візуальну логіку сторінки. Мережі вирівнювання (grid systems), відступи між елементами (padding, margin) і симетрія відіграють важливу роль у формуванні візуальної стабільності та комфорту. Ці принципи дозволяють як дизайнерам, так і алгоритмам розпізнавання зображень створювати інтерфейси, що сприймаються як гармонійні, логічні та легко читані [14].

1.6.2 Стандартизовані шаблони макетів

Застосування стандартизованих лекал макетів є невіддільною частиною процесу генерування інтерфейсів на основі зображень, оскільки такі шаблони надають системі координат, в якій можливо передбачити розташування звичних компонентів. ШІ, навчені на великій кількості зразків, часто базуються саме на розпізнаванні таких регулярних структур, як сітки (grid), блоки заголовків, контентні секції, сайдбари та футери. Типові шаблони, як-от F-подібне сканування в веб-інтерфейсах, мають усталену ієрархію, яка допомагає як користувачам швидко орієнтуватися, так і моделям точно інтерпретувати візуальну логіку макету [11].

При генеруванні інтерфейсу важливо, щоб модель ШІ розпізнавала не лише геометрію блоків, а й функціональне призначення кожного з них. Наприклад, лекала для сторінки товару в e-commerce мають чітко визначену структуру: зображення ліворуч, назва та ціна праворуч, нижче – описи,

відгуки, рекомендовані товари. Такі патерни можуть бути легко узагальнені в навчальних датасетах, що значно полегшує їх розпізнавання на макетах і забезпечує уніфікованість генерування результату [20].

Разом із тим, шаблони не обмежують креативність, а задають каркас, в межах якого можлива варіативність. ШІ може комбінувати елементи з різних шаблонів або адаптувати їх до специфіки пристрою, розміру екрана чи контексту застосування. Наприклад, одна й та сама структура може трансформуватися з десктопної в мобільну версію, зберігаючи ключові точки взаємодії. Отже, шаблони не лише забезпечують ефективність розпізнавання, а й сприяють масштабованості дизайну [14].

1.6.3 Читабельність і доступність у дизайні

Зручність сприйняття та доступність – це не тільки естетичні властивості, але базові потреби до інтерфейсів, котрі створюються або генеруються зображенням. Зручність охоплює комфорт візуального сприйняття тексту: достатній контраст між тлом та шрифтом, оптимальний кегль, міжрядковий інтервал, розмір абзаців. Моделі ШІ повинні вміти розпізнавати текстові блоки, ідентифікувати заголовки, підзаголовки, головний текст, визначати їхні пропорції та палітру кольорів, щоб забезпечити подальшу генерацію коду з правильною типографікою [3].

Доступність, у свою чергу, передбачає створення інтерфейсів, зручних для користувачів з різними можливостями. У контексті ШІ-генерації це означає розпізнавання та дотримання принципів, які відповідають WCAG (Web Content Accessibility Guidelines): наприклад, забезпечення навігації з клавіатури, додавання альтернативного тексту до зображень, уникнення суто колірної диференціації інформації. Якщо система не врахує ці параметри при інтерпретації макету, згенерований інтерфейс може бути непридатним для широкого кола користувачів [8].

Для ефективної реалізації принципів доступності ШІ-моделі мусять не тільки ідентифікувати структуру, а й робити висновки про потенційні бар'єри: дрібний шрифт, недостатній контраст, перевантаженість або складну навігацію. Відповідно, навчання моделі мусить включати приклади як коректних, так і некоректних макетів з позначенням доступності. Це дозволяє не тільки автоматично створювати інтерфейси, але й покращувати їхню якість відповідно до етичних та правових вимог [7].

1.7 Проблематика точного розпізнавання елементів зображеннях

1.7.1 Нерівномірність макетів

Однією з найбільших проблем у розпізнаванні макетів зображень є їх нерівномірність – як у структурному, так і у візуальному сенсі. Не всі макети створюються за чітко заданими шаблонами, особливо в творчих середовищах або під час початкового прототипування. У підсумку розташування елементів може бути несиметричним, відстані між блоками – мінливими, а ієрархія заголовків – порушеною. Такі характеристики значно ускладнюють процес інтерпретації вхідного зображення ШІ-моделлю [12].

Часто моделі стикаються з труднощами при розмежуванні функціонально подібних, але візуально неструктурованих частин, що спричиняє неправильне розміщення компонентів у згенерованому коді. Наприклад, дві текстові області, які знаходяться поруч, можуть бути інтерпретовані як одна, або ж одна секція, розбита на частини, буде розпізнана як окремі блоки. Усе це впливає на цілісність фінального інтерфейсу та потребує подальшої ручної постобробки [5].

Складність також полягає в тому, що нерівномірні макети створюють неоднозначність навіть для людини, не кажучи про алгоритми. І якщо розробник може інтуїтивно розпізнати логіку компонування, то модель базується виключно на формальних ознаках, яким не завжди вдається

охопити весь спектр дизайнерських рішень. Це призводить до частих помилок – таких як неправильні відступи, спотворення пропорцій або зникнення ключових елементів при генерації [1, с. 31–44].

1.7.2 Схожість елементів

Іншою проблемою є велика подібність інтерфейсних елементів, котра часто вводить в оману системи розпізнавання. Наприклад, прямокутники однакового розміру можуть позначати як кнопки, так і поля вводу, а блоки з піктограмами – бути або іконками, або зображеннями контенту. У випадках, коли в макеті відсутні контекстуальні підказки (наприклад, підпис до кнопки), система не здатна зробити точну класифікацію – особливо за умов, коли дизайн виконаний у мінімалістичному або нестандартному стилі [12].

ШІ-моделі, навіть глибокого навчання, не завжди здатні адекватно зважати на семантику. Через це розпізнавання елементів базується переважно на зовнішніх ознаках: формі, розмірах, розміщенні. Такий підхід стає вразливим до візуального дублювання, оскільки один і той самий вигляд можуть мати зовсім різні функціональні одиниці. Наприклад, круглий елемент може бути як чекбоксом, так і аватаром користувача – але модель не завжди спроможна розрізнити їх у відриві від логіки навігації [2].

Наслідком є помилки на етапі генерації, коли один тип елементів замінюється іншим, що шкодить юзабіліті. Також це впливає на доступність, бо невірно згенеровані підписи чи відсутність тегів aria-label можуть зробити інтерфейс незручним або навіть недоступним для окремих груп користувачів. Розв’язання цієї проблеми потребує поглибленого мультимодального аналізу, де зображення доповнюються текстом або метаданими [19].

1.7.3 Обмеження якості зображень

Якість зображення є критичною для вірного аналізу макетів і подальшого генерування інтерфейсного коду. Неякісні скани, фото з екрана, викривлення перспективи або низька роздільна здатність помітно знижують ефективність навіть найсучасніших моделей. У таких випадках виникають помилки на базовому рівні: елементи не розпізнаються взагалі, обтинаються або дублюються. Найбільш поширеним прикладом є невірне виявлення кінців кнопок, посилань або блоків тексту – з чим часто стикаються користувачі генераторів, що працюють на основі скетчів або скріншотів [10].

Особливо вразливою є ситуація, коли макети мають складні фони, невиразні тіні, накладені графічні елементи. В такому випадку системи можуть переплутати декоративні елементи з функціональними або пропустити важливі блоки. Ба більше, відбиття, пересвіт, надто темні зони або артефакти JPEG-компресії можуть знищити ключові візуальні підказки, через що згенерований код втрачає логіку чи частину структури [17].

Також важливо брати до уваги обмеження у варіативності входу. Якщо система була навчена на ідеально відрендерених макетах, але отримує справжнє зображення з похибками, вона не адаптується до таких спотворень. Це пояснює поширені випадки генерації зайвих елементів, появи надмірної кількості пальців у намальованих фігурах або невірної топології інтерфейсу. Вирішенням цієї проблеми може стати попередня нормалізація зображень та адаптація моделей до роботи з «шумними» даними [10].

Ще одним важливим аспектом є те, що моделі генерації часто не мають внутрішніх механізмів для компенсації відсутності чіткості зображення. Якщо, наприклад, контури кнопок розмиті, або шрифт у текстовому блоці занадто дрібний чи неяскравий, то ймовірність правильної інтерпретації різко падає. У результаті – замість інтерактивного елементу

модель може згенерувати статичний блок, або ж взагалі проігнорувати його. Це критично для розробки інтерфейсів, де точність позиціювання та функціональна відповідність є визначальними [20].

Також обмеження якості впливають на здатність моделі враховувати контекст. Наприклад, якщо модель не може чітко розпізнати, що текст розміщений у шапці сайту чи футері через недостатню деталізацію, вона може втратити логіку макету при генерації. Така втрата контексту призводить до того, що елементи з'являються в нетипових або нефункціональних областях інтерфейсу. Це не лише шкодить зручності користувача, а й створює додаткові труднощі для розробника, якому доводиться вручну виправляти результат [11].

При роботі з графічними макетами користувачі часто передають системі неповні або частково обрізані зображення. У таких випадках навіть за наявності потужної моделі нейромережі, генерація виходить неточною: можуть дублюватися заголовки, блоки накладатися один на одного, а розмітка – руйнуватися. Це пояснює потребу в попередній перевірці якості введення перед аналізом, зокрема за допомогою автоматичних фільтрів або оцінювачів зображень. Без подібних механізмів системи штучного інтелекту демонструють суттєве падіння продуктивності [15].

1.8 Постановка задачі дослідження

З огляду на результати аналізу проблемної галузі для досягнення мети даної роботи необхідно:

– дослідити сучасні моделі та інструменти, які реалізують перетворення зображень макетів у структурований інтерфейсний код;

– порівняти rule-based та data-driven підходи до генерації UI, виявити їхні сильні та слабкі сторони;

– обрати найбільш релевантні архітектури та інструменти для побудови системи генерації інтерфейсів на основі зображень;

– проаналізувати формати вхідних і вихідних даних, що застосовуються у подібних системах, з урахуванням вимог до точності, адаптивності та масштабованості;

– змоделювати умовну структуру такої системи, описати її основні компоненти та можливі сценарії функціонування;

– оцінити можливості, виклики та перспективи розвитку генерації UI засобами штучного інтелекту, а також врахувати етичні й юридичні аспекти автоматизації дизайну.

2 ОГЛЯД СУЧАСНИЇ МОДЕЛЕЙ ТА ІНСТРУМЕНТІВ ДЛЯ ГЕНЕРАЦІЇ UI З ЗОБРАЖЕНЬ

2.1 Огляд існуючих моделей

2.1.1 Інструменти для конвертації макетів у код: огляд підходів

В рамках задачі генерування інтерфейсів з графічних макетів утворився окремий клас онлайн-інструментів, призначених для автоматичної інтерпретації візуального зображення з наступною побудовою HTML/CSS-структури.

Ці сервіси працюють з найрізноманітнішими вхідними форматами – від намальованих власноруч ескізів до скріншотів – та використовуються переважно для швидкого створення базових UI-прототипів. Вони не є повноцінною заміною розробки, проте дозволяють автоматизувати рутинні задачі на старті роботи з інтерфейсами [9].

Особливістю таких сервісів є їхня орієнтація на простоту: користувачеві не потрібно готувати спеціалізоване середовище чи володіти знаннями програмування.

Достатньо тільки завантажити або намалювати макет – і система на основі внутрішніх моделей запропонує кодову структуру інтерфейсу. У більшості випадків йдеться про генерацію HTML та CSS, інколи – із базовими інтерактивними елементами або використанням популярних фреймворків.

В цьому підпункті розглядаються два типових представники таких інструментів – Drawww та AutoDraw, які наочно демонструють варіативність підходів: від конвертації креслень до використання попередньо навчених моделей для розпізнавання структури інтерфейсу.

2.1.2 Drawww

Онлайн-сервіс Drawww є однією з найбільш відомих реалізацій задуму генерації інтерфейсу за ескізом. Інструмент дозволяє юзерам створювати прості макети власноруч, котрі потім автоматично перетворюються на візуальні компоненти інтерфейсу. Офіційний сайт показує низку прикладів, де намальовані вручну поля, кнопки та текстові блоки розпізнаються системою й перетворюються у відповідні компоненти з врахуванням структури [16].

Ця модель передбачає наявність попередньо навченого алгоритму, який аналізує контури та геометричні риси намальованих об’єктів. Завдяки зібраним даним про стандартні UX/UI-елементи, Drawww швидко визначає структуру інтерфейсу. Проте система має обмеження у точності – неточні малюнки або нестандартні форми можуть не розпізнаватися чи розпізнаватися неправильно, що є типовою проблемою в подібних інструментах [20].

Однією з важливих характеристик є легкість доступу – сервіс не вимагає авторизації, а результат можна зразу експортувати. Крім того, платформа позиціонується як навчальний інструмент для дизайнерів-початківців, дозволяючи їм швидко перевірити власні ідеї без потреби у складному ПЗ. Це робить Drawww частиною тренду на демократизацію інструментів UI-дизайну, де навіть новачок може взаємодіяти з нейромережею без бар'єрів [15].

2.1.3 AutoDraw

AutoDraw є дослідним інструментом компанії Google, що застосовує алгоритми машинного навчання для розпізнавання примітивних ескізів та заміни їх на чисті векторні картинки. В основі лежить принцип: система аналізує намальовану користувачем форму та пропонує набір готових

ілюстрацій, які найкраще відповідають формі. За задумом розробників, AutoDraw мусить прискорити процес створення інтерфейсних іконок та декоративних елементів для непрофесійних користувачів [13].

Одна з основних переваг – здатність «вгадати» намір користувача ще до завершення малюнка.

Це дозволяє прискорити ітерації при створенні прототипів, особливо на ранніх стадіях, коли дизайнер прагне швидко накидати ідеї. Проте, у випадках, коли малюнок відрізняється від шаблонів, система або не розпізнає елемент, або підставляє некоректний варіант, що може сповільнити процес [10].

Попри це, AutoDraw цікавий як прототип ілюстративної підсистеми для майбутніх інструментів UI-генерації: він показує, як штучний інтелект може виступати не як повноцінний дизайнер, а як підсилювач творчості користувача.

У перспективі подібні інструменти здатні стати частиною більших систем генерації коду за допомогою зображень, зокрема у випадках, коли треба поєднати естетичні елементи з логікою функціонування [14].

Огляд наявних моделей генерування інтерфейсів з зображень демонструє швидкий поступ цього напряму в межах штучного інтелекту. Інструменти як Drawww та AutoDraw показують, як навіть прості в реалізації рішення можуть автоматизувати перетворення візуального на функціональне – макету в код.

Попри певну обмеженість функціоналу та точності, такі сервіси відкривають нові можливості для дизайнерів та розробників, роблячи генерування UI інтерфейсів доступнішим та швидшим.

Ці приклади слугують підґрунтям для подальшого вдосконалення моделей, зокрема в напрямах точності розпізнавання, адаптивності та інтеграції у повноцінні робочі середовища.

2.2 Порівняння підходів

2.2.1 Rule-based підходи

Підходи, засновані на правилах, спираються на неухильному дотриманні заздалегідь визначених правил, що описують відповідність між елементами зображення та відповідним кодом інтерфейсу. Такі системи здебільшого застосовують фіксовані алгоритми, котрі реагують на форми, кольори, просторове розташування елементів та підписані мітки. Наприклад, прямокутна фігура з надписом може сприйматися як кнопка, а група рівновіддалених однакових елементів – як список або меню. Основна перевага цього підходу – передбачуваність і контрольованість результату. Він особливо ефективний у випадках, коли вхідні зображення створені за стандартизованими макетами й не мають візуального шуму або неочікуваних варіацій [3, с. 42].

Водночас ці підходи демонструють суттєві обмеження, коли йдеться про обробку макетів, створених без дотримання структурних правил або з використанням креативного, вільного дизайну. Системи, котрі працюють за фіксованими шаблонами, не здатні адаптуватися до нетипових ситуацій або нових, раніше не передбачених комбінацій елементів. У результаті вони або ігнорують складові інтерфейсу, або генерують некоректний код, що зменшує придатність такого підходу для складних чи нестандартних UI-дизайнів [4, с. 94].

Крім цього, rule-based системи вимагають ручного налаштування й оновлення правил, що значно ускладнює їх масштабування. Зі збільшенням кількості типів інтерфейсних компонентів і дизайнерських трендів розширення бази правил стає дедалі складнішим. Для проєктів, котрі потребують швидкої адаптації до змін у візуальних стилях, rule-based рішення поступаються моделям, здатним до самонавчання або статистичного узагальнення [3, с. 47].

2.2.2 Data-driven моделі

Data-driven підходи ґрунтуються на використанні великих обсягів розмічених даних задля навчання моделей штучного інтелекту. У контексті генерації інтерфейсів із зображень ці методи зазвичай застосовують глибоке навчання – наприклад, convolutional neural networks (CNN) або трансформери – котрі аналізують піксельну структуру зображення та прогнозують відповідні елементи коду.

Цей підхід дозволяє моделі самостійно «збагнути», як виглядає кнопка, форма чи меню, виходячи не з наперед заданих правил, а з прикладів, які вона бачила під час навчання [12].

Однією з головних переваг таких моделей є гнучкість. Вони здатні розпізнавати елементи навіть у випадку їх відмінності від звичних шаблонів – наприклад, за зміненим стилем, кольором або розміром. Це робить data-driven підходи набагато кращими для роботи з нестандартними дизайнами, художніми макетами та рукописними ескізами.

Водночас якість результатів сильно залежить від якості та різноманітності навчального датасету. Якщо модель не бачила достатньо прикладів певного типу елементів або специфічних стилів, вона може помилятися або не розпізнати їх геть [20].

Також варто вказати, що такі підходи потребують значних обчислювальних ресурсів. Для досягнення прийнятної точності генерації необхідно залучати високопродуктивне обладнання, особливо на етапах тренування та тонкого налаштування моделі.

Крім того, результат генерації може бути складним для контролю: на відміну від rule-based підходів, поведінка data-driven моделей не завжди легко передбачувана або пояснювана [12].

2.2.3 Комбіновані гібридні системи

Гібридні системи поєднують rule-based та data-driven підходи, прагнучи об'єднати передбачуваність і контрольованість перших із гнучкістю та пристосованістю других. У таких системах, розпізнавання базових елементів може виконуватися за допомогою навченої нейронної мережі, а подальше структурування та перевірка відповідності правилам UI-дизайну – через заздалегідь визначені шаблони або логіку.

Це дає змогу зменшити кількість помилок і підвищити надійність інтерфейсу, що генерується [20]. Однією з головних переваг гібридного підходу є його здатність до компенсації слабких сторін кожного з компонентів. Наприклад, якщо глибока нейромережа помилково розпізнала деякий об'єкт, система може скористатися логічним фільтром, щоб виявити й виправити помилку на підставі контексту.

У разі обробки нестандартизованих макетів така адаптивність стає ключовим чинником точності та зручності роботи [10]. Водночас створення гібридних систем є значно складнішим завданням, ніж використання одного підходу. Це вимагає не лише глибокого розуміння обох технологій, а й їхньої гармонійної інтеграції.

Програмістам необхідно ретельно обміркувати, як і на яких етапах дані будуть переходити від статистичного аналізу до логічної обробки, та забезпечити цілісність отриманого результату. Однак за правильного виконання гібридні моделі відкривають шлях до створення більш стабільних і масштабованих рішень для генерації інтерфейсів зі зображень [20].

Однією з найбільш важливих проблем при застосуванні моделей для генерування інтерфейсів зі зображень є мінливість результату.

Навіть за ідентичних умов вхідних даних система може щоразу видавати різні варіації структури або коду, що особливо помітно при роботі з нейромережами.

2.3 Основні обмеження, проблеми та виклики

2.3.1 Нестабільна якість результату

Моделі можуть хибно тлумачити межі елементів, поєднувати різні компоненти в один або навпаки – розділяти цілісну структуру на частини, які не мають функціонального сенсу [6].

Ця непостійність у поведінці пов’язана не тільки з випадковістю у самих архітектурах глибокого навчання, а й з впливом шуму, фону, різнорідності шрифтів або кольорів. У підсумку програмістам або дизайнерам доводиться вручну виправляти згенерований код, що суперечить самій задумці автоматизації. Більше того, це знижує довіру до подібних рішень у професійному середовищі та перешкоджає їхній інтеграції в робочі процеси [15].

Важливим фактором, що сприяє нестабільності якості результатів, є параметри попередньої обробки зображень та алгоритми сегментації. Невірне або неточне визначення ключових елементів у вхідних даних може спричинити значні спотворення під час генерації інтерфейсів. Наприклад, зміни в контрастності, освітленні чи фільтрах обробки здатні вплинути на точність розпізнавання структури, що, у свою чергу, призводить до непередбачуваних варіацій кінцевого продукту. Це підкреслює необхідність удосконалення механізмів корекції та адаптації моделей до різноманітних вхідних умов, а також розробки стандартів оцінювання якості згенерованого результату.

2.3.2 Залежність від якості макету

Якість зображення макета безпосередньо впливає на точність розпізнавання UI-компонентів. Якщо зображення має низьку роздільну здатність, стирання, шуми чи викривлення, моделі не здатні адекватно

виявляти елементи. Це особливо актуально для рукописних або сканованих ескізів, де контури нечіткі, а текст не завжди розбірливий. У таких випадках навіть найсучасніші data-driven моделі дають помилки – наприклад, плутають кнопки з полями вводу, або не можуть точно визначити межі блоків [15].

Окрім того, недостатній контраст або невдалий вибір кольорової палітри у макеті також ускладнює розпізнавання. Алгоритми, які працюють на рівні пікселів, залежать від чіткої візуальної структури, а отже, дизайнери повинні адаптувати макети не лише під людське сприйняття, але й під можливості ШІ-аналізу. Це створює додаткове навантаження на процес створення інтерфейсу і змушує узгоджувати художнє бачення з технічними вимогами [6].

2.3.3 Обробка складних макетів

Обробка складних графічних макетів – одне з найважчих завдань у процесі генерації користувацьких інтерфейсів з зображень. Зростаюча складність дизайну UI зумовлює до багатошарової структури макетів, де компоненти переплетені між собою функціонально й візуально. Це створює значні перепони для ШІ-моделей, які повинні коректно інтерпретувати як зміст, так і структуру зображення макету перед генерацією коду [11].

Складність виникає не лише через кількість елементів, а й через їх динамічну поведінку. Багато макетів включають адаптивні або інтерактивні компоненти, що змінюють свій вигляд залежно від платформи, роздільної здатності чи сценарію користувача. При автоматичній обробці таких зображень ШІ-система повинна не тільки виявити елементи, але й зрозуміти їх контексти – наприклад, коли один і той самий блок виступає кнопкою в одному стані і модальним вікном – в іншому [2, с. 372].

Існуючі підходи часто базуються на розпізнаванні компонентів за шаблонами, але при роботі зі складними макетами вони втрачають

ефективність. Навіть сучасні convolutional neural networks (CNN) можуть помилятися у класифікації візуально подібних, але семантично різних елементів. Візуальні декоративні блоки можуть бути сплутані з функціональними, що призводить до помилок у згенерованому коді [12].

Окрему проблему становить багаторівнева вкладеність блоків у макеті, як-от фіксовані панелі, сітки, карточки з динамічним вмістом. У таких випадках простого аналізу DOM-структури зі зображення недостатньо – потрібен семантичний аналіз і постобробка [20].

У теоретичному проєкті через складність створення повноцінної ШІ-системи пропонується багатоступінчаста обробка: початкова генерація, структурний аналіз, корекція, а далі ручне втручання або rule-based логіка. Гібридні моделі виглядають ефективнішими, бо поєднують гнучкість ШІ з формалізованими правилами [3, с. 33].

Якість вхідного зображення суттєво впливає на точність генерації макетів розмиття чи нестандартні елементи можуть призвести до необхідності ручного доопрацювання. Перспективним напрямом є self-correcting AI, що коригує власні помилки, проте поки такі моделі перебувають на стадії досліджень.

2.4 Аналіз ефективності різних штучних інтелектів

2.4.1 Порівняння результатів на прикладах

Порівняння результатів генерації UI різними ШІ-платформами показує помітні відмінності в точності, адаптивності та здатності до контекстного відтворення макетів. Скажімо, AutoDraw забезпечує доволі швидке розпізнавання простих форм, проте значно обмежений у відтворенні складної структури сторінки [13]. Його підхід ґрунтується на класифікації малюнків, що зручно для скетчів, але не забезпечує повноцінної інтерпретації веб-макетів.

Тоді як Drawww генерує HTML-розмітку на базі намальованих вручну UI-компонентів, точність його результатів часто залежить від чіткості зображення та коректності введення [16]. Хоча він підтримує базову логіку макету, нестандартні розміри або дизайнерські стилі можуть призвести до помилок. Це зменшує стабільність під час створення кросплатформених інтерфейсів.

Натомість Meitu Inc. демонструє приклад глибшої адаптації до стилістики, працюючи не лише з формою елементів, а й з кольором, просторовим розташуванням і візуальними патернами [17]. Цей підхід ближчий до машинного зору, що дозволяє генерувати інтерфейси, які точніше відтворюють макет, навіть якщо він був зроблений від руки чи з недосконалим освітленням.

Важливо вказати, що результати ШІ залежать не лише від якості моделі, а й від її навчання та способу введення. Так, Scribble Diffusion, хоч і спеціалізується на образному введенні, працює значно точніше, якщо макет подається в лінійному стилі або зі спрощеною геометрією [18]. Це підкреслює потребу в підготовці даних перед генерацією, наприклад, шляхом фільтрації чи стилізації вхідного зображення [11].

2.4.2 Продуктивність та час обробки

Продуктивність генераторів UI за допомогою ШІ тісно пов’язана з видом архітектури, обсягом даних та апаратною платформою, на якій вони функціонують. Наприклад, AutoDraw обробляє просте зображення за 1–2 секунди на браузерній платформі, оскільки працює з бібліотекою навчених шаблонів [13]. Однак він не оптимізований для складних багатошарових макетів, де час генерації може суттєво зрости.

Сервіси на кшталт Scribble Diffusion використовують глибокі нейромережі, що дозволяє досягати більшої точності, але потребує значно більшого часу обробки – до 10–15 секунд навіть на сучасних GPU [18]. У

тестах було зафіксовано, що складність макету прямо пропорційна часу обробки: чим більше елементів та деталей, тим повільніше результат.

Додатково, під час аналізу спостерігалися випадки зниження продуктивності при серійному запуску генерації – зокрема в Drawww. При запуску кількох генерацій підряд швидкість знижувалася, що свідчить про відсутність механізмів кешування або оптимізації ресурсів [16]. Це важливо враховувати при масштабному використанні.

Окрім продуктивності, треба оцінювати споживання ресурсів. Meitu Inc., наприклад, значно ефективніший у хмарному середовищі, де використовується паралельна обробка запитів, що забезпечує стабільність навіть при високому навантаженні [17]. Також деякі сервіси дозволяють користувачу вибрати режим генерації: «швидкий», «збалансований» або «детальний», що впливає на продуктивність [20].

2.4.3 Збалансованість між якістю та витратами ресурсів

Пошук балансу між якістю створеного інтерфейсу та витратами ресурсів є ключовим завданням при інтеграції ШІ в процес UI-дизайну. Наприклад, високоточні генератори, як-от Scribble Diffusion, забезпечують значно вищу візуальну відповідність, проте вимагають великих обчислювальних потужностей [18]. Це ускладнює їхнє використання на локальних машинах або в учбових проєктах із обмеженим бюджетом.

У порівнянні з цим AutoDraw, який базується на простому розпізнаванні форм, дозволяє здійснювати генерування з мінімальними ресурсами, але результат часто виглядає примітивно та потребує ручної доробки [13]. Тому для багатьох юзерів важливо мати змогу обирати між точністю та швидкістю.

Одним із рішень є гібридна модель, запропонована в Drawww, де на першому етапі формується базова розмітка, а далі користувач може вручну вносити зміни або запускати доопрацювання більш складним

генератором [16]. Це дозволяє досягти балансу без значного зростання витрат.

Крім того, сервіси на кшталт Meitu Inc. впроваджують системи пріоритетної генерації, де в залежності від типу макету і цілей юзера застосовуються різні моделі – від швидких CNN до великих трансформерів [17]. Це відповідає сучасним тенденціям до кастомізації та оптимізації ШІ для конкретних задач.

Згідно з дослідженнями у сфері ШІ для UX, все більшого значення набуває адаптивність генерації: юзерам надають інструменти для швидкої зміни режиму, візуальної перевірки й інтеграції з редакторами [4, с. 64]. Такі механізми стають визначальними для успішного застосування ШІ у сфері UI-дизайну.

2.5 Напрямки розвитку у сфері генерації UI

2.5.1 Інтеграція генеративних моделей

Один із ключових напрямків розвою в сфері генерування UI – це інтеграція генеративних моделей на основі дифузійних мереж та GAN (Generative Adversarial Networks). Ці моделі спроможні створювати цілісні інтерфейси з врахуванням контексту, стилістики та взаємозв’язків між елементами, що наближає результат до роботи професійного дизайнера [12].

До прикладу, застосування таких архітектур у Scribble Diffusion дозволяє зображенню, що на перший погляд виглядає як ескіз, перетворитися у зрозумілу структуру інтерфейсу з чіткими межами та логічним розміщенням компонентів [18].

Також активно розвивається підхід використання дифузійних моделей у поєднанні з класифікаторами зображень – це надає змогу не лише генерувати форму елементів, а й передбачати їхню функціональність у

майбутньому вебдодатку [11]. Зокрема, нові моделі здатні розпізнавати, що певний прямокутник є кнопкою, інший – формою введення, а ще інший – заголовком, навіть без прямого маркування в зображенні.

Окремо варто відзначити використання генеративних моделей у вебсервісах на кшталт Meitu Inc., де машинне навчання поєднується зі збереженням UX-патернів – це означає, що інтерфейси будуються не тільки візуально привабливо, а й з дотриманням юзабіліті [17]. Таким чином, генеративні мережі перестають бути лише інструментом для експериментів та стають практичним засобом UI-розробки.

2.5.2 Використання трансформерів у CV

Застосування трансформерів у комп'ютерному зорі докорінно міняє підходи до генерування інтерфейсів. Завдяки моделі типу Vision Transformer (ViT) або інших візуальних трансформерів, системи можуть аналізувати структуру зображення з високим рівнем деталізації та гнучкістю [12]. Це важливо при генеруванні UI з реального макету, де є як візуальні, так і функціональні елементи.

На відміну від класичних CNN, трансформери обробляють зображення в контексті всього полотна, а не через локальні фільтри. Це дає змогу ШІ бачити повний контекст макету: наприклад, розуміти, як заголовок співвідноситься з навігаційним меню або які блоки повторюються по всій сторінці [20]. Таке уявлення критично важливе для створення UI зі збереженням логіки і стилю, властивих сучасному UX-дизайну.

До того ж, трансформери краще дають раду з нестандартними форматами: вони можуть ідентифікувати навіть ті компоненти, що виконані вручну або подані в нестандартному стилі, що особливо корисно у проєктах з використанням ручних скетчів або концептуальних малюнків [11]. У цьому напрямку активно рухаються розробники Drawww, впроваджуючи комбіновані моделі з Transformer-блоками для точнішого аналізу форм [16].

Перспектива подальшого розвитку – це мультимодальні трансформери, які аналізують не лише візуальну інформацію, а й текст на зображеннях, розпізнаючи значення написів і зберігаючи їх у HTML-структурі. Це особливо актуально для локалізованих інтерфейсів, де текст грає важливу роль у логіці взаємодії [3, с. 75].

2.5.3 Перехід до генерації повноцінних SPA

Останні роки демонструють збільшення зацікавлення до генерації не лише окремих UI-елементів, а повноцінних SPA (односторінкових додатків), де весь функціонал виводиться динамічно без перезавантаження сторінки. Завдяки глибокій інтеграції ШІ в генерацію компонентів, сьогодні можливо не просто відтворити візуальну частину інтерфейсу, а згенерувати структуру проєкту з логікою рендерингу, роутингом та взаємодією між блоками [14].

У цьому контексті ключову роль відіграють моделі, що здатні розпізнавати логіку поведінки. Наприклад, генерація кнопки "Пошук" не обмежується виведенням стилізованого прямокутника, а передбачає створення відповідного подієвого хендлера, а також маршруту, до якого переходить юзер [20]. Такі підходи уже реалізуються у прототипах AI-фреймворків, що створюють не HTML-файли, а повноцінні React/Vue-компоненти.

Крім того, з огляду на розширення можливостей генеративного коду, деякі системи вже генерують базову структуру API-запитів до бекенду, створюючи умови для повної генерації MVP-продукту без участі програміста [4, с. 60]. Це відкриває перспективи нових форматів rapid-prototyping, коли ідея від макету до робочого додатку реалізується за кілька хвилин.

Водночас актуальним залишається питання якості такого коду, його масштабованості та безпеки. Тому сучасні дослідники пропонують

комбінований підхід, коли AI створює початкову структуру SPA, яку потім доопрацьовує людина. Це дозволяє зберегти баланс між швидкістю генерації та якістю результату [20].

2.6 Етичні та юридичні питання автоматизації дизайну

2.6.1 Авторське право

Застосування генеративного ШІ задля створення інтерфейсів невідворотно ставить питання авторського права. Головна проблема полягає в тому, що чинне законодавство України, зокрема Закон «Про авторське право і суміжні права» №2811-IX, передбачає, що суб'єктом авторського права може бути тільки фізична особа – людина, а не алгоритм [7]. Тому результати, згенеровані штучним інтелектом (наприклад, автоматично згенерований UI за зображенням), за чинною нормою не мають повноцінного правового захисту чи визначеного автора.

Це породжує ризики для розробників, які інтегрують такі результати у власні комерційні продукти. У випадку судового розгляду правовласник макету або навчених даних може претендувати на частину прав на кінцевий продукт. Проблема також у тому, що система ШІ може випадково або зумисне відтворити стиль або конкретні фрагменти інтерфейсу, що вже охороняються авторським правом. Це ставить розробників у складне становище стосовно ліцензування таких «творів».

2.6.2 Право на результат, створений ШІ

Важливим аспектом залишається питання правового статусу об'єктів, створених ШІ без участі людини. Приміром, коли генерація UI відбувається винятково на основі обробки зображення, без втручання користувача в дизайн чи структуру – постає питання: кому належить здобуток? Деякі

фахівці, зокрема в аналітиці ЮРЛІГА, підкреслюють, що такий результат може бути прирівняний до творів без авторства – себто публічне надбання, якщо відсутня людська творчість [6].

Проте в умовах реального використання більшість генерацій трапляються в рамках сервісу чи ПЗ, що має власну ліцензію, тому права можуть належати не розробнику моделі, а провайдеру, який надає інструмент. Це спричиняє труднощі при виведенні таких продуктів на ринок: застосування навіть частини інтерфейсу, створеної ШІ, може вимагати перевірки джерела, обґрунтування оригінальності та одержання ліцензії.

Сучасне українське право поки що не передбачає окремої категорії для таких випадків, але це питання вже порушується в професійних колах, і, вірогідно, незабаром з’являться регуляторні рамки для визначення володіння результатами штучного інтелекту.

2.6.3 Конфіденційність та захист даних при навчанні моделей

З розвитком автоматизованого дизайну особливо актуальною стає тема приватності: під час навчання моделей для генерації UI нерідко застосовуються великі обсяги персоналізованих даних або внутрішніх дизайнерських рішень компаній. Це створює потенційну загрозу витоку конфіденційної інформації, яка могла бути частиною тренувального датасету. Згідно із Законом України «Про захист персональних даних» №2297-VI, такі дії можуть бути визначені як порушення прав користувача [8].

Автоматизовані системи не завжди здатні фільтрувати вхідні дані за ознаками приватності – наприклад, інтерфейс, що містить ім’я користувача або специфічну бізнес-логіку, може бути включений до набору тренувальних прикладів і повторно з’явитися в іншій генерації. Це не лише

етична проблема, а й юридична – розробник таких рішень несе відповідальність за обробку персональних даних без згоди суб’єкта.

У відповідь на це уряд розглядає способи вдосконалення законодавства. Наприклад, у Концепції розвитку штучного інтелекту в Україні передбачається впровадження вимог прозорості для таких систем – тобто алгоритми мають зберігати інформацію про джерела навчання та механізми захисту приватних даних [9]. Такий підхід може стати основою для безпечного впровадження генеративних AI-рішень у дизайні.

Таким чином, майбутнє автоматизації UI-дизайну тісно пов’язане не лише з технічними інноваціями, а й з розвитком правової інфраструктури, здатної регулювати обробку даних. Врахування прав користувача на приватність і наявність контролю над навчанням моделей – ключові принципи, без яких впровадження таких рішень ризикує натрапити на суспільний та юридичний опір.

Розробники мають адаптуватися до нових викликів, інтегруючи в систему інструменти ідентифікації чутливої інформації, автоматичне її вилучення або анонімізацію. Це дозволить не лише забезпечити відповідність правовим нормам, а й підвищити довіру користувачів до автоматизованих систем генерації UI як на внутрішньому, так і на міжнародному рівні.

3 МОДЕЛЮВАННЯ СИСТЕМИ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ ГРАФІЧНИХ МАКЕТІВ

3.1 Побудова умовної системи

3.1.1 Модулі розпізнавання

У рамках теоретичного моделювання системи генерації інтерфейсів з графічного макету, одним із ключових складників виступає модуль розпізнавання. Його умовна функціональність полягає в аналізі зображення інтерфейсу з подальшою класифікацією елементів – таких як кнопки, поля введення, блоки зображень – на основі візуальних властивостей. Теоретично такий модуль може базуватися на моделях комп’ютерного зору (CV), які застосовують глибокі згорткові мережі або трансформерні архітектури [12].

Передбачається, що система здатна виокремлювати семантичні блоки – наприклад, форму реєстрації, меню або підвал сайту. Для збільшення точності умовна модель могла б містити функціональність оптичного розпізнавання тексту (OCR), що дає змогу пов'язати елементи з написами, полегшуючи подальшу генерацію структури [20]. Це також дає змогу відфільтровувати декоративні елементи, які не мають функціонального сенсу. Теоретично, результатом роботи такого модуля буде структуроване представлення макету у форматі JSON або іншої схеми, яке містить ієрархію елементів, їх координати, тип і текстовий вміст. Такий підхід відповідає підходам, які нині реалізуються в експериментальних системах, зокрема у царині UI-аналітики [17].

3.1.2. Умовні сценарії роботи

Під час моделювання функціонування системи важливо передбачити базові сценарії її уявного застосування. Один з них – цілковита генерація UI

на основі статичного зображення макету, котрий користувач завантажує в систему. У цьому випадку всі етапи – від розпізнавання до породження коду відбуваються автоматично на основі заздалегідь визначених алгоритмів і правил трансляції візуальних елементів у цифрову структуру [20].

Інший сценарій – напівавтоматичне редагування. В ньому користувач може взаємодіяти з проміжним результатом: наприклад, змінювати розташування елементів, додавати маркування або вказувати цільову платформу. Теоретично, це потребує наявності проміжного інтерфейсу, який поєднує візуалізацію макету з можливістю його правки, що також передбачено у багатьох сучасних дизайнерських середовищах [15].

Третім сценарієм можна вважати адаптивну генерацію – наприклад, створення кількох варіантів UI для мобільних, планшетних і десктопних пристроїв. В такому разі модель мусила б враховувати принципи responsive design, а також мати набір правил для зміни структури інтерфейсу відповідно до розміру екрана. У теоретичному підході це описується через адаптивну логіку на кшталт Media Query у CSS або Grid-систему Tailwind [10].

Усі ці сценарії не реалізовані, проте розглядаються як підґрунтя майбутнього практичного втілення та дослідження уявних можливостей генеративних систем.

3.1.3 Інтерфейси взаємодії

В рамках умовного проєктування системи важливим є також теоретичне моделювання її інтерфейсної частини. З одного боку, це може бути простий вебінтерфейс, в котрому користувач завантажує зображення, отримує попередній перегляд та має змогу експортувати результат (умовно – у форматі HTML, JSX або Flutter). Головний акцент у такому інтерфейсі робиться на візуальність і мінімальну потребу в технічних знаннях [17].

З іншого боку, в теоретичній системі передбачається і розширений режим – наприклад, для дизайнерів або розробників, котрі хочуть інтегрувати генератор у власні сервіси. У цьому випадку модель могла б взаємодіяти через API або плагіни, зокрема інтегруватися в середовище Figma, Adobe XD чи інші UI-редактори [13].

На концептуальному рівні також варто передбачити підтримку зворотного зв’язку: користувач може оцінювати якість генерації, вносити правки, а система – вчитись на основі цих дій. Це підходить до принципів UX-дизайну та відповідає вимогам до адаптивних UI-інструментів [4, с. 90].

Отже, інтерфейс у цьому проєкті розглядається не як реалізована система, а як набір умовних функцій і засобів взаємодії, що можуть бути застосовані в майбутньому для забезпечення повного циклу генерації UI з візуального макету.

3.2 Формати даних: вхідна графіка та вихідні інтерфейси

3.2.1 Підтримувані формати зображень

Система, що генерує UI з графічного макету, мусить підтримувати основні растрові формати зображень, що використовуються для передавання інтерфейсних концептів: PNG, JPEG, BMP, SVG, а також WEBP – останній через його високу ефективність стиснення без значних втрат якості. Зокрема, формат PNG є бажаним через підтримку прозорості, що полегшує сегментацію окремих елементів макету [11].

Умовна система також могла б підтримувати зображення, отримані з середовищ макетування – таких як Figma або Adobe XD – або ж скріншоти готових вебсторінок. Проте важливо відмітити, що векторні формати, зокрема SVG, значно спрощують розпізнавання елементів, оскільки вже містять структурування об’єктів, яку можна розглядати як напівавтоматичну розмітку [20].

Окрім перетворення складних форматів у більш універсальні, важливим моментом є оптимізація таких процесів задля збереження ключових елементів дизайну. Зі збільшенням ролі AI-моделей, здатних працювати з multimodal input, перспективним напрямком є підтримка складних форматів, як-от PDF чи PSD (Photoshop Document), з подальшим перетворенням у PNG чи SVG. Це дозволяє об’єднувати дизайнерські прототипи з реальними інтерфейсами та автоматично пристосовувати макети під різні платформи, враховуючи специфіку середовища й потреби кінцевого користувача. Використання AI для аналізу та екстракції важливих візуальних і текстових компонентів у реальному часі значно покращує інтеграцію дизайнерських рішень у практичне використання.

3.2.2 Вихідний код: HTML, CSS, JS

Метою розробки є створення системи, котра на основі розпізнаного макету здатна згенерувати структурований код інтерфейсу, що містить HTML для розмітки, CSS для стилізації та JavaScript для базової функціональності [14].

HTML-структура має будуватись відповідно до ієрархії блоків, які система умовно виокремлює на етапі аналізу графіки. CSS – переважно через Flexbox або Grid – визначає позиціювання, розміри, кольори й типографіку елементів. Також можлива умовна інтеграція з TailwindCSS або Bootstrap для швидшого компонування стилів [10].

JavaScript у такій системі – це інструмент взаємодії: приміром, обробка кліків, відкриття меню, валідація форм. У випадках, коли потрібно створити інтерфейс для SPA-додатків, умовно можна інтегрувати JSX (React) як варіант генерації, що дає змогу розширити систему до функціонального фронтенду [16]. Таке моделювання коду є виключно концептуальним, однак відображає сучасні тренди в автоматизації UI-генерації на основі вхідної графіки.

Отже, в теоретичній моделі система повинна не просто виробляти HTML, CSS і JS як окремі файли, а утворювати їх як цілісну, логічно взаємопов’язану структуру, що відповідає вихідному макету [20].

3.2.3 Метадані, координатні структури

Одним із ключових елементів умовної системи генерації інтерфейсів із графічного макету є побудова внутрішнього уявлення – своєрідної координатної моделі, яка дозволяє інтерпретувати зображення як набір елементів інтерфейсу. Таке уявлення базується на визначенні позицій і розмірів кожного візуального об'єкта, а також на зборі супутніх метаданих – інформації, яка описує властивості цих об'єктів. До таких властивостей можуть належати: відступи, колір фону, колір тексту, шрифт, рівень прозорості, радіус заокруглення кутів тощо [12].

Координати визначаються умовно – в пікселях або відносно контейнера – і можуть зберігатися у структурованому форматі, наприклад, таблиці або об’єктах у пам’яті програми. Кожен елемент має набір обов’язкових параметрів: положення на осі X та Y, ширина, висота, а також тип, що допомагає ідентифікувати функціональну роль – заголовок, кнопка, іконка, поле вводу тощо. Це дозволяє ще на ранньому етапі підготовки розмітки гарантувати відповідність між візуальним виглядом і семантичним наповненням інтерфейсу [20].

Метадані, які збираються паралельно з координатами, є основою для подальшої генерації стилів. Наприклад, кольорові коди елементів можуть бути використані для створення CSS-стилів, а виявлені типографічні особливості – для підключення відповідних шрифтів. Окрім візуальних параметрів, до метаданих можуть входити і значення, що стосуються інтерактивності: очікувана дія при натисканні, зміна стану елемента при наведенні, призначення для клавіатурного фокусу та інше. Це дозволяє сформувати умовну модель поведінки майбутнього інтерфейсу [11].

Окремим шаром опису можуть виступати умовні категорії або «теги», які класифікують елементи за їх функціональним призначенням. Наприклад, навігаційне меню, форма реєстрації, блок товару, заголовок секції. Завдяки такій класифікації стає можливим перетворення розпізнаного візуального контенту на структурований шаблон інтерфейсу. У межах умовної системи це відкриває шлях до напівавтоматичного компонування майбутнього HTML-коду з коректною структурою та відповідним CSS-оформленням [20].

Таким чином, координатна структура разом із метаданими виконує роль проміжного рівня між зображенням і фінальним результатом – кодом. Вона забезпечує гнучкість, масштабованість та прозорість усіх наступних етапів умовної генерації інтерфейсу на основі зображення. Навіть в умовному теоретичному моделюванні її значення є визначальним для якості кінцевого продукту.

3.3 Аналіз роботи обраних підходів на прикладах

3.3.1 Результати генерації різних моделей

Під час порівняльного аналізу підходів до створення UI-інтерфейсів за графічним зразком можливо виявити чималу варіативність якості результатів. Умовно-дослідні моделі, що функціонують на основі трансформерів, демонструють точнішу передачу структури інтерфейсу, ніж класичні CNN-моделі. Зокрема, моделі з додатковим етапом сегментації елементів дають змогу більш правильно передавати розташування об'єктів, типи кнопок чи форм, що особливо важливо в контексті UX [11].

Більш сучасні підходи, зокрема застосування diffusion-механізмів, забезпечують деталізовану генерацію дрібних елементів, як-от іконки чи тонкі лінії. Проте ці методи можуть втрачати логіку компонування, зокрема – відповідність між взаємодіючими блоками

інтерфейсу (наприклад, кнопка не має зв'язку з формою введення). Це створює враження візуальної завершеності, але не завжди функціональної цілісності [12].

Системи, що поєднують CV-розпізнавання з наперед натренованими шаблонами, можуть видавати найстабільніший результат, однак часто втрачають гнучкість.

Наприклад, інтерфейси на основі шаблонів не підтримують унікальні стилістичні елементи або кастомні компоненти дизайну, що знижує їх застосовність у креативних проєктах [20].

3.3.2 Типові помилки і викривлення

Однією з найпоширеніших проблем є неправильне визначення кордонів окремих елементів. Наприклад, система може вважати зображення кнопки текстовою міткою або ж злиття кількох блоків у єдиний прямокутник. Такі помилки пов’язані як із недостатньою кількістю навчальних даних, так і з обмеженнями алгоритмів сегментації [11].

Ще однією типовою втратою є стилістична цілісність. Часто згенерований інтерфейс має неконсистентні кольори або шрифти, не узгоджені між різними блоками. Особливо це кидається в очі при генерації CSS – якщо модель не розуміє ролі стилю у контексті загальної візуальної ієрархії, результат виглядає фрагментарно [15].

Крім того, низка моделей не враховує логіку адаптивного дизайну, що призводить до створення інтерфейсів, непридатних до масштабування або перегляду на різних пристроях. Це обмеження важливе в умовах сучасної розробки SPA або мобільних застосунків, де чітка структура DOM і гнучкість CSS – критично важливі [4, с. 40].

3.3.3 Інтерпретація результатів

Враховуючи описані вище риси, результати генерації варто тлумачити не тільки як візуальний вихід, а як проміжну ланку в системі UI-дизайну. Це значить, що навіть якщо зображення-результат виглядає функціонально завершеним, важливо оцінювати його з огляду інтерактивності, відповідності вимогам UX та подальшої можливості інтеграції у фреймворки [4, с. 44].

Тлумачення також повинно брати до уваги контекст: одна й та ж модель може показувати геть різні результати в залежності від типу вхідного зображення – його контрасту, деталізації, кольорової палітри. Саме тому підхід до аналізу мусить бути багаторівневим: технічним (набір тегів і стилів), структурним (логіка взаємозв’язків) і семантичним (відповідність задуму макету) [20].

Отже, аналіз результатів дає змогу не тільки оцінити ефективність обраної моделі, але й сформувати практичні висновки стосовно її придатності в конкретному проєкті, особливо в контексті автоматизованої генерації UI за зображеннями [12].

3.4 Основні труднощі при реалізації

3.4.1 Обмеження в комп’ютерному баченні при обробці графіки

Зважаючи на стрімкий розвиток CV-моделей (Computer Vision), переважна частина з них все ще показує помітні обмеження в точності розпізнавання структур графічних макетів. Складність полягає в тому, що зображення UI здатні мати різні стилі, шрифти, палітри кольорів та нестандартні компоненти, котрі складно класифікувати навіть досвідченому дизайнеру. Наприклад, без попереднього навчання на спеціалізованому

наборі UI-компонентів модель здатна переплутати елемент меню з звичайним декоративним блоком [12].

Інша проблема – недостатня адаптація CV-моделей до логіки взаємозв'язків між елементами. Навіть якщо модель вдало розпізнала кнопку, вона не завжди вірно визначає її функціональне призначення – чи це кнопка «Пошук», чи частина форми, чи просто елемент декоративного навігаційного меню. Це ускладнює наступну генерацію інтерактивного інтерфейсу [4, с. 52].

3.4.2 Невизначеність форматів розмітки

Ще однією вагомою перешкодою є брак уніфікованих стандартів для представлення графічних макетів у цифровій формі. Більшість наявних рішень, як-от Figma, Adobe XD чи Sketch, мають власні внутрішні формати, які не завжди зручно інтерпретуються сторонніми системами або AI-моделями. Це утруднює автоматизоване зчитування даних та побудову єдиної моделі генерації HTML/CSS-коду [15].

Окрім того, у багатьох випадках графічний макет не містить відомостей про функціональність компонентів. Наприклад, візуально поля вводу, кнопки чи вкладки можуть виглядати однаково, однак функціональне призначення кожного з них суттєво різниться. Без чіткої структури або метаданих системі генерації важко побудувати інтерфейс, що дійсно функціонує, а не просто виглядає подібно до оригіналу [20].

3.4.3 Людські й ресурсні обмеження в реалізації проєкту

Найбільшою перепоною для реалізації подібного проєкту є велика вартість ресурсів і обсяг навчальних даних, потрібних для якісної роботи ШІ. Створення ефективної моделі генерації інтерфейсів на основі зображень потребує величезної кількості розмічених прикладів UI, що мають

відповідність між вхідною картинкою та вихідним кодом. Зібрати таку базу даних самостійно – майже нереально [11].

Окрім цього, обробка тисяч графічних зображень, побудова їхніх координатних сіток, сегментація та тренування трансформерів потребують високопродуктивного обчислювального середовища. Для однієї людини, яка працює без підтримки великої інфраструктури або команди – це майже нездійсненна задача. Навіть за умов використання готових бібліотек і open-source рішень, час і складністьї такого проєкту перевищують можливості індивідуального виконання [15].

3.5 Теоретичне бачення структури ШІ

3.5.1 Узагальнена структура моделі

У контексті генерації інтерфейсів з графічних зображень умовна модель ШІ передбачає розподіл системи на кілька логічних блоків. Перший – модуль попередньої обробки, котрий відповідає за нормалізацію вхідного зображення, масштабування, сегментацію на зони та базову класифікацію елементів. Далі діє блок розпізнавання – зазвичай трансформерна або CNN-архітектура, що виконує детальну ідентифікацію компонентів: кнопок, полів, текстів, іконок [12].

Після розпізнавання дані передаються до інтерпретаційного модуля, що співвідносить виявлені візуальні елементи з шаблонами HTML/CSS. В ідеалі цей блок застосовує навчання з підкріпленням або комбіновані підходи, де система вчиться на основі зворотного зв’язку.

Останній рівень – генератор коду, що формує текстовий опис інтерфейсу відповідно до логіки розташування та стилю, зокрема метадані. Такий підхід наближає архітектуру до універсальної схеми генеративних систем [20].

3.5.2 Можливості адаптації під різні цілі

Теоретична система, описана вище, є умовно модульною, а отже легко пристосовується під різні сценарії. Наприклад, для створення адміністративних панелей інтерфейсів достатньо спростити модель стилізації, натомість для мобільних UI можна змінити підхід до сегментації, зосереджуючи увагу на touch-friendly елементах. Модель здатна працювати з різними шаблонізаторами, включно з Bootstrap, Tailwind чи навіть кастомними фреймворками [16].

Окрім візуального розпізнавання, систему можна налаштувати для додаткового аналізу текстових підписів, що дозволяє враховувати семантику назв елементів. Наприклад, якщо кнопка має напис «Submit», система може самостійно призначити їй функціонал форми, навіть якщо розташування неочевидне. Подібна гнучкість забезпечує широкі перспективи її використання не лише в дизайні, а й у прототипуванні або тестуванні UX-рішень [15].

3.5.3 Перспективи уніфікованої генерації UI

Ідея уніфікованої генерації UI – це не тільки спроба автоматизувати верстку, а й бажання стандартизувати підхід до створення інтерфейсів в цілому. У цьому напрямку активно розвиваються проєкти, що поєднують генеративні моделі з UX-патернами, що були сформовані роками дизайнерської діяльності. Наприклад, систему можна навчити типізованим підходам до побудови карток товарів, форм замовлення, та інших – і відтворювати їх, зберігаючи адаптивність та доступність [15].

У майбутньому такі системи можуть підтримувати багатомовність, адаптивну стилізацію та навіть персоналізацію – коли інтерфейс генерується відповідно до вподобань кінцевого користувача. У цьому контексті надлюдський потенціал ШІ, про який говорять експерти [21], не

стільки в потужності обчислень, скільки в здатності системи самостійно навчатися, змінювати логіку генерації під конкретні задачі та працювати як самостійний дизайнер-кодер.

Отже, побудова теоретичної системи генерації UI на базі зображень потребує ясної структуризації на рівні архітектури ШІ, гнучкості у застосуванні окремих блоків, а також адаптації до різних форматів та потреб. За правильного моделювання така система здатна суттєво пришвидшити процес створення інтерфейсів та зменшити витрати на початкову розробку.

Разом з тим, її реалізація потребує глибокого розуміння не лише комп’ютерного бачення та коду, але й UX-дизайну, що є викликом як для окремого розробника, так і для цілої команди. Проте з огляду на динаміку розвитку ШІ, створення такої універсальної системи – лише питання часу

ВИСНОВКИ

Проведене теоретичне дослідження підтвердило актуальність і перспективність розробки систем автоматичної генерації інтерфейсів користувача на основі зображень макетів. В сучасних умовах, коли темпи створення програмного забезпечення невпинно збільшуються, а вимоги до його якості та адаптивності стають дедалі вищими, автоматизація буденних процесів дизайну та верстки є потребою. Розробка рішень, здатних перетворювати графічний макет у HTML/CSS/JS-код без участі людини, дає змогу не лише прискорити створення продукту, а й мінімізувати кількість помилок, які можуть виникати на етапі ручної розмітки інтерфейсу.

У роботі було проаналізовано теоретичну архітектуру подібної системи, визначено ключові функціональні модулі, зокрема компоненти розпізнавання, обробки та генерації коду. Оскільки проєкт є суто теоретичним, основна увага була приділена моделюванню логіки системи, вибору вхідних і вихідних форматів даних, умовним сценаріям роботи, а також опису можливостей адаптації системи під різні цілі та потреби. Розглянуто типи вхідної графіки, специфіку HTML/CSS/JS як формату вихідного коду, а також структуру метаданих, які можуть бути корисними для точного позиціонування елементів. Також запропоновано узагальнену структуру моделі, що дає змогу зрозуміти потенційні напрями її розвитку та вдосконалення.

Особлива увага була приділена аналізу складнощів, які постають під час реалізації подібної системи. Найбільш значущими з них є потреба у великому обсязі навчальних даних – тисячах або навіть мільйонах прикладів пар «макет-код», що практично неможливо зібрати та обробити на індивідуальному рівні. Крім того, обчислювальні ресурси, необхідні для навчання потужної генеративної моделі, також можуть перевищувати можливості невеликої команди або одного дослідника. Це обмежує реальні

можливості втілення такого проєкту у короткостроковій перспективі, проте не зменшує його потенційної цінності для майбутнього.

У контексті правових та етичних аспектів було розглянуто питання авторського права, обробки персональних даних, а також можливості використання даних, згенерованих штучним інтелектом, у комерційних продуктах. Враховуючи стрімкий розвиток технологій, законодавча база не завжди встигає за новими викликами. Як зазначено в [22] джерелі, створення надлюдського ШІ ставить нові запитання щодо відповідальності, авторства та безпеки, тому питання етики та регулювання у сфері UI-генерації мають вирішальне значення.

Узагальнюючи результати, можна стверджувати, що теоретичне моделювання системи генерації інтерфейсів зображень за допомогою ШІ є важливим кроком до майбутньої реалізації повноцінних рішень. Хоча практичне втілення наразі пов'язане з низкою обмежень, закладені в цій роботі концепції дозволяють сформувати бачення напрямів подальших досліджень: від вибору архітектур до збору даних і забезпечення юридичної чистоти продукту. Подальші дослідження можуть бути зосереджені на зменшенні вимог до обсягів даних, удосконаленні генеративних моделей та створенні відкритих фреймворків для навчання ШІ у сфері UI-дизайну.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Баранов О. А. Інтернет речей (IoT): мета застосування та правові проблеми: Інформація і право. 2018. С. 31–44.

2. Зайченко Ю. П. Обчислювальний інтелект (результати, проблеми, перспективи): І міжнар. наук.-техн. конф. : матеріали першої міжнар. наук.-техн. конф. *«Обчислювальний інтелект (ОІ-2011)»*, 10–13 трав. 2011 р., м. Черкаси, Україна. 2011. 510 с.

3. Маркович Р., Ляо Б., Ван Ю. Н. Логіка для штучного інтелекту нового покоління. College Publications, 2022. 120 с.

4. МС. Штучний інтелект та UX: чому штучному інтелекту потрібен користувацький досвід. Apress, 2020. 179 с.

5. Авторське право та ШІ: хто власник і як захистити творчість. Читомо. URL: https://chytomo.com/avtorske-pravo-ta-shi-khto-vlasnyk-i-iak-z-=akhystyty-tvorchist/ (дата звернення: 30.04.2025).

6. Авторські права на об'єкт, створений штучним інтелектом. ЮРЛІГА. URL: https://jurliga.ligazakon.net/analitycs/225383_avtorsk-prava-na-obkt-stvoreniy-shtuchnim-ntelektom (дата звернення: 01.05.2025).

7. Про авторське право і суміжні права: Закон України від 01.12.2022 № 2811-IX : станом на 15.11.2024. URL: https://zakon.rada.gov.ua/laws/show/2811-20#Text (дата звернення: 25.04.2025).

8. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI : станом на 18.01.2025. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 25.04.2025).

9. Про схвалення Концепції розвитку штучного інтелекту в Україні: Розпорядж. КМУ від 02.12.2020 № 1556-р : станом на 29.12.2021. URL: https://zakon.rada.gov.ua/laws/show/1556-2020-p#Text (дата звернення: 25.04.2025).

10. Що таке штучний інтелект: історія, види та складові – GigaCloud. URL: https://gigacloud.ua/articles/shho-take-shtuchnyj-intelekt-istoriya-vydy-ta-skladovi/ (дата звернення: 26.04.2025).

11. Як працює генерація зображень за допомогою AI: детальний огляд. Комп'ютерна школа Hillel. URL: https://blog.ithillel.ua/articles/how-ai-image-generation-works (дата звернення: 23.04.2025).

12. Artificial intelligence and image analysis/ ed. by R.P. Barneva et al. Cham: Springer Nature Switzerland, 2024. URL: https://doi.org/10.1007/978-3-031-63735-3 (дата звернення: 01.05.2025).

13. AutoDraw. URL: https://autodraw.com/ (дата звернення: 28.04.2025).

14. Built In. The future of AI: how AI is changing the world. URL: https://builtin.com/artificial-intelligence/artificial-intelligence-future (дата звернення: 01.05.2025).

15. CareerFoundry. Using AI for UX design is incredible! (an UX / AI introduction: 2024), 2023. *YouTube*. URL: https://www.youtube.com/watch?v=vyWFS4DPpqA (дата звернення: 24.04.2025).

16. Drawww app. URL: https://www.drawww.app/ (дата звернення: 29.04.2025).

17. Meitu Inc. URL: https://www.meitu.com/en/ (дата звернення: 30.04.2025).

18. Scribble diffusion. URL: https://scribblediffusion.com/ (дата звернення: 27.04.2025).

19. SKVOT. Штучний інтелект vs авторське право. URL: https://skvot.io/uk/blog/ai-and-copyright (дата звернення: 30.04.2025).

20. Team A. E. AI image generation, explained. AltexSoft. URL: https://www.altexsoft.com/blog/ai-image-generation/ (дата звернення: 24.04.2025).

21. YouTube / BBC News Україна. Надлюдський штучний інтелект. URL: https://www.youtube.com/watch?v=sUG-L87RxZs (дата звернення: 23.04.2025).