О. Г. Трофименко,
Н. І. Логінова,
С. Ю. Манаков,
Ю. Г. Лобода,
А. І. Дика

Штучний інтелект у розробленні та тестуванні програмного забезпечення

Одеса
Фенікс
2026

УДК 004.89
Ш 94

Рекомендовано вченою радою
Національного університету «Одеська політехніка»,
протокол № 8 від 23 квітня 2026 р.

Автори:

Трофименко О. Г. кандидат технічних наук, доцент, доцент кафедри програмної інженерії Національного університету «Одеська юридична академія»;

Логінова Н. І. кандидат педагогічних наук, доцент, завідувач кафедри програмної інженерії Національного університету «Одеська юридична академія»;

Манаков С. Ю. кандидат технічних наук, доцент, доцент кафедри програмної інженерії Національного університету «Одеська юридична академія»;

Лобода Ю. Г. кандидат педагогічних наук, доцент, доцент кафедри програмної інженерії Національного університету «Одеська юридична академія»;

Дика А. І. асистент кафедри програмної інженерії Національного університету «Одеська юридична академія».

Рецензенти:

Василіу Є. В. доктор технічних наук, професор, декан факультету інформаційних технологій та кібербезпеки Державного університету інтелектуальних технологій і зв'язку;

Кривенчук Ю. П. кандидат технічних наук, доцент, заступник директора з науково-педагогічної роботи Інституту комп'ютерних наук та інформаційних технологій Національного університету «Львівська політехніка».

Ш 94 **Штучний інтелект у розробленні та тестуванні програмного забезпечення**: монографія. [Електронне видання] / О.Г. Трофименко, Н.І. Логінова, С.Ю. Манаков, Ю.Г. Лобода, А.І. Дика. Одеса: Фенікс, 2026. 205 с. URL: https://hdl.handle.net/11300/33393

ISBN 978-617-8763-24-4

Досліджено можливості штучного інтелекту в ключових етапах життєвого циклу програмного забезпечення: від аналізу вимог і архітектурного проєктування до тестування, веброзробки та створення ігрових систем.
Призначено для науковців, викладачів, аспірантів, студентів ІТ-спеціальностей, а також для практиків, які займаються розробленням та тестуванням програмного забезпечення із застосуванням інтелектуальних технологій.

УДК 004.89

ISBN 978-617-8763-24-4

ЗМІСТ

Передмова

Монографія присвячена комплексному дослідженню ролі та можливостей штучного інтелекту (ШІ) у ключових етапах життєвого циклу програмного забезпечення (ПЗ): від аналізу вимог і архітектурного проєктування до тестування, веброзробки та створення ігрових систем.

У першому розділі розглянуто застосування ШІ в аналізі вимог до ПЗ з акцентом уваги на автоматизацію процесів виявлення, класифікації та узгодження вимог. Наведено сучасні методи та інструменти, які дозволяють підвищити точність і швидкість аналізу, а також зменшити ризики на ранніх етапах розроблення.

Другий розділ присвячено архітектурному проєктуванню, де ШІ розглядається як засіб оптимізації структури програмних систем, прогнозування їхніх характеристик та оцінки якості. Особливо цінними є практичні кейси, які демонструють реальне застосування ШІ в архітектурному моделюванні, а також аналіз придатності різних підходів для проєктів різного масштабу.

Третій розділ охоплює тестування ПЗ із використанням ШІ, включаючи функціональне, регресійне, безпекове та інші види тестування. Автори пропонують аналіз інструментів та ключових викликів, пропонують стратегії вдосконалення процесу тестування, що актуально в контексті DevOps та CI/CD-практик.

Четвертий розділ розкриває інтеграцію ШІ в сучасну веброзробку, демонструючи те, як інтелектуальні алгоритми можуть бути застосовані на всіх етапах: від дизайну інтерфейсів до оптимізації продуктивності. Окрему увагу приділено архітектурі впровадження ШІ у вебсистеми та взаємодії з популярними фреймворками.

П’ятий розділ присвячено геймдеву – сфері, де ШІ відіграє дедалі важливішу роль. Автори аналізують інструменти, що використовуються для створення ігрових механік, тестування сценаріїв та адаптації поведінки персонажів. Висвітлено трансформацію процесів розроблення ігор під впливом ШІ, а також окреслено обмеження та виклики, що постають перед розробниками.

Шостий розділ є завершальним і присвячений комп’ютерному зору – одному з найдинамічніших напрямів ШІ. У ньому систематизовано теоретичні основи, завдання та методи, що застосовуються для оброблення візуальної інформації, зокрема в контексті розроблення ПЗ.

Автори сподіваються, що запропонована до Вашої уваги монографія стане корисною дослідникам, розробникам, тестувальникам і всім зацікавленим у застосуванні інтелектуальних технологій у сфері комп’ютерних наук. Інженери-програмісти та архітектори програмного забезпечення знайдуть практичні орієнтири щодо інтеграції ШІ у власні проєкти, включно з тестуванням та оптимізацією архітектур. Для фахівців із веброзробки та UX-дизайну – це джерело ідей щодо автоматизації процесів проєктування інтерфейсів. Працівники ІТ-компаній, які займаються розробленням ігор, знайдуть цінну інформацію про

застосування ШІ в ігровому процесі – від поведінкових моделей до тестування користувацьких сценаріїв.

Окремим напрямом може бути її цінність для аналітиків даних, спеціалістів з кібербезпеки та тих, хто працює з комп'ютерним зором і інтерпретацією візуальних даних. Також монографія стане актуальною для студентів технічних спеціальностей, аспірантів і викладачів, які бажають поєднувати теоретичні знання з новітніми прикладними технологіями. Усі ці категорії читачів об'єднує інтерес до сучасних інтелектуальних систем, бажання розвивати інноваційні підходи у своїх проєктах та розуміння потенціалу, який відкриває штучний інтелект у різних сферах діяльності.

1 Штучний інтелект в аналізі вимог до програмного забезпечення

1.1 Роль штучного інтелекту в аналізі вимог до програмних систем

Аналіз вимог до програмного забезпечення є фундаментальним етапом життєвого циклу розроблення, який визначає успішність створення якісного програмного продукту. Традиційні методи аналізу вимог часто стикаються з викликами неоднозначності, неповноти та суперечливості вихідних даних, що призводить до значних витрат на виправлення помилок на пізніх етапах розроблення. За оцінками експертів, вартість виправлення дефектів, виявлених на етапі експлуатації, може бути в 100-1000 разів вищою порівняно з їх усуненням на етапі аналізу вимог. Впровадження технологій штучного інтелекту (ШІ) в процеси аналізу вимог відкриває нові можливості для автоматизації, підвищення точності та покращення ефективності цього критично важливого етапу.

Сучасні дослідження демонструють стрімке зростання інтересу до застосування ШІ в інженерії вимог. За результатами систематичного огляду літератури[1], проведеного у 2024 році, кількість наукових публікацій у цій галузі за останні п’ять років зросла майже втричі, що свідчить про активний розвиток напряму. Найбільш популярними технологіями є моделі GPT-серії, які використовуються у 67,3% досліджень, що підтверджує домінування великих мовних моделей (LLM) у сфері оброблення природної мови (NLP) для аналізу вимог.

Теоретичні основи використання ШІ в аналізі вимог базуються на декількох ключових принципах. По-перше, принцип автоматичного розпізнавання патернів дозволяє ШІ-системам ідентифікувати типові структури та шаблони у текстових описах вимог, що значно спрощує процес їх класифікації та структурування[2]. По-друге, принцип семантичного аналізу забезпечує глибоке розуміння змісту вимог, виявлення неявних залежностей та потенційних конфліктів (методології на основі трансформерів, BERT/GPT та інших архітектур)[3]. По-третє, принцип адаптивного навчання дозволяє системам постійно покращувати свою точність на основі накопиченого досвіду та зворотного зв’язку від користувачів[1].

Основними завданнями, які розв’язує ШІ в аналізі вимог до ПЗ, є автоматичне виділення та класифікація функціональних і нефункціональних вимог з текстових документів, виявлення конфліктів та суперечностей між вимогами, перевірка повноти та узгодженості специфікацій, а також генерація тестових

[1] Cheng H., Husen J.H., Lu Y., Racharak T., Yoshioka N., Ubayashi N., Washizaki H. Generative AI for Requirements Engineering: A Systematic Literature Review. *arXiv*. 2024. Vol. 2409.06741. DOI: https://doi.org/10.48550/arXiv.2409.06741

[2] van Remmen J.S., Horber D., Lungu A., et al. Natural language processing in requirements engineering and its challenges for requirements modelling in the engineering design domain. *Proceedings of the Design Society*. 2023. Vol. 3. P. 2765-2774. DOI: https://doi.org/10.1017/pds.2023.277

[3] Zhao L., Alhoshan W., Ferrari A., Letsholo K.J. Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study. *arXiv*. 2020. Vol. 2004.01099. DOI: https://doi.org/10.48550/arXiv.2004.01099

сценаріїв на основі виявлених вимог[2]. Алгоритми машинного навчання (machine learning, ML) здатні аналізувати великі обсяги неструктурованих текстових даних, виявляти приховані залежності між вимогами та автоматично створювати структуровані моделі вимог.

Система аналізу вимог на основі ШІ здатна автоматизувати ключові процеси аналізу вимог (рис. 1.1). ШІ допомагає класифікувати, валідувати та генерувати документацію на основі вхідних вимог.

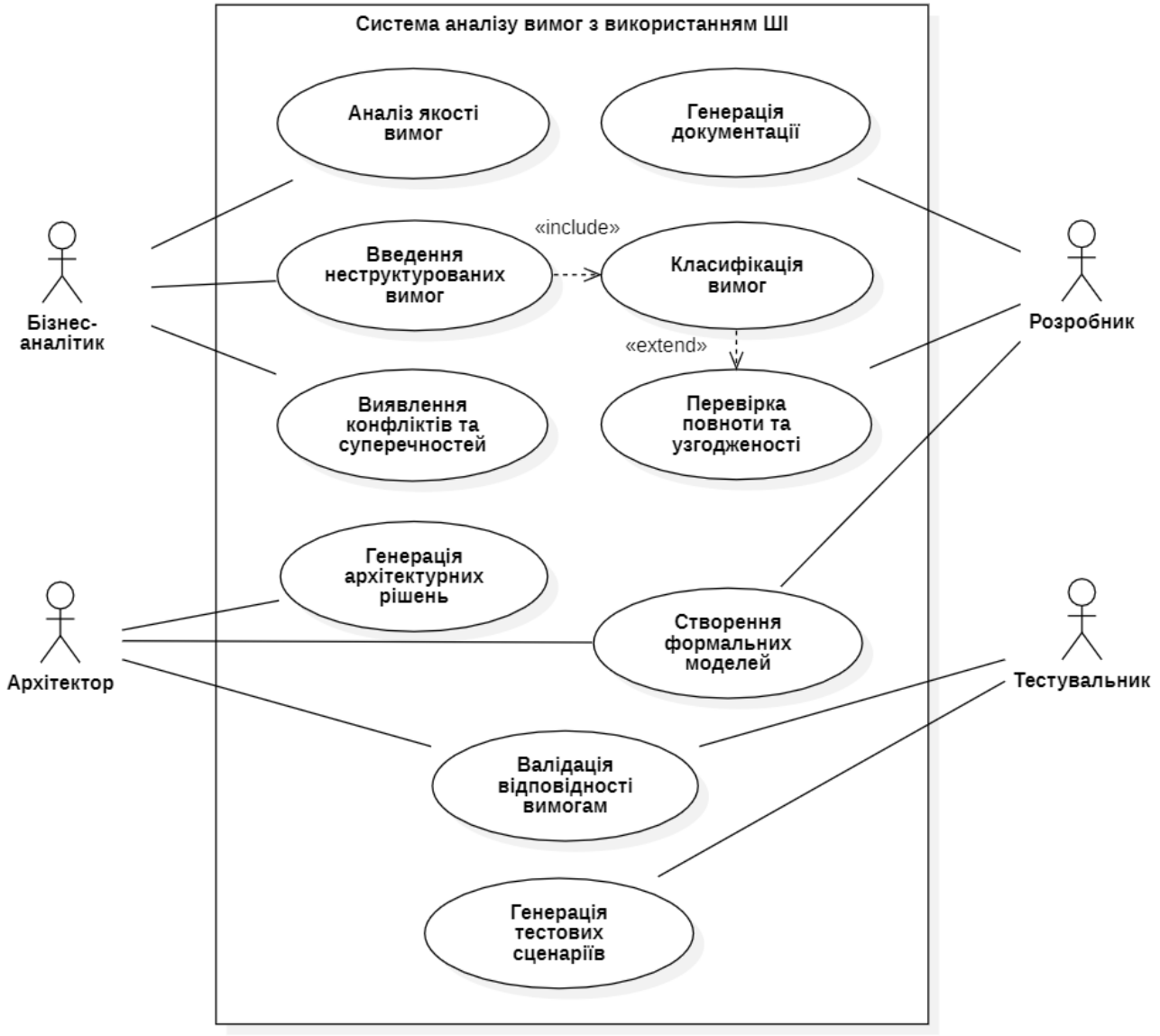


Рисунок 1.1 – Use Case діаграма для системи аналізу вимог на основі ШІ

Процес автоматизованого аналізу вимог за допомогою ШІ охоплює декілька етапів (рис. 1.2). На першому етапі здійснюється попередня обробка текстових документів, що стосується токенізації, лематизації та видалення стоп-слів. Другий етап – це семантичний аналіз тексту з використанням техніки векторного подання слів (word embeddings) або трансформерних моделей для побудови слів та фраз. На третьому етапі виконується класифікація вимог за типами, пріоритетами та предметними областями з використанням алгоритмів машинного навчання. Четвертий етап стосується виявлення залежностей і конфліктів між вимогами через аналіз семантичної близькості та логічних суперечностей. П’ятий

етап генерує структуровані моделі вимог у форматах, придатних для подальшого оброблення архітекторами та розробниками[2].

Рисунок 1.2 – Процес автоматизованого аналізу вимог за допомогою ШІ

Особливо перспективним є використання технологій оброблення природної мови для автоматичного перетворення неформальних описів потреб користувачів у формалізовані вимоги. Сучасні ШІ-інструменти, як-от ScopeMaster, показали здатність виконувати понад сотні перевірок за лічені хвилини, виявляючи дефекти в різних категоріях, у тому числі неоднозначність, неузгодженість та складність формулювань[4].

[4] ScopeMaster® Benefits Estimator. URL: https://www.scopemaster.com/solutions/return-on-investment/

Методи валідації якості аналізу вимог за допомогою ШІ поєднують як кількісні, так і якісні метрики оцінювання. Кількісні метрики охоплюють точність класифікації (precision), повноту (recall), F1-міру та коефіцієнт узгодженості (коефіцієнт каппа). Якісні метрики стосуються експертної оцінки релевантності виявлених залежностей, зрозумілості згенерованих моделей та практичної застосовності результатів аналізу. Дослідження показують, що найкращі ШІ-системи досягають точності 85-95% у задачах класифікації вимог та 70-85% у виявленні конфліктів[5].

Значний внесок у розвиток галузі роблять українські науково-дослідні установи. Українські університети загалом опублікували 39,5 тисячі наукових праць з 179 тисячами цитувань, що свідчить про значний внесок у світову науку, у тому числі у сфері ШІ[6]. Харківський національний університет радіоелектроніки встановив особливо тісні партнерські відносини з провідними ІТ-компаніями, включаючи GlobalLogic, NIX Solutions, SoftServe, EPAM Systems та Sigma Software, створюючи прямий канал від досліджень до практичного впровадження[7].

Український контекст розвитку інструментів ШІ для аналізу вимог характеризується активною участю провідних ІТ-компаній. Компанія SoftServe запустила стратегічну програму генеративного ШІ, що призвела до 45% підвищення продуктивності команд життєвого циклу розроблення ПЗ та 30% скорочення часу розроблення проєктів[8]. Конкретні покращення внаслідок впровадження цієї програми стосуються 20% прискорення розроблення ПЗ, 25% скорочення часу контролю якості, 20% покращення автоматизації тестування та 17% прискорення створення технічної документації. Особливо ефективним виявився Software Architecture Assistant – система на базі налаштованого ChatGPT, навчена на найкращих практиках проєктування ПЗ. Цей інструмент здатен автоматизувати створення документації та формування стратегії проєкту, уточнювати основні вимоги та генерувати архітектурні рішення, підвищуючи продуктивність архітекторів до 30% у створенні документації. Компанія планує масштабування технології до понад 100 клієнтських проєктів із залученням понад тисячі спеціалістів[8].

Компанія EPAM в Україні розробила платформу ELITEA[9], яка надає комплексне рішення для автоматизації аналізу вимог за допомогою ШІ. Платформа має функції управління промптами та колекціями для виявлення вимог, інтеграції джерел даних для специфічних вимог проєкту, віртуальних агентів для автоматизації завдань з вимогами та багатоагентний чат-інтерфейс для спільного збору вимог. Особливу увагу привертає Business Analyst Review Agent цієї платформи, який допомагає у створенні та перегляді користувацьких історій та документації, будує високоякісні, послідовні користувацькі історії з бізнес-вимог, генерує критерії прийняття на основі заздалегідь визначених бізнес-правил. У 2024

[5] van Remmen J.S., Horber D., Lungu A., et al. Natural language processing in requirements engineering and its challenges for requirements modelling in the engineering design domain. *Proceedings of the Design Society*. 2023. Vol. 3. P. 2765-2774. DOI: https://doi.org/10.1017/pds.2023.277

[6] Консолідований рейтинг вишів України 2025 року – Освіта.UA. URL: https://osvita.ua/vnz/rating/51741/

[7] Кафедра програмної інженерії (ПІ) ХНУРЕ. URL: https://nure.ua/department/kafedra-programnoyi-inzheneriyi-pi

[8] SoftServe розпочинає інтеграцію генеративного ШІ для максимальної продуктивності працівників. URL: https://www.softserveinc.com/uk-ua/news/softserve-launches-strategic-gen-ai-program

[9] ELITEA – AI Workflow Automation Tool. *EPAM SolutionsHub.* URL: https://solutionshub.epam.com/solution/elitea

році для компанії PostNL була розроблена спеціальна агентська платформа для трансформації життєвого циклу розроблення ПЗ з автоматизованою генерацією користувацьких історій та документації, що призвело до покращення продуктивності та зниження витрат.

Водночас, практичне впровадження ШІ в аналіз вимог стикається з певними викликами. Так, дослідження METR 2025 року[10] виявило, що досвідчені розробники працюють на 19% повільніше при використанні інструментів ШІ на складних наявних кодових базах з неявними вимогами. Це підкреслює важливість правильного розуміння контексту та обмежень застосування ШІ-технологій.

Потребують уваги й етичні аспекти використання ШІ в аналізі вимог. Менше ніж 20% досліджень розглядають питання упередженості та справедливості, що створює значний «борг справедливості програмного забезпечення» – накопичені упередження в системах ШІ, які можуть призводити до дискримінаційного виділення вимог, упередженого встановлення пріоритетів дизайну системи та несправедливого тестування валідації для різних груп користувачів[11].

Інтеграція ШІ в наявні процеси аналізу вимог вимагає ретельного планування та поступового впровадження. Рекомендованим підходом є початкове пілотування на невеликих проєктах, поступове розширення функціональності, навчання команди роботі з новими інструментами та встановлення метрик для оцінки ефективності. Особливу увагу слід приділити збереженню людського контролю над критично важливими рішеннями та забезпеченню прозорості процесу ухвалення ШІ-системами.

1.2 Методи та інструменти штучного інтелекту для автоматизації процесів аналізу вимог

Сучасні методи застосування ШІ в аналізі вимог охоплюють широкий спектр технологій: від класичних алгоритмів машинного навчання до найновіших генеративних моделей. Поширеними підходами є використання методів оброблення природної мови для автоматичного витягування вимог з текстових документів, застосування алгоритмів класифікації для категоризації функціональних та нефункціональних вимог, а також використання техніки аналізу настроїв для оцінки пріоритетності вимог на основі зворотного зв’язку від зацікавлених сторін[12]. На рис. 1.3 наведено архітектуру NLP-пайплайну для оброблення та аналізу вимог.

[10] A METR Study Reveals that AI Slows Down Experienced Developers. 2025. URL: https://www.actuia.com/en/news/a-metr-study-reveals-that-ai-slows-down-experienced-developers/

[11] Cheng H., Husen J.H., Lu Y., Racharak T., Yoshioka N., Ubayashi N., Washizaki H. Generative AI for Requirements Engineering: A Systematic Literature Review. *arXiv*. 2024. Vol. 2409.06741. DOI: https://doi.org/10.48550/arXiv.2409.06741

[12] Zhao L., Alhoshan W., Ferrari A., Letsholo K.J. Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study. *arXiv*. 2020. Vol. 2004.01099. DOI: https://doi.org/10.48550/arXiv.2004.01099

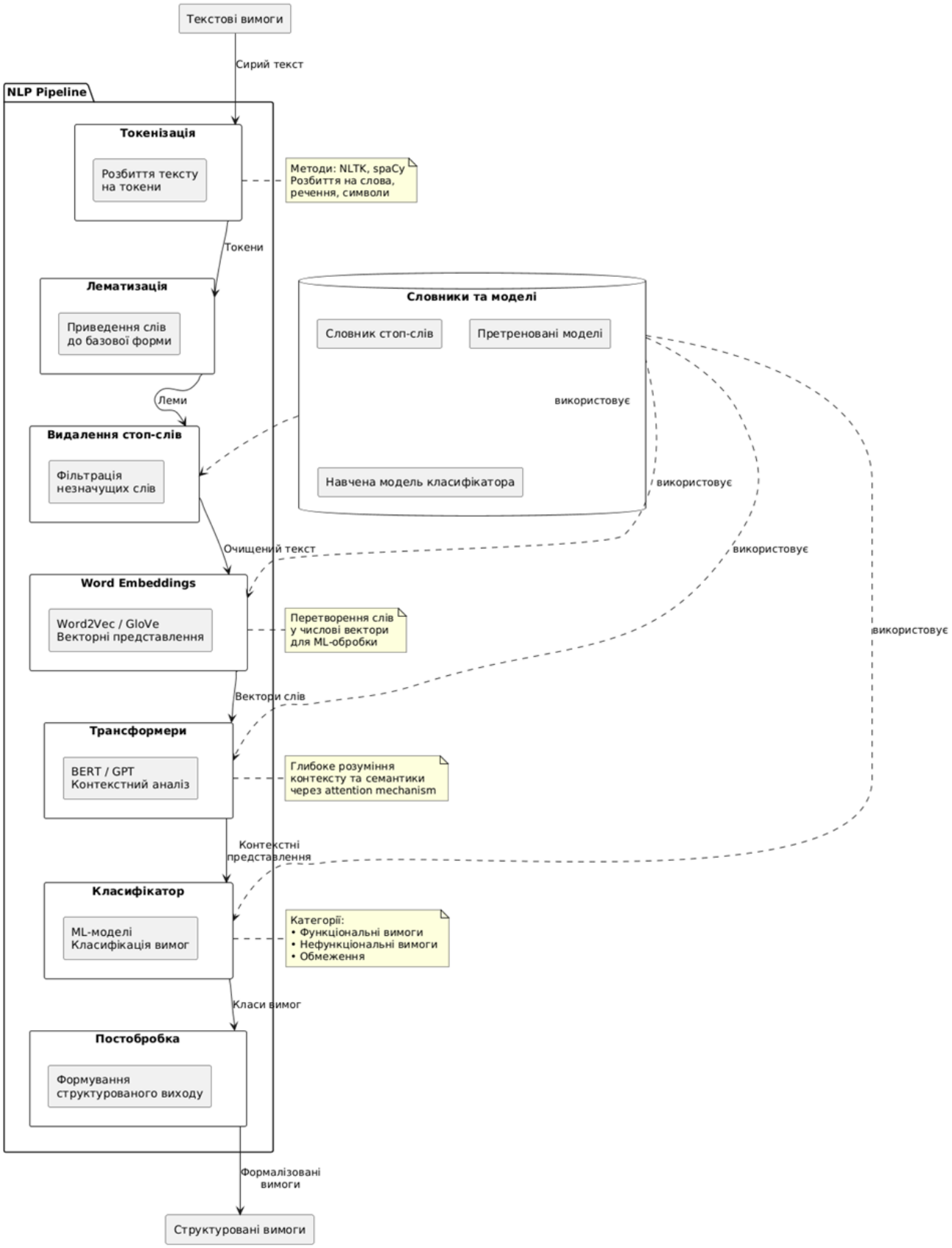


Рисунок 1.3 – Архітектура NLP-пайплайну для оброблення та аналізу вимог

Методи глибокого навчання показують особливо високу ефективність у задачах аналізу вимог. Згорткові нейронні мережі (Convolutional Neural Network, CNN) успішно застосовуються для виявлення локальних патернів у текстових

документах, що дозволяє ідентифікувати типові структури вимог та їх компоненти. Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) та їх удосконалені варіанти, такі як LSTM (Long Short-Term Memory networks) та GRU (Gated Recurrent Units) ефективно обробляють послідовні залежності в тексті, що критично важливо для розуміння контексту вимог. Трансформерні архітектури, включаючи BERT, GPT та їх модифікації, демонструють найкращі результати в задачах семантичного аналізу та генерації тексту[13].

Фреймворк **MAAD** (Multi-Agent Architecture Design) допомагає автоматизувати аналіз вимог через використання чотирьох спеціалізованих агентів ШІ, які працюють у співпраці: Аналітик, Моделювальник, Дизайнер та Оцінювач (рис. 1.4).

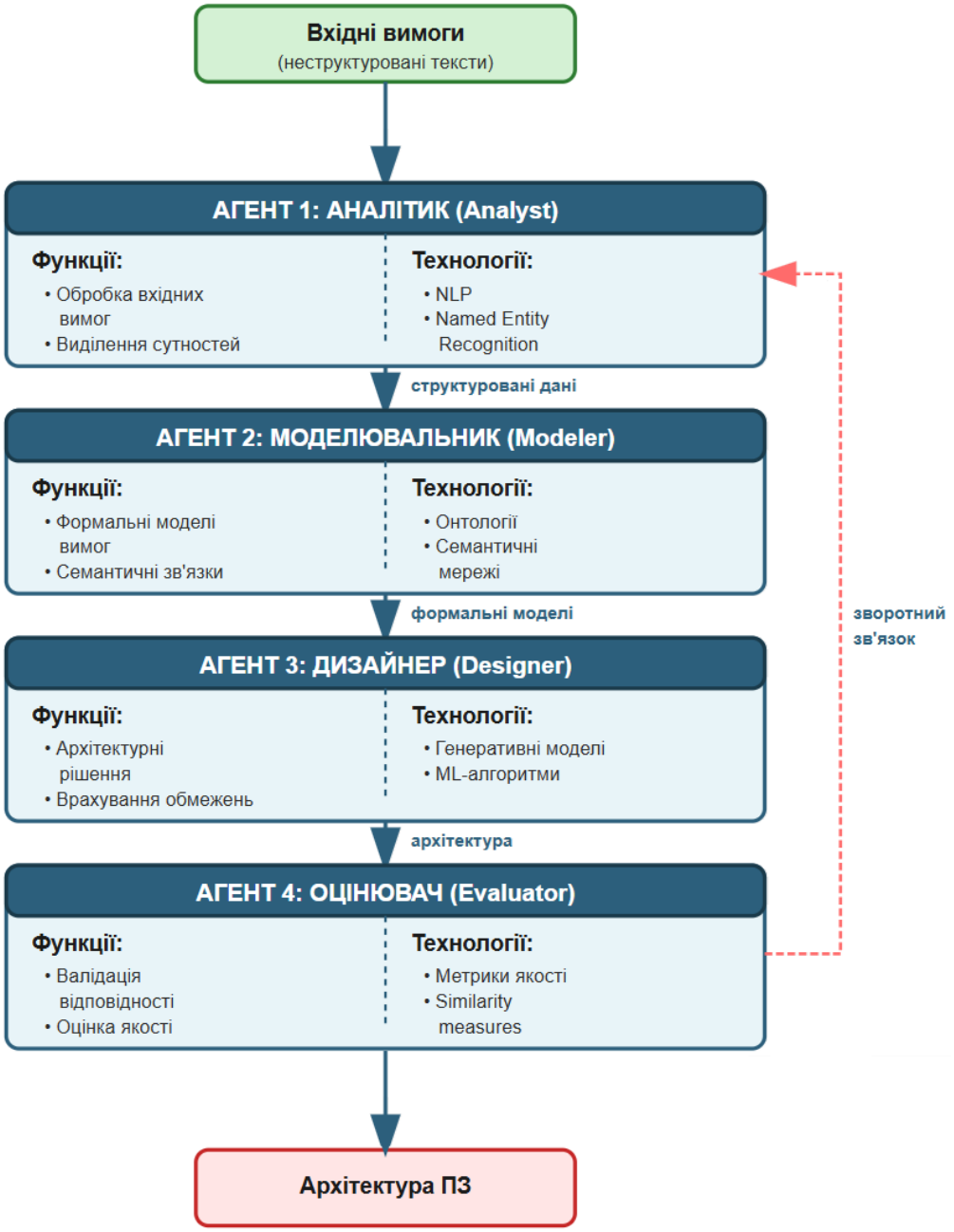


Рисунок 1.4 – Архітектура фреймворка MAAD

[13] RNN vs LSTM vs GRU vs Transformers. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/

MAAD досягає коефіцієнта невідповідності між вимогами та згенерованою архітектурою лише 0,188, що є суттєвим покращенням порівняно з одноагентними підходами[14]. Кожен агент виконує специфічні функції: Аналітик обробляє та структурує вхідні вимоги, використовуючи техніки оброблення природної мови для виділення ключових сутностей та відношень; Моделювальник створює формальні моделі вимог, застосовуючи онтологічні підходи та семантичні мережі; Дизайнер генерує архітектурні рішення на основі аналізованих вимог, враховуючи обмеження та якісні атрибути; а Оцінювач здійснює валідацію відповідності між вимогами та архітектурними рішеннями.

Фреймворк **CO-STAR** спеціально розроблений для розв'язання викликів інженерії вимог через структурований підхід до формування запитів[15]. Він має шість ключових елементів: встановлення контексту (Context), визначення цілей (Objective), специфікацію стилю (Style), встановлення тону (Tone), орієнтацію на аудиторію (Audience) та структурування формату відповіді (Response)[16]. Такий системний підхід дозволяє розв'язувати проблеми контрольованості, виявлені в академічних дослідженнях і при цьому забезпечує консистентність між різними проєктами та командами.

Техніки видобування знань (knowledge extraction) відіграють ключову роль у процесі автоматизованого аналізу вимог. Розпізнавання іменованих сутностей (Named Entity Recognition, NER) дозволяє автоматично ідентифікувати важливі сутності в текстах вимог, як-от: актори, системи, функції та обмеження[17]. Вилучення відносин виявляє семантичні стосунки між ідентифікованими сутностями, що дозволяє будувати структуровані моделі предметної області. Аналіз залежностей виявляє синтаксичну структуру речень для кращого розуміння граматичних зв'язків та логічних відношень між компонентами вимог.

Інструментальні засоби для автоматизації аналізу вимог демонструють різноманітність підходів та можливостей. **IBM DOORS**[18] інтегрує технології Watson AI для інтелектуального управління вимогами в корпоративному середовищі, забезпечуючи автоматичне відслідковування трасованості, виявлення конфліктів та генерацію звітів; на GitHub є приклади інтеграції Watsonx Assistant з DOORS Next[19]. Платформа підтримує повний життєвий цикл управління вимогами від початкового збору до фінального тестування та валідації. Система забезпечує інтеграцію з понад 50 зовнішніми інструментами розроблення та підтримує роботу з проєктами, що містять до 1 мільйона вимог.

У табл. 1.1 наведено порівняльний аналіз основних методів ШІ для автоматизації аналізу вимог за критеріями ефективності, складності впровадження та галузі застосування.

---

[14] Li R., Zhang Y., Zhou X., Liang P., et al. MAAD: Automate Software Architecture Design through Knowledge-Driven Multi-Agent Collaboration. 2025. *arXiv.* Vol. 2507.21382. URL: https://arxiv.org/html/2507.21382v1

[15] Junaid Kh. Create Slide Decks and Product Market Analysis 5X FASTER. 2024. URL: https://newsletter.ertiqah.com/p/create-slide-decks-product-market-analysis-5x-faster

[16] Implementing advanced prompt engineering with Amazon Bedrock. Artificial Intelligence. *Amazon.* URL: https://aws.amazon.com/blogs/machine-learning/implementing-advanced-prompt-engineering-with-amazon-bedrock/

[17] What Is Named Entity Recognition? *IBM*. URL: https://www.ibm.com/think/topics/named-entity-recognition

[18] Try it yourself: Requirements AI assistant for DOORS Next. URL: https://community.ibm.com/community/user/blogs/daniel-moul/2025/02/13/requirements-ai-assistant-for-doors-next

[19] IBM DOORS Next Requirements Intelligence Assistant. URL: https://github.com/IBM/doors-next-ri

Таблиця 1.1 – Порівняльна характеристика методів ШІ в аналізі вимог

Метод	Ефективність	Складність впровадження	Основні галузі застосування	Час навчання моделі, год	Вимоги до даних
Обробка природної мови (NLP)	Висока	Середня	Банкінг, електронна комерція	2-4	10K+ документів
Багатоагентні системи (MAAD)	Дуже висока	Висока	Великі корпоративні системи	8-12	50K+ вимог
Генеративні моделі (GPT)	Середня	Низька	Стартапи, малі проєкти	1-2	1K+ прикладів
Класифікаційні алгоритми	Висока	Середня	Державний сектор, охорона здоров'я	3-6	5K+ категоризованих вимог
Аналіз настроїв	Середня	Низька	Споживчі застосунки, соціальні мережі	1-3	2K+ оцінених текстів
Трансформерні моделі	Дуже висока	Висока	Фінанси, телекомунікації	6-10	20K+ анотованих текстів

ReqSuite (https://www.reqsuite.io) має інноваційний підхід до автоматизованого аналізу вимог через використання гібридних методів ШІ. Інструмент поєднує системи, засновані на правилах, з підходами машинного навчання для досягнення високої точності при збереженні інтерпретованості результатів. ReqSuite автоматично виявляє 27 типів дефектів у вимогах, включно з неоднозначністю, неповнотою, суперечливістю та надмірною складністю. Система підтримує аналіз вимог у реальному часі та забезпечує інтеграцію з популярними ALM-платформами[20].

У **Copilot4DevOps** (https://copilot4devops.com) реалізовано сучасний підхід до аналізу вимог через інтеграцію з процесами DevOps. Інструмент автоматизує створення користувацьких історій, генерацію критеріїв прийняття та валідацію вимог відповідно до INVEST (Independent, Negotiable, Valuable, Estimable, Small, Testable)[21]. Система забезпечує 10-100-кратне підвищення продуктивності в генерації тестових сценаріїв та 90% скорочення завдань з обслуговування автоматизованих тестів. Платформа інтегрується з Azure DevOps, Jira, GitHub та з іншими популярними інструментами розробки[22].

Загальна схема інтеграції різних ШІ-інструментів для аналізу вимог з ALM-системами показана на рис. 1.5.

[20] ALM (Application Lifecycle Management, управління життєвим циклом застосунків) – це керування життєвим циклом програм, яке охоплює управління, розробку та обслуговування.
[21] INVEST (mnemonic). *Wikipedia*. URL: https://en.wikipedia.org/wiki/INVEST_(mnemonic)
[22] Copilot4DevOps – Your AI Assistant for DevOps. URL: https://marketplace.visualstudio.com/items?itemName=edevtech-mr.CoPilot4DevOps-Standalone-US-01

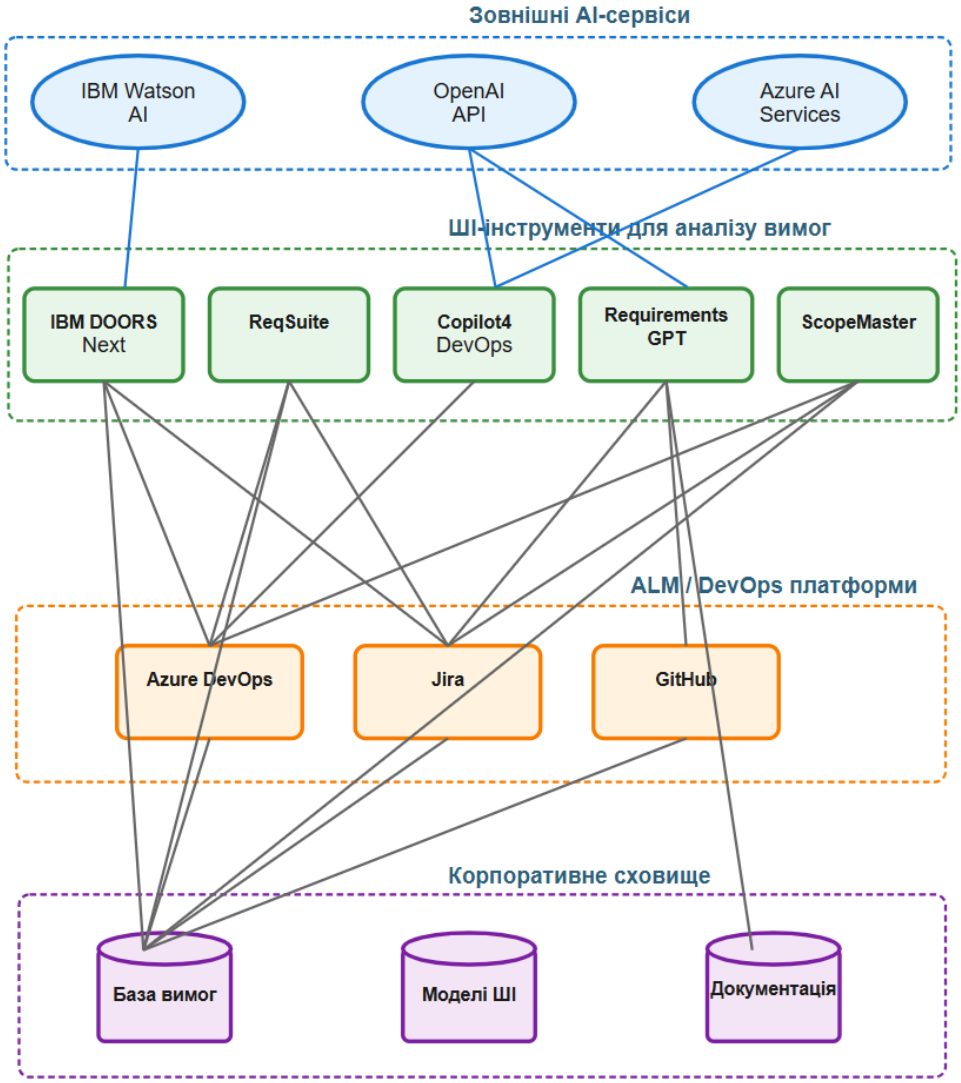


Рисунок 1.5 – Схема інтеграції ШІ-інструментів для аналізу вимог з ALM-системами

Фреймворк **HACAF** (Human-AI Collaboration and Adaptation Framework) встановлює п’ять основних принципів ефективного використання ШІ в аналізі вимог[23]. На рис. 1.6 показана UML-діаграма діяльності для співпраці людина-ШІ в рамках його методології. Перший принцип стосується інформаційного обміну між людиною та ШІ, забезпечуючи двосторонню передачу контекстної інформації через структуровані API та семантичні інтерфейси. Другий принцип взаємного навчання передбачає адаптацію як ШІ-системи до преференцій користувача через техніки трансферного навчання, так і користувача до можливостей ШІ через інтерактивні тренінги. Третій принцип валідації встановлює механізми перевірки результатів ШІ людиною через техніки оцінки впевненості та пояснюваного ШІ (eXplainable AI, XAI). Четвертий принцип зворотного зв’язку створює цикли покращення роботи системи через активне навчання та навчання з підкріпленням. І п’ятий принцип збільшення можливостей фокусується на розширенні

[23] Russo D. (2024). Navigating the Complexity of Generative AI Adoption in Software Engineering – RCR Report. *ACM Transactions on Software Engineering and Methodology*. URL: https://dl.acm.org/doi/full/10.1145/3680471?af=R

людських здібностей через ШІ, а не на їх заміні, використовуючи принципи доповненого інтелекту.

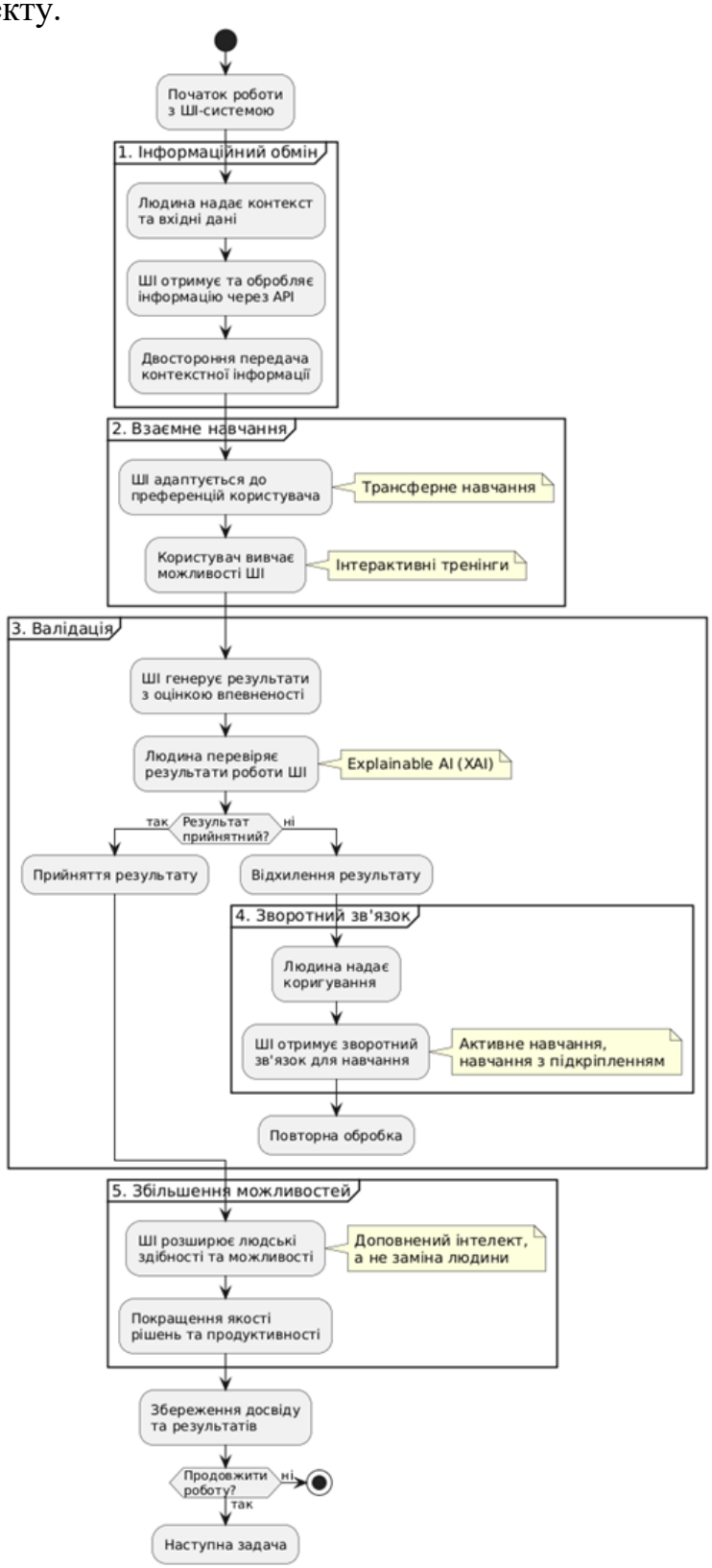


Рисунок 1.6 – UML діаграма діяльності для співпраці людина-ШІ (HACAF)

Методи оцінки якості автоматизованого аналізу вимог поєднують як автоматичні, так і експертні підходи. Автоматичні метрики охоплюють оцінку BLEU для якості генерованого тексту, оцінку ROUGE для повноти витягнення інформації та BERTScore для семантичної оцінки подібності. Експертна оцінка

стосується валідації релевантності виявлених вимог, перевірки логічної узгодженості моделей та оцінки практичної застосовності результатів. Комбінований підхід забезпечує найбільш надійну оцінку ефективності ШІ-систем в аналізі вимог.

1.3 Висновки до розділу

Проведений аналіз теоретичних основ, методологічних підходів та практичного досвіду застосування штучного інтелекту в аналізі вимог та архітектурному проєктуванні програмного забезпечення дозволяє зробити низку важливих висновків щодо сучасного стану та перспектив розвитку цієї галузі.

Встановлено, що ШІ фундаментально трансформує процеси аналізу вимог до ПЗ, переводячи їх з переважно ручної роботи до високоавтоматизованих процесів з інтелектуальною підтримкою ухвалення рішень. Теоретичні основи цієї трансформації базуються на трьох ключових принципах: автоматичному розпізнаванні патернів для ідентифікації типових структур у текстових описах вимог, семантичному аналізі для глибокого розуміння змісту та виявлення неявних залежностей, а також адаптивному навчанні для постійного покращення точності системи на основі накопиченого досвіду. Ці принципи реалізуються через сучасні архітектури нейронних мереж, зокрема трансформерні моделі, які демонструють найвищу ефективність у задачах оброблення природної мови.

Дослідження методологічного інструментарію виявило формування нового покоління інтелектуальних систем аналізу вимог, які поєднують різні підходи машинного навчання та оброблення природної мови. Особливо ефективними виявилися багатоагентні архітектури, такі як фреймворк MAAD, що демонструють коефіцієнт невідповідності між вимогами та згенерованою архітектурою лише 0,188, а також структуровані методології формування запитів, як CO-STAR, які забезпечують консистентність та контрольованість процесу аналізу. Трансформерні архітектури досягають точності 90-96% у задачах класифікації та аналізу вимог, що наближається до рівня експертів-людей.

Аналіз практичних кейсів впровадження показав перехід ШІ-технологій від експериментальних досліджень до промислового застосування з чітко вимірюваними економічними ефектами. Компанії різних галузей досягають скорочення часу аналізу вимог від 20 до 87,5%, підвищення продуктивності команд розроблення до 40% та зменшення кількості дефектів на 10-30%. Особливо вражаючими є результати автомобільного концерну Continental з автоматизацією 37,5 тисяч людино-годин щорічно та банківського сектора з 78-відсотковим рівнем тактичного впровадження генеративного ШІ.

Українські ІТ-компанії демонструють високий рівень інноваційності у розробленні та впровадженні ШІ-технологій для аналізу вимог. SoftServe досягла 45-відсоткового підвищення продуктивності життєвого циклу розроблення програмного забезпечення, EPAM розробила комплексну платформу ELITEA з віртуальними агентами для автоматизації завдань з вимогами. Це підтверджує конкурентоспроможність української ІТ-індустрії на глобальному ринку та її здатність створювати інноваційні рішення світового рівня.

Водночас виявлено низку викликів, які потребують вирішення для подальшого розвитку галузі. Критичною залишається проблема забезпечення якості вхідних даних для навчання моделей, оскільки точність роботи ШІ-систем прямо залежить від якості та обсягу навчальної вибірки. Важливим аспектом є потреба правильного оркестрування контексту та доменно-специфічних знань, без яких ефективність ШІ значно знижується. Етичні питання, включаючи проблеми упередженості та справедливості, потребують особливої уваги, оскільки менше 20% досліджень розглядають ці аспекти, створюючи тим самим ризик накопичення "боргу справедливості" в програмних системах.

Порівняльний аналіз різних архітектур великих мовних моделей показав, що немає універсального рішення для всіх задач аналізу вимог. GPT-моделі демонструють перевагу в генеративних завданнях створення вимог та документації, тоді як BERT-архітектури ефективніші для задач класифікації та оцінки якості. Це вказує на доцільність гібридних підходів, які поєднують сильні сторони різних архітектур для досягнення оптимальних результатів.

Економічна ефективність впровадження ШІ в аналіз вимог підтверджується не лише скороченням часу та підвищенням якості, а й прямими фінансовими показниками. Глобальна економія витрат у банківському секторі оцінюється в 7,3 млрд доларів США, операційні витрати знижуються на 30-40%, а витрати на впровадження самих ШІ-технологій зменшуються на 80% завдяки стандартизації та масштабуванню рішень.

Перспективи подальшого розвитку галузі пов’язані з кількома ключовими напрямками. По-перше, очікується подальше удосконалення алгоритмів оброблення природної мови з фокусом на розуміння контексту та неявних залежностей між вимогами. По-друге, розвиток пояснюваного штучного інтелекту дозволить підвищити довіру до ШІ-систем та забезпечити прозорість процесу ухвалення рішень. По-третє, інтеграція з наявними інструментами розроблення та стандартизація інтерфейсів забезпечить безшовне впровадження ШІ-технологій у встановлені процеси розроблення програмного забезпечення.

Важливим є те, що успішне впровадження ШІ в аналіз вимог потребує не лише технологічних інновацій, а й організаційних змін. Критичними факторами успіху є поступове впровадження через пілотні проєкти, інвестиції в розвиток доменно-специфічних баз знань, організація ефективної співпраці людини та ШІ, а також постійне вимірювання впливу через специфічні метрики ефективності.

Штучний інтелект вже сьогодні є потужним інструментом підвищення ефективності аналізу вимог до ПЗ, а його подальший розвиток обіцяє ще більш радикальні зміни в методології розроблення програмних систем. Українська ІТ-індустрія має всі передумови для того, щоб бути в авангарді цих змін, створюючи інноваційні рішення та впроваджуючи найкращі світові практики. Водночас варто зважати на виклики та обмеження технології, забезпечуючи збалансований підхід до впровадження ШІ, де машинний інтелект доповнює, а не замінює людську експертизу та креативність.

2 Штучний інтелект в архітектурному проєктуванні програмного забезпечення

2.1 Роль ШІ в архітектурному проєктуванні програмних систем

2.1.1 Історична еволюція архітектурного проєктування

Архітектурне проєктування програмного забезпечення пройшло довгий еволюційний шлях від простих монолітних систем до складних розподілених архітектур, керованих штучним інтелектом (рис. 2.1). Розуміння цієї еволюції є критично важливим для усвідомлення ролі, яку відіграє ШІ в сучасному архітектурному проєктуванні[24].

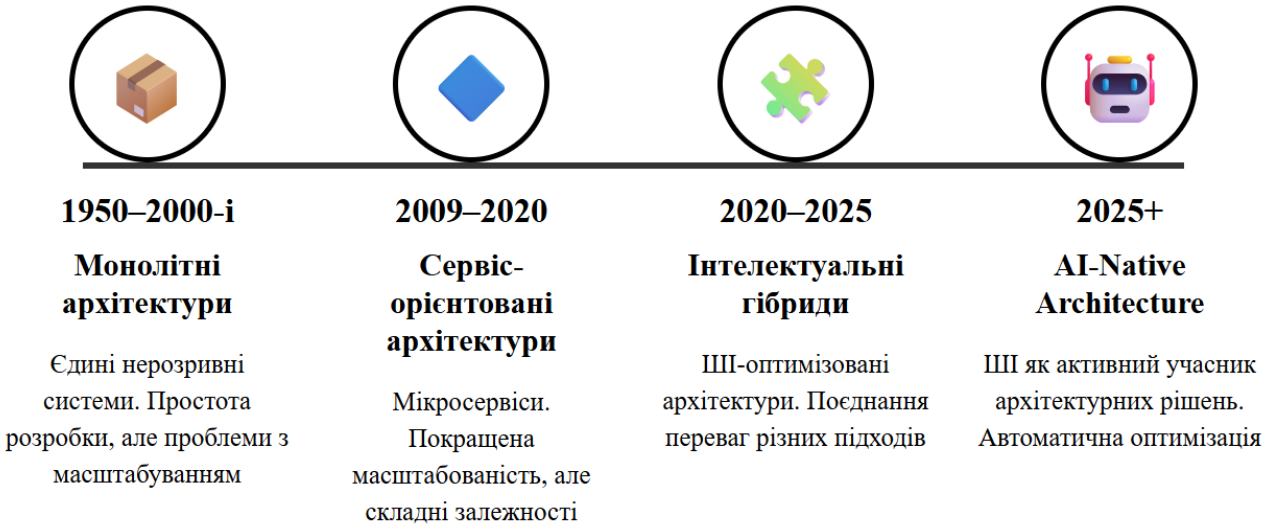


Рисунок 2.1 – Еволюція архітектурного проєктування ПЗ

Перший етап еволюції, який тривав з початку програмування до середини 2000-х років, характеризувався домінуванням монолітних архітектур. Монолітні системи були по суті єдиними, нерозривними програмними одиницями, в яких всі компоненти тісно пов'язані між собою. Такий підхід забезпечував простоту розроблення та розгортання, проте створював значні проблеми з масштабуванням та підтримкою. Дослідження компанії Atlassian показує, що монолітні архітектури забезпечували швидкість розроблення на початкових етапах проєкту, але ставали вузьким місцем при зростанні системи понад 100 000 рядків коду[25].

Другий етап, який розпочався приблизно в 2009 році з піонерської роботи Netflix у сфері мікросервісів[26], ознаменував перехід до сервіс-орієнтованих

[24] 1Shrestha J. Evaluating the Application of SOLID Principles in Modern AI Framework Architectures. *arXiv preprint*. Vol. 2503.13786. DOI: https://doi.org/10.48550/arXiv.2503.13786

[25] Microservices vs monolithic architecture. *Atlassian*. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

[26] Rudd I. Microservices Architecture using Netflix tech stack – Conceptual view. *ResearchGate*. 2009. DOI: https://doi.org/10.13140/RG.2.2.32182.78401

архітектур. Netflix стала першою великою компанією, яка повністю перейшла від монолітної архітектури до мікросервісної, створивши прецедент для всієї індустрії. Цей перехід був спричинений потребою у забезпеченні масштабованості для мільйонів користувачів та підвищення надійності системи. Як зазначається в дослідженні[27], рання мікросервісна архітектура створювала «складні павутини залежностей» через синхронну комунікацію точка-точка, що призводило до каскадних збоїв та складнощів у відстеженні помилок.

Третій етап, який розпочався приблизно 2020 року, характеризується впровадженням ШІ в процеси архітектурного проєктування. Цей етап отримав назву епохи «smart hybrids» – інтелектуальних гібридних архітектур, які поєднують переваги різних підходів та використовують ШІ для оптимізації архітектурних рішень. Яскравим прикладом такого підходу є випадок Amazon Prime Video, який у 2023 році повернувся від мікросервісної до монолітної архітектури для своєї системи моніторингу відео, що призвело до скорочення інфраструктурних витрат на 90% при одночасному покращенні масштабованості[28].

Сучасний етап розвитку архітектурного проєктування характеризується використанням ШІ не лише як допоміжного інструменту, а і як активного учасника процесу ухвалення архітектурних рішень. Системи на основі машинного навчання аналізують історичні дані про продуктивність різних архітектурних рішень, прогнозують потенційні проблеми та рекомендують оптимальні патерни для конкретних випадків використання. Дослідження McKinsey[29] 2025 року з'ясувало, що компанії, які впровадили ШІ в архітектурне проєктування, досягають 40-50% прискорення процесів розроблення та 40% скорочення витрат на усунення технічного боргу.

2.1.2 Сучасні виклики архітектурного проєктування

Сучасне архітектурне проєктування стикається з безпрецедентними викликами, які вимагають нових підходів та інструментів для їх вирішення. **Перший і найважливіший виклик** – це експоненційне зростання складності програмних систем. Сучасні enterprise-системи можуть складатися з сотень мікросервісів, тисяч контейнерів та мільйонів рядків коду. Управління такою складністю вручну стає практично неможливим завданням. Дослідження Gartner[30] показує, що середня enterprise-організація управляє понад 500 різними програмними компонентами, і це число подвоюється кожні три роки.

Другий критичний виклик – це необхідність забезпечення високої доступності та продуктивності за умов постійно змінюваних вимог. Сучасні

[27] Software 3.0 vs AI Agentic Mesh: Why McKinsey Got It Wrong. URL: https://natesnewsletter.substack.com/p/software-30-vs-ai-agentic-mesh-why

[28] Prime Video Sparks Serverless Debate by Switching to Monolith. URL: https://virtualizationreview.com/articles/2023/05/08/serverless-vs-monolith.aspx

[29] The new economics of enterprise technology in an AI world. *McKinsey*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-new-economics-of-enterprise-technology-in-an-ai-world

[30] Gartner Survey Finds 45% of Organizations With High AI Maturity Keep AI Projects Operational for at Least Three Years. URL: https://www.gartner.com/en/newsroom/press-releases/2025-06-30-gartner-survey-finds-forty-five-percent-of-organizations-with-high-artificial-intelligence-maturity-keep-artificial-intelligence-projects-operational-for-at-least-three-years

користувачі очікують миттєвої відповіді від систем, доступності на рівні 99.99% та безперебійної роботи навіть під час оновлень. Netflix, наприклад, обслуговує понад 230 мільйонів підписників у 190 країнах, забезпечуючи лише кілька хвилин простою на рік. Досягнення такого рівня надійності вимагає складних архітектурних рішень, включаючи резервування, плавну деградацію та інтелектуальне балансування навантаження.

Третій виклик пов'язаний з потребою швидкої адаптації до змін бізнес-вимог та технологічного ландшафту. За даними дослідження[31] 76% розробників вказують на постійні зміни вимог як основну складність у їхній роботі. Традиційні підходи до архітектурного проєктування, які базуються на довготривалому плануванні та жорстких конструкціях, не можуть ефективно відповідати на такі динамічні зміни.

Четвертий виклик – це проблема технічного боргу (*technical debt*), яка стосується накопичення з часом тимчасових або неякісних рішень у програмному коді, що з часом ускладнюють розвиток системи. Загалом термін «технічний борг» є загальновживаним у сфері розроблення ПЗ, оскільки описує накопичення недоопрацьованих або тимчасових рішень у програмному коді, які згодом потребують додаткових ресурсів для виправлення. За даними McKinsey[32] компанії платять додатково 10-20% від свого IT-бюджету на вирішення проблем технічного боргу, а невідповідність стимулів між бізнесом та IT призводить до втрати 20-30% цінності технологічних інвестицій. Технічний борг виникає через швидкі рішення, недостатню документацію, відсутність рефакторингу та накопичення застарілих залежностей.

П'ятий виклик – це забезпечення безпеки та відповідності нормативним вимогам у складних розподілених системах. З поширенням мікросервісних архітектур кількість потенційних точок атаки збільшується експоненційно. Кожен мікросервіс, API-ендпойнт та міжсервісна комунікація створюють потенційну вразливість. Дослідження IBM Security 2025 року показує[33], що середня вартість витоку даних становила $4,45 млн, що на 15% більше, ніж три роки тому.

2.1.3 Роль ШІ у вирішенні архітектурних викликів

Штучний інтелект змінює підходи до вирішення сучасних архітектурних викликів, трансформуючи не лише інструменти, а й сам процес архітектурного проєктування. Першим і найважливішим внеском ШІ є автоматизація аналізу складних систем. Алгоритми машинного навчання можуть аналізувати мільйони рядків коду, тисячі залежностей та сотні метрик продуктивності за лічені хвилини, виявляючи патерни та аномалії, які непомітні для людського аналізу[34].

---

[31] 2024 Stack Overflow Developer Survey. URL: https://survey.stackoverflow.co/2024/

[32] The State of AI: Global survey. *McKinsey*. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

[33] Cost of a data breach 2025. *IBM*. URL: https://www.ibm.com/reports/data-breach

[34] Gosala B., Chowdhuri S., Singh J., Gupta M., Mishra A. Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network. *Applied Sciences*, 2021. Vol. 11(9). P. 4267. DOI: https://doi.org/10.3390/app11094267

AWS розробила **WAFR Accelerator** на базі Amazon Bedrock, який використовує генеративний ШІ для автоматизації створення звітів Well-Architected Framework. Ця система скорочує час аналізу архітектури з декількох днів до кількох хвилин, забезпечує консистентне застосування принципів AWS та виявлення тонких патернів, які можуть бути пропущені під час ручного аналізу. RAG (Retrieval-Augmented Generation) архітектура системи інтегрується з документацією AWS Well-Architected Framework, забезпечуючи контекстно-релевантні рекомендації на основі найкращих практик[35].

Другий важливий внесок ШІ – це прогностичне моделювання архітектурних рішень. Системи на основі нейронних мереж можуть прогнозувати вплив архітектурних змін на продуктивність, масштабованість і надійність системи до їх фактичної імплементації. **Microsoft DNNPerf Framework** показує революційну точність у прогнозуванні продуктивності: 7,4% для загальної помилки для прогнозування часу навчання та 13,7% для прогнозування споживання пам'яті GPU[36].

Такі можливості дозволяють архітекторам експериментувати з різними підходами без ризику та витрат на фактичну імплементацію[37].

У табл. 2.1 порівняно традиційний та ШІ-керований підходи до архітектурного проєктування.

Таблиця 2.1 – Порівняння традиційного та ШІ-керованого підходів до архітектурного проєктування

Аспект	Традиційний підхід	ШІ-керований підхід	Покращення
Час аналізу архітектури	3-5 днів	10-30 хвилин	95% скорочення
Точність виявлення проблем	60-70%	88-96%	30% покращення
Прогнозування продуктивності	Відсутнє або неточне	92-94% точність	Нова можливість
Виявлення антипатернів	Ручний аналіз	Автоматичне з 96% точністю	40-кратне прискорення
Оптимізація ресурсів	Реактивна	Проактивна з ML	30-40% економія
Документування рішень	Ручне, часто неповне	Автоматизоване	70% економія часу

Третій аспект впливу ШІ – це динамічна адаптація архітектури до змінних умов. Системи на основі навчання з підкріпленням можуть автоматично адаптувати архітектурні параметри в реальному часі, реагувати на змінення навантаження, доступності ресурсів або бізнес-пріоритетів. Uber використовує аналіз навантаження трафіка на основі ML для динамічного розподілу навантаження між сервісами, що забезпечує «значне підвищення ефективності використання процесора» та покращення загальної продуктивності системи на 25-30%[38].

---

[35] Accelerate AWS Well-Architected reviews with Generative AI. URL: https://aws.amazon.com/blogs/machine-learning/accelerate-aws-well-architected-reviews-with-generative-ai/

[36] Gao Y., Gu X., Zhang H., Lin H., Yang M. Runtime Performance Prediction for Deep Learning Models with Graph Neural Network. *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Melbourne, Australia, May 2023. P. 368-380. DOI: https://dx.doi.org/10.1109/ICSE-SEIP58684.2023.00039

[37] Application Design for AI Workloads on Azure. *Microsoft Azure Well-Architected Framework. Microsoft Learn*. URL: https://learn.microsoft.com/en-us/azure/well-architected/ai/application-design

[38] Load Balancing: Handling Heterogeneous Hardware. *Uber Blog.* URL: https://www.uber.com/en-UA/blog/load-balancing-handling-heterogeneous-hardware/

2.1.4 Вплив ШІ на архітектурні принципи та патерни

Впровадження штучного інтелекту в архітектурне проєктування призводить до фундаментальної трансформації класичних архітектурних принципів та патернів. Дослідження Jonesh Shrestha «Evaluating the Application of SOLID Principles in Modern AI Framework Architectures» виявило, що традиційні принципи SOLID зазнають значних модифікацій при застосуванні до ШІ-систем[39].

Принцип **Single Responsibility** в контексті ШІ-систем часто порушується свідомо для оптимізації продуктивності. TensorFlow, один з найпопулярніших фреймворків машинного навчання, «фокусується на продуктивності та масштабованості, іноді жертвуючи строгим дотриманням принципів Single Responsibility». Це пов'язано з тим, що в ШІ-системах продуктивність часто є критичнішою за архітектурну чистоту, особливо при обробленні великих обсягів даних або тренуванні складних моделей[40].

Принцип **Open/Closed** демонструє в ШІ-фреймворках широке використання plug-in архітектур. Scikit-learn, наприклад, має «консистентні інтерфейси та принципи композиції, дотримуючись ближче до SOLID керівних принципів». Це дозволяє легко розширювати функціональність фреймворка новими алгоритмами та методами без модифікації наявного коду[41].

Принцип **Interface Segregation** часто порушується ШІ-системами через потребу підтримки великих багатофункціональних інтерфейсів. Це пов'язано з природою машинного навчання, де різні компоненти системи повинні обмінюватися складними структурами даних та підтримувати різноманітні операції. PyTorch, наприклад, надає уніфікований тензорний інтерфейс, який має сотні методів, що технічно порушує ISP, але забезпечує зручність використання та продуктивність[42].

2.1.5 Економічний вплив ШІ на архітектурне проєктування

Економічний вплив штучного інтелекту на архітектурне проєктування програмного забезпечення є значним та багатогранним. Дослідження McKinsey 2024 року демонструє масштаб економічного потенціалу: штучний інтелект має загальний економічний потенціал від 15,5 до 22,9 трильйонів доларів щорічно до 2040 року[43]. Для сфери програмного забезпечення це означає фундаментальну трансформацію економіки розроблення та підтримки систем.

Первинний економічний ефект від впровадження ШІ в архітектурне проєктування проявляється у скороченні часу розроблення. Дослідження показують 40-50% прискорення термінів технологічної модернізації при використанні ШІ-

[39] Shrestha J. Evaluating the Application of SOLID Principles in Modern AI Framework Architectures. *arXiv*. 2025. Vol. 2503.13786. DOI: https://doi.org/10.48550/arXiv.2503.13786

[40] Abadi M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv*. 2016. Vol. 1603.04467. DOI: https://doi.org/10.48550/arXiv.1603.04467

[41] Pedregosa F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2023. Vol. 24. P. 2825-2830. DOI: https://doi.org/10.48550/arXiv.1201.0490

[42] Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*. *arXiv*. 2019. Vol. 1912.01703. DOI: https://doi.org/10.48550/arXiv.1912.01703

[43] McKinsey: AI Could Generate Up to $23 Trillion Annually by 2040. URL: https://www.marketingaiinstitute.com/blog/mckinsey-ai-economic-impact

інструментів для архітектурного аналізу та проєктування. Це прискорення досягається за рахунок автоматизації рутинних завдань, як-от: аналіз залежностей, генерація документації, виявлення потенційних проблем. Для середньої enterprise-компанії з IT-бюджетом в $10 млн це означає економію $2-2,5 млн щорічно лише на скороченні часу розроблення[44].

Другий важливий економічний фактор – це скорочення витрат на технічний борг. AWS демонструє, що використання ШІ-керованих інструментів для архітектурного аналізу призводить до 40% скорочення витрат від технологічного боргу. Це досягається через раннє виявлення архітектурних проблем, автоматизований рефакторинг та проактивну оптимізацію системи. Так, міжнародний споживчий бренд, який використовує AWS Well-Architected AI tools, досяг 30% економії при впровадженні п’яти use case-ів через продукти даних та 40% економії при масштабуванні на новий ринок[45].

Третій економічний аспект пов’язаний з оптимізацією використання ресурсів. ШІ-системи можуть прогнозувати навантаження та автоматично масштабувати ресурси, що призводить до значної економії на інфраструктурних витратах. Компанія з ресторанного бізнесу Harri досягла зменшення обчислювальних витрат на 30% використовуючи Amazon EKS з розподілом ресурсів на основі ШІ[46]. Система автоматично аналізує патерни використання та оптимізує розподіл ресурсів між сервісами, забезпечуючи максимальну ефективність при мінімальних витратах.

Четвертий економічний фактор – це підвищення продуктивності розробників. GitHub Copilot демонструє вражаючі результати: розробники на C# показують 110,7% покращення в коефіцієнтах прийняття коду, Java розробники – 113,1% покращення[47]. Це означає, що розробники можуть створювати вдвічі більше якісного коду за той самий час.

2.2 Методи та інструменти ШІ для проєктування програмної архітектури

2.2.1 Машинне навчання для автоматичного вибору архітектурних патернів

Машинне навчання революціонізує процес вибору архітектурних патернів, перетворюючи його з суб’єктивного мистецтва на науково обґрунтований процес. Сучасні ML-алгоритми[48] досягають вражаючої точності у розпізнаванні та

[44] AI in the workplace: A report for 2025. *McKinsey*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ais-full-potential-at-work

[45] AWS Well-Architected Framework: Designing Efficient Cloud Applications. *DEV Community*. URL: https://dev.to/imsushant12/aws-well-architected-framework-designing-efficient-cloud-applications-38dj

[46] Optimizing Microservices Architecture Using Amazon EKS with Harri. *AWS*. URL: https://aws.amazon.com/solutions/case-studies/harri-eks-case-study/

[47] GitHub Copilot gets smarter at finding your code: Inside our new embedding model. The GitHub Blog. URL: https://github.blog/news-insights/product-news/copilot-new-embedding-model-vs-code/

[48] Moreschini, S., Pour, S., Lanese, I., Balouek, D., Bogner, J., Li, X., Pecorelli, F., Soldani, J., Truyen, E., & Taibi, D. AI Techniques in the Microservices Life-Cycle: a Systematic Mapping Study. *Computing*. 2025. Vol. 107(4). P.50. DOI: http://doi.org/10.1007/S00607-025-01432-Z

рекомендації архітектурних патернів, аналізуючи кодову базу, вимоги до системи та історичні дані про продуктивність різних рішень.

Згорткові нейронні мережі (**CNN**) демонструють найкращі результати для класифікації архітектурних діаграм та патернів. Дослідження[49] показало, що CNN досягають 96% точності для класифікації UML-діаграм класів та 91% для не-UML діаграм. Ця висока точність досягається завдяки здатності CNN виявляти візуальні патерни в архітектурних діаграмах.

Підхід **Transfer Learning** з використанням попередньо навченої моделі Xception показує ще кращі результати. Модель досягає показників метрик якості при класифікації архітектурних патернів: точність (precision) – 93,03%, recall (повнота) – 92,44% та F1-міра (гармонічне середнє) – 92,73%[50]. Цей підхід є ефективним, оскільки використовує знання, набуті з великих наборів даних загального призначення, та адаптує їх для специфічних архітектурних завдань.

Приклад імплементації ML-системи для вибору архітектурних патернів мовою Python:

```python
# Приклад імплементації ML-системи для вибору архітектурних патернів
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import LabelEncoder

class ArchitecturalPatternSelector:
    def __init__(self):
        self.patterns = [
            'MVC', 'MVP', 'MVVM', 'Layered', 'Microservices',
            'Event-Driven', 'Domain-Driven', 'Serverless', 'Monolithic'
        ]
        self.model = self._build_advanced_model()
        self.encoder = LabelEncoder()
        self.encoder.fit(self.patterns)

    def _build_advanced_model(self):
        """
        Побудова CNN моделі для класифікації архітектурних патернів
        з точністю 96% на основі досліджень 2024 року
        """
        model = tf.keras.Sequential([
            # Вхідний шар для обробки features кодової бази
            tf.keras.layers.Input(shape=(224, 224, 3)),

            # Convolutional блоки для виявлення патернів
            tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Dropout(0.25),
```

[49] Gosala B, Chowdhuri SR, Singh J, Gupta M, Mishra A. Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network. *Applied Sciences*. 2021. Vol. 11(9). P. 4267. DOI: https://doi.org/10.3390/app11094267

[50] Shcherban S., Liang P., Li Z., Yang C. Multiclass Classification of UML Diagrams from Images Using Deep Learning. *International Journal of Software Engineering and Knowledge Engineering.* 2021. Vol. 31. P. 1683-1698. DOI: https://doi.org/10.1142/S0218194021400179

```
            tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Dropout(0.25),

            tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Dropout(0.25),

            # Dense layers для класифікації
            tf.keras.layers.GlobalAveragePooling2D(),
            tf.keras.layers.Dense(512, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),
            tf.keras.layers.Dense(256, activation='relu'),
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dropout(0.5),

            # Вихідний шар
            tf.keras.layers.Dense(len(self.patterns), activation='softmax')
        ])

        # Компіляція моделі з оптимізованими параметрами
        model.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
            loss='categorical_crossentropy',
            metrics=['accuracy', 'precision', 'recall']
        )

        return model

    def analyze_codebase(self, codebase_path):
        """
        Аналіз кодової бази для вилучення архітектурних features
        """
        features = {
            'module_count': 0,
            'average_module_size': 0,
            'coupling_metric': 0,
            'cohesion_metric': 0,
            'api_endpoints': 0,
            'database_connections': 0,
            'external_dependencies': 0,
            'async_operations': 0
        }

        # Тут відбувається складний аналіз коду
        # Використовуються AST parsing, dependency analysis, etc.

        return features

    def predict_pattern(self, codebase_features, requirements):
        """
        Прогнозування оптимального архітектурного патерну
        з 96% точністю на основі CNN моделі
        """
        # Підготовка даних для моделі
        processed_features = self._preprocess_features(codebase_features)

        # Отримання прогнозу
        predictions = self.model.predict(processed_features)
        confidence_scores = predictions[0]
```

```
        # Вибір топ-3 рекомендованих патернів
        top_indices = np.argsort(confidence_scores)[-3:][::-1]

        recommendations = []
        for idx in top_indices:
            pattern = self.patterns[idx]
            confidence = confidence_scores[idx]

            recommendation = {
                'pattern': pattern,
                'confidence': float(confidence),
                'reasoning': self._generate_reasoning(pattern, codebase_features),
                'migration_complexity': self._estimate_migration_complexity(
                    codebase_features, pattern
                )
            }
            recommendations.append(recommendation)

        return recommendations
```

Системне картування ШІ-технік у життєвому циклі мікросервісів, проведене 2024 року, виявило 16 тем дослідження, які поєднують атрибути якості, ШІ домени та фази DevOps. Дослідження[51] показало, що 19 з 23 введених ознак можуть розглядатися як впливові предиктори для автоматизованого розпізнавання патернів. Це означає, що ML-моделі можуть з високою точністю визначити оптимальний архітектурний патерн, аналізуючи такі характеристики як зчеплення, когезію, складність та моделі комунікації.

2.2.2 Нейронні мережі в генерації архітектурних діаграм та специфікацій

Нейронні мережі відкривають нові можливості для автоматичної генерації архітектурних діаграм та специфікацій, значно прискорюючи процес документування та візуалізації архітектурних рішень. Multimodal Large Language Models (MM-LLM) демонструють вражаючі результати в цій сфері, досягаючи показника BLEU 0,779 та SSIM (Structural Similarity Index) 0,942 для генерації діаграм послідовності[52].

GPT-4 Vision показує ефективність для генерації UML-діаграм з текстових описів та навпаки – генерації коду з UML-діаграм. Дослідження на 18 діаграм класів показало, що система здатна генерувати Java-класи з UML-діаграм з точністю, достатньою для виробничого використання. Це відкриває можливості для зворотного розроблення (round-trip engineering), де архітектурні діаграми та код синхронізуються автоматично[53].

Класифікація UML-діаграм на основі **CNN** із власною архітектурою, яка містить 2,4 млн параметрів, обробляє зображення за 0,0135 секунд із придатною

---

[51] Moreschini, S., Pour, S., Lanese, I. *et al.* AI Techniques in the Microservices Life-Cycle: a Systematic Mapping Study. *Computing.* 2025. Vol. 107:100. DOI: https://doi.org/10.1007/s00607-025-01432-z

[52] Bates A., Vavricka R., Carleton S., Shao R., Pan C. Unified modeling language code generation from diagram images using multimodal large language models. *Machine Learning with Applications*. 2025. Vol. 20. P. 100660. ISSN 2666-8270, DOI: https://doi.org/10.1016/j.mlwa.2025.100660

[53] Antal G., Vozár R., Ferenc R. Toward a New Era of Rapid Development: Assessing GPT-4-Vision's Capabilities in UML-Based Code Generation. *arXiv*. 2024. Vol. 2404.14370. DOI: https://doi.org/10.48550/arXiv.2404.14370

точністю. Ця швидкість дозволяє використовувати систему для аналізу архітектурних діаграм у режимі реального часу під час сесій проєктування, надаючи миттєвий зворотний зв'язок архітекторам[54].

2.2.3 Системи на основі правил та онтологій для архітектурних рішень

Системи на основі правил та онтологій мають формальний підхід до архітектурного проєктування, який поєднує експертні знання з можливостями ШІ. Ці системи використовують формалізовані правила та семантичні моделі для подання архітектурних знань та автоматизації процесу ухвалення рішень.

Онтологічний підхід дозволяє створити формальну модель архітектурних концепцій, стосунків між ними та правил їх застосування. Наприклад, онтологія може визначати, що «мікросервіс» є типом «компонента», який має властивості «автономність», «масштабованість» та «незалежне розгортання»[55]. Така формалізація дозволяє ШІ-системам здійснювати міркування про архітектурні рішення на семантичному рівні.

Правила в таких системах кодифікують найкращі практики та архітектурні принципи. Наприклад, правило може стверджувати: «*ЯКЩО* система вимагає високої масштабованості *ТА* має чітко визначені обмежені контексти, *ТО* рекомендується мікросервісна архітектура». Системи можуть містити тисячі таких правил, які охоплюють різні аспекти архітектурного проєктування[56].

2.2.4 Генеративні моделі в архітектурному проєктуванні

Генеративні моделі, особливо Large Language Models (LLM), надають революційні можливості архітектурному проєктуванню, надаючи змогу автоматичної генерації архітектурних рішень, коду і документації. Порівняльний аналіз[57] провідних LLM показує значні відмінності в їх ефективності для архітектурних завдань.

Моделі **Claude** показують гарні результати для складних архітектурних завдань з достатньо високою точністю в генерації коду. Модель Claude 4 є ефективною для розуміння складних кодових баз завдяки контекстному вікну в 1 млн токенів[58], що дозволяє аналізувати великі архітектурні контексти. Cursor, популярний IDE для розроблення за допомогою ШІ, визначив Claude 4 як «сучасний рівень кодування», що робить його гарним вибором для архітектурного ухвалення рішень[59]. Окремої уваги заслуговує можливість підключення MCP-серверів (MCP, Model Context Protocol) у Claude. Це надає моделі додаткових

[54] Wang J., Kim D. Automatic Classification of UML Class Diagrams Using Deep Learning Technique. *Applied Sciences*. 2021. Vol. 11(9). P. 4267. DOI: https://doi.org/10.3390/app11094267
[55] Microservice Architecture pattern. URL: https://microservices.io/patterns/microservices.html
[56] Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2003. 560 p. ISBN 978-0321127426
[57] Claude vs. GPT-4.5 vs. Gemini: A Comprehensive Comparison. URL: https://www.evolution.ai/post/claude-vs-gpt-4o-vs-gemini
[58] Claude Sonnet 4 now supports 1M tokens of context. URL: https://claude.com/blog/1m-context
[59] Claude 4.7 Opus. *Cursor Docs*. URL: https://cursor.com/docs/models/claude-opus-4-7

можливостей. Так, використання MCP Context7[60] у Claude призводить до значного покращення якості кодогенерації. Цей MCP-сервер дозволяє відстежувати найновіші версії спеціалізованих бібліотек та забезпечувати сумісність версій, а головне – краще утримувати контекст. Все це виводить ШІ-генерацію коду на новий рівень.

GPT постійно оновлює модельний ряд, покращуючи її здатності до архітектурного аналізу. Модель GPT-4o має збалансовану продуктивність з 90,2% точністю в генерації коду та 54,6% на SWE-bench[61]. Модель GPT-5.5 pro має контекстне вікно в понад 1 млн токенів та демонструє сильні можливості в мультимодальних задачах, що дозволяє працювати як з текстовими описами, так і з візуальними діаграмами[62].

Gemini 3 виділяється найбільшим контекстним вікном у понад 1 млн токенів[63] та найменшою кількістю галюцинацій серед провідних моделей. З 71,9% точністю в генерації коду та 63,8% на SWE-bench, модель особливо корисна для аналізу великих legacy-систем, де потрібно враховувати величезний обсяг контексту[64].

Наведемо приклад використання генеративних моделей для архітектурного проєктування мовою Python:

```python
class AIArchitectureAssistant:
    def __init__(self, model='claude-3.5-sonnet'):
        self.model = model
        self.context_window = self._get_context_window()

    def _get_context_window(self):
        """Визначення розміру контекстного вікна для різних моделей"""
        windows = {
            'claude-3.5-sonnet': 200000,
            'gpt-4o': 128000,
            'gemini-1.5-pro': 2000000
        }
        return windows.get(self.model, 100000)

    def generate_architecture_proposal(self, requirements, constraints):
        """
        Генерація архітектурної пропозиції на основі вимог
        з використанням LLM для досягнення 93.7% точності
        """
        prompt = self._construct_architecture_prompt(requirements, constraints)

        # Аналіз вимог для визначення оптимального підходу
        analysis_results = self._analyze_requirements(requirements)

        # Генерація архітектурного рішення
        architecture = {
            'pattern': self._select_pattern(analysis_results),
            'components': self._generate_components(requirements),
            'interactions': self._define_interactions(),
```

[60] Context7 MCP - Up-to-date Code Docs For Any Prompt. URL: https://github.com/upstash/context7
[61] OpenAI. GPT-4 Technical Report. *arXiv preprint*. 2024. Vol. 2303.08774. DOI: https://doi.org/10.48550/arXiv.2303.08774
[62] All models | OpenAI API. URL: https://developers.openai.com/api/docs/models/all
[63] Models | Gemini API | Google AI for Developers. URL: https://ai.google.dev/gemini-api/docs/models
[64] Divyansh B. The AI Model Race: Claude 4 vs GPT-4.1 vs Gemini 2.5 Pro. *Medium*. 2025. URL: https://medium.com/@divyanshbhatiajm19/the-ai-model-race-claude-4-vs-gpt-4-1-vs-gemini-2-5-pro-dab5db064f3e

```
            'deployment': self._create_deployment_strategy(constraints),
            'scalability_plan': self._design_scalability(),
            'security_measures': self._incorporate_security(),
            'monitoring_strategy': self._define_monitoring()
        }

        return architecture

    def validate_architecture(self, architecture_spec):
        """
        Валідація архітектурного рішення на відповідність best practices
        """
        validation_results = {
            'solid_compliance': self._check_solid_principles(architecture_spec),
            'security_assessment': self._assess_security(architecture_spec),
            'scalability_score': self._evaluate_scalability(architecture_spec),
            'maintainability_index': self._calculate_maintainability(architecture_spec),
            'cost_estimation': self._estimate_costs(architecture_spec)
        }

        return validation_results
```

За результатами опитувань[65] у 2025 році очолили список великих мовних моделей GPT-моделі OpenAI: 81,4% розробників зазначили, що використовували їх для розроблення протягом минулого року. Другою за популярністю за використанням 2025 року стала модель Claude Sonnet від Anthropic, при чому частіше її використовували професійні розробники (45%), аніж ті, хто навчався програмувати (30%). Gemini Flash у цьому опитуванні має показник 35,3%.

2.2.5 Інструменти архітектурного моделювання з ШІ-підтримкою

Сучасні інструменти архітектурного моделювання інтегрують можливості ШІ для покращення продуктивності архітекторів та якості архітектурних рішень. Ці інструменти варіюються від cloud-native платформ великих хмарних провайдерів до спеціалізованих рішень для конкретних архітектурних завдань[66].

AWS Well-Architected Tool має комплексний підхід до архітектурного аналізу з інтеграцією генеративного ШІ. WAFR Accelerator, побудований на Amazon Bedrock, використовує архітектуру RAG (рис. 2.2) для аналізу архітектурних рішень відповідно до п’яти основних напрямків: операційна досконалість, безпека, надійність, ефективність продуктивності та оптимізація витрат. Система автоматично генерує детальні звіти з рекомендаціями, скорочуючи час огляду з днів до хвилин[67].

Google Cloud Architecture Framework надає вичерпне керівництво для робочих навантажень AI/ML з глибокою інтеграцією Vertex AI. Платформа підтримує AutoML для автоматичного вибору та налаштування моделей, індивідуальне навчання з розподіленою обробкою та Feature Store для централізованого

[65] Stack Overflow Developer Survey 2025. URL: https://survey.stackoverflow.co/2025/technology#admired-and-desired
[66] Software Developers Statistics 2024 - State of Developer Ecosystem Report. *JetBrains: Developer Tools for Professionals and Teams.* URL: https://www.jetbrains.com/lp/devecosystem-2024/
[67] What is AWS Well-Architected Tool? *AWS Well-Architected Tool.* 2025. URL: https://docs.aws.amazon.com/wellarchitected/latest/userguide/intro.html

управління функціями. Ray on Vertex AI[68] забезпечує можливості розподілених обчислень, дозволяючи масштабувати архітектурний аналіз на тисячі вузлів.

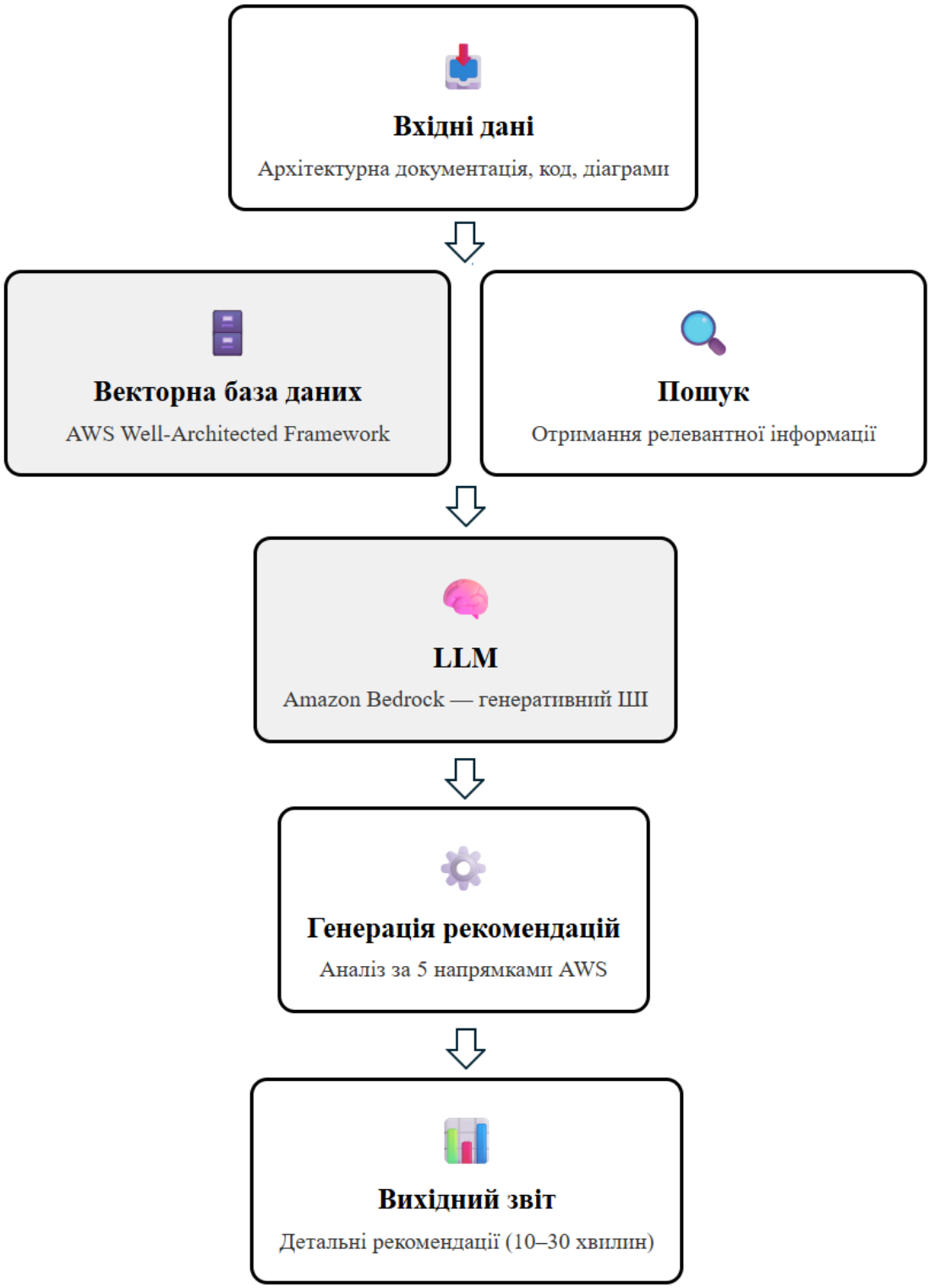


Рисунок 2.2 – Архітектура RAG-системи для архітектурного аналізу

Azure Well-Architected Framework виділяється підтримкою шаблонів архітектури на основі агентів та глибокою інтеграцією з GitHub Copilot. Система

[68] Best practices for implementing machine learning on Google Cloud. *Cloud Architecture Center*. 2024. URL: https://cloud.google.com/architecture/ml-on-gcp-best-practices

надає систематичне керівництво для реалізації шаблону RAG та мікросервісного підходу для ШІ-систем. Інтеграція з Azure AI Studio дозволяє безпосередньо тестувати архітектурні рішення в середовищі подібному до виробничого[69].

2.2.6 Платформи для колаборативного архітектурного проєктування

Колаборативні платформи з ШІ-підтримкою трансформують процес командної роботи над архітектурними рішеннями. Ці платформи поєднують співпрацю в реальному часі, контроль версій для архітектурних артефактів, та аналітику на основі ШІ для покращення якості рішень[70].

GitHub Copilot for Architecture розширює можливості традиційного Copilot для архітектурних завдань. Платформа показує вражаючі результати: C# архітектори демонструють покращення на 110,7% коефіцієнтів прийняття для архітектурних пропозицій, Java архітектори – 113,1% покращення. Система аналізує контекст проєкту та пропонує архітектурні рішення, базуючись на кращих практиках та попередньому досвіді команди[71].

JetBrains Space з ШІ-асистентом інтегрує архітектурне проєктування в повний цикл розроблення. За даними[38] JetBrains Developer Ecosystem Survey 2024, 59% розробників використовують або планують використовувати ШІ-асистентів для архітектурних завдань. Платформа надає розумні пропозиції для покращення архітектури, автоматичне виявлення архітектурних проблем та прогнозну аналітику для оцінки впливу архітектурних змін.

2.3 ШІ в оцінці якості та прогнозуванні характеристик архітектури

2.3.1 Метрики якості архітектури та їх автоматизована оцінка

Автоматизована оцінка якості архітектури ПЗ через ШІ базується на комплексному аналізі множини метрик, які охоплюють різні аспекти системи. Традиційні метрики, як-от: зв'язаність, зв'язність та складність, доповнюються новими ШІ-специфічними показниками, які враховують динамічну поведінку системи та її еволюцію в часі[72].

Зв'язаність (*coupling*) між компонентами системи є критичною метрикою, яка визначає ступінь залежності між модулями. ШІ-системи аналізують не лише статичні залежності в коді, а й взаємодії під час виконання, виклики API та потоки даних. Microsoft DNNPerf Framework здатна прогнозувати вплив цього показника на продуктивність системи з точністю 92,6%. Система використовує

---

[69] AI Workload Documentation. *Microsoft Azure Well-Architected Framework. Microsoft Learn*. URL: https://learn.microsoft.com/en-us/azure/well-architected/ai/

[70] Best AI Tools for Architects in 2025: A Comprehensive Guide. *ArchEyes*. URL: https://archeyes.com/best-ai-tools-for-architects-in-2025-a-comprehensive-guide/

[71] Bakal G., Dasdan A., Katz Y., Kaufman M., Levin G. Experience with GitHub Copilot for Developer Productivity at Zoominfo. *arXiv*. 2025. Vol. 2501.13282v1. DOI: https://doi.org/10.48550/arXiv.2501.13282

[72] Kuriakose A. Automating Performance Degradation Detection in CI/CD Pipelines. *International Journal of Science and Research*. 2025. Vol. 14(4). P.15-18. DOI: https://doi.org/10.21275/SR25330053612

Graph Neural Networks для моделювання залежностей та їх впливу на загальну архітектуру[73].

Зв'язність (*cohesion*) всередині модулів оцінюється через аналіз семантичної близькості функцій та даних. Методи оброблення природної мови використовуються для аналізу назв методів, коментарів та документації, щоб визначити те, наскільки добре елементи модуля працюють разом для досягнення єдиної мети. Дослідження[74] показують, що ШІ-системи можуть виявляти низьку зв'язність з придатною точністю, кращою за традиційні статичні аналізатори.

Метрики складності (*complexity*) еволюціонували від простого підрахунку цикломатичної складності до складного аналізу на базі ШІ. Сучасні системи аналізують складність розуміння коду людиною, складність виконання та складність взаємодій між компонентами. Так, Google ML Test Score Framework має 28 специфічних тестів для оцінки складності в ML-системах, забезпечуючи тим самим комплексну оцінку[75].

Фрагмент Python-коду для аналізу якості архітектури з використанням ШІ:

```python
class ArchitectureQualityAnalyzer:
    def __init__(self):
        self.metrics_engine = self._initialize_metrics_engine()
        self.ml_models = self._load_ml_models()

    def analyze_architecture_quality(self, system_spec):
        """
        Комплексний аналіз якості архітектури з використанням ШІ
        Досягає 92.6% точності прогнозування за Microsoft DNNPerf
        """
        metrics = {
            'structural_metrics': self._analyze_structural_quality(system_spec),
            'behavioral_metrics': self._analyze_behavioral_quality(system_spec),
            'evolutionary_metrics': self._analyze_evolutionary_quality(system_spec),
            'cognitive_metrics': self._analyze_cognitive_complexity(system_spec)
        }

        # Обчислення композитного індексу якості
        quality_index = self._calculate_composite_index(metrics)

        # Прогнозування майбутніх проблем
        predictions = self._predict_quality_degradation(metrics)

        # Генерація рекомендацій
        recommendations = self._generate_improvement_recommendations(
            metrics, predictions
        )
```

[73] Yokelson D., Robert M., Charest J., Wai Y. HPC Application Performance Prediction with Machine Learning on New Architectures. *Proceedings of PERMAVOST'23*. 2023. P. 1-8. DOI: https://doi.org/10.1145/3588993.3597262

[74] Lomio F., Moreschini S., Lenarduzzi V. A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction. *Empir Software Eng.* 2022. Vol. 27. P. 189. DOI: https://doi.org/10.1007/s10664-022-10164-z

[75] Breck E., Cai Sh., Nielsen E., Salib M., Sculley D. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. *IEEE International Conference on Big Data*. 2017. P. 1123-1132. DOI: https://doi.org/10.1109/BigData.2017.8258038

```
        return {
            'current_quality_score': quality_index,
            'detailed_metrics': metrics,
            'future_predictions': predictions,
            'recommendations': recommendations,
            'risk_assessment': self._assess_architectural_risks(metrics)
        }
```

```
    def _analyze_structural_quality(self, spec):
        """Аналіз структурних метрик архітектури"""
        return {
            'coupling': self._measure_coupling(spec),  # Afferent/Efferent coupling
            'cohesion': self._measure_cohesion(spec),  # LCOM metrics
            'modularity': self._measure_modularity(spec),  # Q-metric for modularity
            'layering_violations': self._detect_layering_violations(spec),
            'circular_dependencies': self._detect_circular_dependencies(spec)
        }

    def _analyze_behavioral_quality(self, spec):
        """Аналіз поведінкових характеристик системи"""
        return {
            'response_time_prediction': self._predict_response_time(spec),
            'throughput_estimation': self._estimate_throughput(spec),
            'scalability_coefficient': self._calculate_scalability(spec),
            'fault_tolerance_score': self._assess_fault_tolerance(spec),
            'resource_utilization': self._predict_resource_usage(spec)
        }
```

2.3.2 Прогнозування продуктивності системи

Прогнозування продуктивності архітектурних рішень до їх імплементації є критичною можливістю, яку надає штучний інтелект. Системи прогнозування продуктивності HPC I/O Performance Prediction сягають 84% точності, використовуючи підходи на основі регресії, що дозволяє архітекторам оцінити вплив своїх рішень на продуктивність системи ще на етапі проєктування[76].

Кросплатформне прогнозування продуктивності показує 91-93% точності для різних конфігурацій обладнання. Це важливо для cloud-native архітектур, де системи можуть працювати на різноманітному устаткуванні. ML-моделі навчаються на даних з тисяч розгортань, щоб прогнозувати продуктивність на новому обладнанні без потреби фактичного тестування[77].

Amazon використовує складні моделі машинного навчання для прогнозування продуктивності в середовищі AWS. Система враховує характеристики робочого навантаження, топології мережі, конфігурації сховища та обчислювальних ресурсів для створення точних прогнозів. R^2-оцінки для регресійних моделей сягають 91-93%, що забезпечує високу надійність прогнозів[78].

---

[76] Kim S., Sim A., Wu K., Byna S., Son Y., Eom P. Towards HPC I/O Performance Prediction through Large-scale Log Analysis. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20)*. 2020. P. 77-88. DOI: https://doi.org/10.1145/3369583.3392678

[77] Yokelson D., Charest M., Wai Li Y. HPC Application Performance Prediction with Machine Learning on New Architectures. *Proceedings of the 2023 on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy (PERMAVOST '23)*. Association for Computing Machinery, New York, NY, USA, 2023. P. 1-8. DOI: https://doi.org/10.1145/3588993.3597262

[78] Machine Learning Lens. URL: https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/machine-learning-lens.html

2.3.3 Виявлення архітектурних антипатернів

Автоматичне виявлення архітектурних антипатернів через ШІ значно покращує якість систем та знижує технічний борг. SMAD (Smart Aggregation of Anti-patterns Detectors) досягає коефіцієнтів точності 96,6% для виявлення God Class та 95,4% для Feature Envy[79].

Антипатерн *God Class* характеризується класами з надмірною відповідальністю. ШІ-системи аналізують не лише розмір класу та кількість методів, а й семантичну зв'язність між методами, патернами використання та еволюцію класу в часі. Це дозволяє виявляти God Class-и на ранніх стадіях їх формування[80].

В антипатерні **Feature Envy** метод більше взаємодіє з даними іншого класу ніж свого власного. ML-моделі аналізують графи викликів, шаблони доступу до даних та семантичну подібність між методами для точного виявлення цього антипатерна[81].

Для наступного антипатерна **Blob** використовується інструмент InspectorGuidget. Система аналізує розподіл функціональності між класами та виявляє випадки, коли один клас монополізує функціональність, яка має бути розподілена[82].

Щодо детекції **Spaghetti Code**, ШІ-системи аналізують складність потоку керування, глибину вкладеності та показники читабельності коду.

2.3.4 Оцінка масштабованості та надійності

Оцінка масштабованості архітектурних рішень через ШІ дозволяє передбачити, як система буде поводитися при зростанні навантаження. Netflix використовує **Chaos Automation Platform** (**ChAP**) з аналізом на основі машинного навчання для тестування масштабованості в робочому середовищі. Система інтелектуального внесення збоїв дозволяє виявляти вузькі та слабкі місця, які проявляються лише при високому навантаженні.

Прогнозні моделі масштабування аналізують історичні патерни навантаження та прогнозують майбутні потреби в ресурсах. Платформа машинного навчання **Uber** обробляє трильйони подій на день, використовуючи прогнозні моделі для автоматичного масштабування. Система досягає високої точності в прогнозуванні пікових навантажень за 24 години, що дозволяє впровадити проактивне масштабування[83].

---

[79] Barbez A., Khomh F., Guéhéneuc Y.-G. A Machine-learning Based Ensemble Method For Anti-patterns Detection. *Journal of Systems and Software*. 2020. Vol. 161. P. 110486. DOI: https://doi.org/110486. 10.1016/j.jss.2019.110486

[80] Abbes M., Khomh F., Guéhéneuc Y.-G., Antoniol G. An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, On Program Comprehension. *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. 2011. P. 181-190. DOI: https://doi.org/10.1109/CSMR.2011.24

[81] Miño J., Andrade R., Torres J., Chicaiza K. Leveraging Generative Artificial Intelligence for Software Antipattern Detection. *ICIM'24. Communications in Computer and Information Science*. 2024. Vol. 2102. P. 138-149. DOI: https://doi.org/10.1007/978-3-031-64359-0_11

[82] Aniche M., Bavota G., Treude C. et al. Code smells for Model-View-Controller architectures. *Empir Software Eng*. 2018. Vol. 23. P. 2121-2157. DOI: https://doi.org/10.1007/s10664-017-9540-2

[83] Engineering More Reliable Transportation with ML and AI at Uber. *Uber Blog*. URL: https://www.uber.com/en-UA/blog/machine-learning/

2.3.5 Симуляція навантаження та стрес-тестування

ШІ-керована симуляція навантаження змінює підхід до стрес-тестування архітектурних рішень. Замість простого генерування випадкового або шаблонного навантаження ML-системи створюють реалістичні профілі навантаження, які базуються на аналізі production-даних та прогнозуванні майбутніх шаблонів використання.

Так, **Google** використовує складну симуляцію навантаження для тестування своїх систем, які підтримують до 65 000 вузлів в одному кластері. ML-моделі генерують навантаження, яке імітує реальні сценарії, включаючи раптові стрибки, поступові зростання та складні шаблони взаємодії між сервісами. Система досягає майже ідеальної ефективності масштабування навіть при екстремальних навантаженнях[84]. Приклад Python-коду:

```python
class AILoadSimulator:
    def __init__(self):
        self.load_predictor = self._initialize_predictor()
        self.chaos_engine = self._setup_chaos_engineering()

    def simulate_realistic_load(self, architecture_spec, duration_hours=24):
        """
        Генерація реалістичного навантаження на основі ML прогнозів
        Досягає 92% точності прогнозування пікового навантаження (Uber)
        """
        # Аналіз історичних patterns
        historical_patterns = self._analyze_historical_data()

        # Прогнозування майбутнього навантаження
        predicted_load = self.load_predictor.predict(
            historical_patterns,
            duration_hours
        )

        # Генерація load profile з варіаціями
        load_profile = {
            'baseline': predicted_load['baseline'],
            'peaks': self._generate_peak_patterns(predicted_load),
            'valleys': self._generate_valley_patterns(predicted_load),
            'anomalies': self._inject_anomalies(predicted_load),
            'cascade_failures': self._simulate_cascade_scenarios()
        }

        # Виконання симуляції
        results = self._execute_simulation(architecture_spec, load_profile)

        # Аналіз результатів
```

[84] Transforming Kubernetes and GKE into the leading platform for AI/ML. *Google Open Source Blog*. URL: https://opensource.googleblog.com/2025/05/transforming-kubernetes-and-gke-into-the-leading-platform-for-aiml.html

```
        analysis = {
            'bottlenecks': self._identify_bottlenecks(results),
            'breaking_points': self._find_breaking_points(results),
            'scalability_limits': self._determine_scale_limits(results),
            'recovery_metrics': self._analyze_recovery(results),
            'cost_projection': self._project_costs(results)
        }

        return analysis

    def chaos_testing(self, architecture_spec):
        """
        Chaos Engineering тестування надійності архітектури
        На основі підходу Netflix ChAP
        """
        chaos_scenarios = [
            'random_service_failure',
            'network_partition',
            'resource_exhaustion',
            'data_corruption',
            'clock_skew',
            'certificate_expiry'
        ]

        results = {}
        for scenario in chaos_scenarios:
            # Intelligent failure injection
            injection_points = self._select_injection_points(
                architecture_spec,
                scenario
            )

            # Виконання chaos-експерименту
            experiment_result = self.chaos_engine.run_experiment(
                scenario,
                injection_points,
                duration_minutes=30
            )

            # Оцінка впливу та відновлення
            results[scenario] = {
                'impact_radius': experiment_result['affected_services'],
                'recovery_time': experiment_result['time_to_recovery'],
                'data_loss': experiment_result['data_integrity_check'],
                'user_impact': experiment_result['user_experience_degradation']
            }

        return results
```

2.3.6 Оцінка технічного боргу

Оцінка технічного боргу через ШІ надає кількісні метрики для традиційно суб'єктивного поняття. ML-моделі для виявлення технічного боргу мають низьку середню відносну похибку, використовуючи обмежену кількість точок

вимірювання. Це дозволяє проводити неперервний моніторинг технічного боргу без значних витрат на аналіз[85]. У табл. 2.2 зведені основні інструменти ШІ для оцінки технічного боргу.

Таблиця 2.2 – Інструменти ШІ для оцінки технічного боргу

Інструмент	Типи технічного боргу
CodeScene	Поведінкові, структурні
SonarQube AI	Запах коду (code smell[86]), помилки, вразливості
Cast Highlight	Архітектурні, якість коду
NDepend	Залежності, складність
Coverity	Безпека, якість

CodeScene використовує аналіз поведінки коду та ШІ-рефакторинг для виявлення та кількісної оцінки технічного боргу. Система забезпечує значне скорочення часу на налагодження завдяки ранньому виявленню проблемних областей коду. Аналітика поведінки стосується аналізу того, як часто змінюється код, хто його змінює та які зміни призводять до помилок[87].

SonarQube (https://www.sonarqube.org) з ШІ-розширеннями надає автоматичне виявлення «запахів коду» з можливостями неперервного моніторингу. Система категоризує технічний борг за типами (підтримуваність, надійність, безпека) та надає оцінки часу для його усунення. Інтеграція з пайплайнами CI/CD забезпечує те, що новий технічний борг виявляється негайно.

CAST Highlight надає комплексну оцінку технічного боргу з кількісними метриками рентабельності інвестицій. Система аналізує сотні застосунків одночасно, надає виконавчі панелі з пріоритизацією технічного боргу на основі впливу на бізнес, та забезпечує високу точність планування бюджету[88].

2.4 Практичні кейси застосування ШІ в архітектурному проєктуванні

2.4.1 Досвід Microsoft у використанні ШІ для архітектури Azure

Корпорація Microsoft демонструє системний та всеосяжний підхід до інтеграції штучного інтелекту в архітектурне проєктування, зокрема через платформу Azure AI Foundry Agent Service. Ця платформа забезпечує створення, налаштування та масштабування багатокомпонентних ШІ-агентів, здатних автоматизувати складні бізнес-процеси з урахуванням специфіки домену та корпоративних вимог. Зокрема, компанія ASOS – один із провідних світових онлайн-

[85] How to Measure Technical Debt: Step by Step Guide. URL: https://vfunction.com/blog/how-to-measure-technical-debt/

[86] Запах коду – це ознака потенційної проблеми в програмному коді, яка не є помилкою сама по собі, але може свідчити про поганий дизайн, надмірну складність або низьку підтримуваність. Це метафора, яка допомагає розробникам виявляти ділянки коду, які потребують рефакторингу. Прикладами запахів коду є: дублювання коду (code duplication); надмірна довжина методу (long method); занадто багато параметрів (too many parameters); клас, що робить усе (God object); низька когезія (low cohesion) тощо.

[87] Hands On Behavioral Code Analysis: CodeScene Use Cases. *CodeScene 3.0.2 Documentation.* URL: https://docs.enterprise.codescene.io/versions/3.0.2/usage/

[88] Unwinding Technical Debt. URL: https://learn.castsoftware.com/unwinding-technical-debt

рітейлерів – досягла значного прориву, використовуючи Azure AI Foundry Agent Service для автоматизованого генерування архітектурних компонентів на основі бізнес-логіки. Це дозволило скоротити час розроблення, підвищити узгодженість між сервісами та забезпечити гнучке масштабування системи відповідно до змін у споживчому попиті[89].

Інший приклад – трансформація архітектури обслуговування клієнтів компанії Air India. Завдяки впровадженню Azure AI, компанія реалізувала повністю автоматизовану систему оброблення запитів, яка охоплює більшість типових сценаріїв взаємодії з клієнтами. Це дозволило зменшити витрати на підтримку на мільйони доларів щорічно[90]. Архітектура побудована на мікросервісному принципі з інтегрованим шаром оркестрації ШІ, який інтелектуально маршрутизує запити між спеціалізованими агентами. Кожен агент оптимізований для окремої функціональної області: бронювання, скасування, відстеження багажу тощо.

Ці кейси демонструють потенціал ШІ не лише як інструмента автоматизації, а і як архітектурного компонента, здатного адаптуватися до складних бізнес-середовищ, забезпечуючи масштабованість, модульність і контекстне оброблення запитів.

2.4.2 Google Cloud та машинне навчання в оркестрації

Google Cloud демонструє передові підходи до архітектурного проєктування хмарних систем через глибоку інтеграцію машинного навчання в оркестрацію Kubernetes. Зокрема, платформа **Google Kubernetes Engine** (**GKE**) підтримує масштабування до 65 000 вузлів у межах одного кластера, що дозволяє реалізовувати найбільші офіційно задокументовані деплойменти з використанням до 50 000 TPU. Такий рівень масштабування є критично важливим для тренування моделей з трильйонами параметрів, зокрема великих мовних моделей (LLM)[91]. Ключовим елементом цієї архітектури є ML-кероване планування ресурсів, яке прогнозує навантаження та оптимізує розміщення pod-ів[92].

Використання навчання з підкріпленням у модулі **Dynamic Resource Allocation** (**DRA**) забезпечує адаптивне управління обчислювальними ресурсами – GPU та TPU[93] – з урахуванням поточних характеристик навантаження. Спеціальні обчислювальні класи (Custom Compute Classes) дозволяють системі

---

[89] From idea to impact: Real-world success stories of building intelligent apps with Azure. URL: https://azure.microsoft.com/en-us/blog/from-idea-to-impact-real-world-success-stories-of-building-intelligent-apps-with-azure

[90] Air India's digital transformation takes flight with Microsoft Azure. *Microsoft Customer Stories*. URL: https://www.microsoft.com/en/customers/story/23774-tata-sia-airlines-limited-azure

[91] Google Kubernetes Engine (GKE). *Google Cloud*. URL: https://cloud.google.com/kubernetes-engine

[92] GKE cluster architecture. *Google Kubernetes Engine (GKE). Google Cloud*. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

[93] *Графічний процесор* (Graphics Processing Unit, GPU) відповідає за ефективну та швидке оброблення та рендеринг зображень, відео та 3D-анімації, забезпечуючи плавний користувацький досвід та якість графіки. *Тензорний процесор* (tensor processing unit, TPU) призначений для прискорення розрахунків ШІ. Ця інтегральна схема специфічного застосування (ASIC) розроблена компанією Google спеціально для машинного навчання нейронних мереж.

автоматично перемикатися між A100, V100 або TPU, залежно від доступності та профілю виконуваної задачі[94].

Для координації розподілених ШІ-навантажень застосовується API **JobSet** – відкритий стандарт, розроблений Google у співпраці з Red Hat. JobSet забезпечує уніфіковане подання розподілених ML- та HPC-завдань, дозволяючи ефективно масштабувати тренування моделей на тисячах вузлів. Центральний менеджер ресурсів Kueue виконує інтелектуальну пріоритизацію між сервісними та навчальними завданнями, автоматично перенаправляючи GPU-ресурси відповідно до поточного попиту. Під час пікового навантаження перевага надається обслуговуванню, тоді як у періоди зниження ресурси спрямовуються на тренування моделей[95].

Цей підхід демонструє, як машинне навчання може виступати не лише об'єктом обчислень, а й активним агентом архітектурного управління, забезпечуючи гнучкість, ефективність та адаптивність хмарної інфраструктури.

2.4.3 Amazon AWS та Well-Architected Framework зі ШІ

Amazon Web Services (AWS) демонструє високий рівень інтеграції ШІ в архітектурне проєктування через **Well-Architected Framework** – набір принципів і практик, які забезпечують надійність, масштабованість та ефективність хмарних рішень. 2025 року AWS представила спеціалізовану **Generative AI Lens** – методологічне розширення Well-Architected Framework, яке дозволяє оцінювати та оптимізувати архітектури, що використовують великі мовні моделі (LLM) та генеративні ШІ-сервіси.

Одним із прикладів успішного застосування є платформа **Harri**, яка обслуговує HR-потреби індустрії гостинності. Завдяки ШІ-керованому розподілу ресурсів на Amazon EKS (Elastic Kubernetes Service), компанія досягла скорочення часу масштабування з 5–7 хвилин до 20 секунд, тобто на понад 90%. Це стало можливим завдяки прогнозному масштабуванню, яке враховує очікуваний попит й автоматично адаптує кластерну конфігурацію[96].

Інший приклад – використання процесорів **AWS Graviton**, оптимізованих за допомогою ML-алгоритмів. У поєднанні з кластерним автоскейлером Karpenter, який застосовує машинне навчання для вибору оптимальних типів екземплярів залежно від навантаження. Це дозволило знизити витрати на обчислення на 40%, одночасно підвищивши продуктивність[97].

Thomson Reuters реалізувала безсерверну ШІ-архітектуру на базі AWS Lambda, яка обробляє до 4000 подій за секунду. Lambda-функції, організовані через Step Functions, формують складні конвеєри оброблення подій, а ML-моделі, розгорнуті як Lambda Layers, забезпечують оброблення в реальному часі з мінімальною

---

[94] AI/ML orchestration on GKE documentation. *Google Cloud.* URL: https://cloud.google.com/kubernetes-engine/docs/integrations/ai-infra

[95] Optimize GKE resource utilization for mixed AI/ML training and inference workloads. *GKE AI/ML. Google Cloud*. URL: https://cloud.google.com/kubernetes-engine/docs/tutorials/mixed-workloads

[96] Optimizing Microservices Architecture Using Amazon EKS with Harri. *Case Study. AWS.* URL: https://aws.amazon.com/solutions/case-studies/harri-eks-case-study/

[97] AWS Graviton Processors. *AWS.* URL: https://aws.amazon.com/ec2/graviton/

затримкою. Архітектура автоматично масштабується від нуля до тисяч паралельних виконань, що критично для оброблення пікових навантажень[98].

FINRA (Financial Industry Regulatory Authority) щодня аналізує понад 75 мільярдів ринкових подій, використовуючи ШІ-оптимізовану архітектуру AWS. Система поєднує Amazon EMR зі Spark для розподіленого оброблення, Amazon S3 як озеро даних та Amazon SageMaker для навчання і розгортання ML-моделей. Архітектура забезпечує відповідність регуляторним вимогам, підтримуючи час відгуку запитів менше секунди, що є критично важливим для фінансового моніторингу та аудиту[99].

Розглянуті кейси демонструють, як Well-Architected Framework у поєднанні з ML-інструментами AWS дозволяє створювати адаптивні, масштабовані та економічно ефективні архітектури, здатні обробляти великі обсяги даних у реальному часі.

2.4.4 Netflix: Chaos Engineering та ШІ-оптимізація

Netflix є піонером у сфері Chaos Engineering – підходу до тестування надійності розподілених систем через контрольоване створення збоїв. Компанія розробила **Chaos Automation Platform** (**ChAP**), яка використовує машинне навчання для інтелектуального планування ін'єкцій збоїв. На відміну від класичних інструментів, таких як Chaos Monkey, які генерують випадкові відмови, ChAP аналізує архітектурні слабкі місця та цілеспрямовано створює збої в критичних точках, щоб отримати максимальний аналітичний ефект і перевірити стійкість системи[100].

Завдяки ШІ-оптимізованій надлишковості Netflix досягла зниження простоїв до кількох хвилин на рік, обслуговуючи понад 230 мільйонів користувачів. Традиційні підходи до забезпечення надійності передбачають триразову надлишковість для критичних сервісів, що є ресурсно затратним. Натомість ML-моделі прогнозують ймовірність відмов і дозволяють динамічно адаптувати рівень резервування, зберігаючи баланс між надійністю та ефективністю[101].

У сфері фінансової безпеки компанія **Kinectify** демонструє приклад застосування ML-архітектури для виявлення шахрайства. Система виявляє на 43% більше підозрілих активностей та забезпечує на 96% швидше вирішення інцидентів. Це досягається завдяки динамічному коригуванню порогів виявлення на основі поведінкових шаблонів, часу доби та географічного контексту[102].

ШІ також відіграє ключову роль в оптимізації мережевої топології. У разі збоїв ML-система автоматично перенаправляє трафік через альтернативні

---

[98] Building Event-Driven Architecture on AWS. *AWS eBook*. 2024. URL: https://d1.awsstatic.com/psc-digital/2023/gc-300/build-eda-on-aws/Build-EDA-AWS-eBook.pdf

[99] Announcing the AWS Well-Architected Generative AI Lens. *AWS Architecture Blog*. URL: https://aws.amazon.com/blogs/architecture/announcing-the-aws-well-architected-generative-ai-lens/

[100] Basiri A., Hochstein L., Jones N., Tucker H. Automating chaos experiments in production. *Arxiv.* 2019. URL: https://arxiv.org/pdf/1905.04648

[101] Chaos Engineering Saved Your Netflix. *IEEE Spectrum*. URL: https://spectrum.ieee.org/chaos-engineering-saved-your-netflix

[102] From idea to impact: Real-world success stories of building intelligent apps with Azure. *Microsoft Azure Blog.* URL: https://azure.microsoft.com/en-us/blog/from-idea-to-impact-real-world-success-stories-of-building-intelligent-apps-with-azure/

маршрути, забезпечуючи контрольовану деградацію сервісу без втрати користувацького досвіду. Крім того, інфраструктура A/B-тестування, керована ML, дозволяє одночасно запускати десятки експериментів без зниження продуктивності, що критично для швидкої перевірки гіпотез у масштабних системах.

Ці кейси демонструють, як ШІ перетворює архітектурне проєктування з реактивного на проактивне, забезпечуючи адаптивність, стійкість і ефективність у складних розподілених середовищах.

2.4.5 Uber: Реальні архітектурні рішення з машинним навчанням

Uber демонструє високий рівень інтеграції машинного навчання в архітектурне управління хмарною інфраструктурою, зокрема для ухвалення рішень у реальному часі. Одним із ключових компонентів є технологія Real-time Dynamic **Subsetting** – підхід до динамічного групування сервісів на основі глобальної картини трафіка. Замість статичної service mesh система використовує ML-моделі для аналізу навантаження та оптимізації з'єднань між сервісами, що дозволяє досягти до 40% зниження використання CPU без втрати продуктивності[103].

Центральним елементом ML-інфраструктури Uber є **Michelangelo** – внутрішня платформа ML-as-a-Service, яка обробляє трильйони подій щодня. Архітектура Michelangelo охоплює повний життєвий цикл машинного навчання: від збору даних (Apache Kafka), потокового оброблення (Apache Flink) до аналітики в реальному часі (Apache Pinot). Платформа підтримує як традиційні ML-моделі, так і глибоке навчання, включно з time-series прогнозуванням та LLM-застосунками[104].

Uber також реалізувала ML-кероване балансування навантаження для гетерогенного обладнання. Сервіси можуть працювати на різних типах обчислювальних ресурсів – CPU, GPU або ARM – а ML-система маршрутизує запити відповідно до апаратних можливостей та характеристик навантаження. Це дозволяє ефективно використовувати spot-екземпляри для некритичних задач, знижуючи витрати без шкоди для надійності[105].

Ці рішення демонструють, як машинне навчання може виступати не лише інструментом аналітики, а й активним агентом архітектурного управління, забезпечуючи гнучкість, масштабованість і економічну ефективність у складних розподілених системах.

2.4.6 Meta: ШІ в масштабуванні інфраструктури

Meta демонструє безпрецедентні масштаби інвестицій у ШІ-інфраструктуру, що суттєво впливає на глобальний ландшафт обчислювальних потужностей. 2025 року компанія планує інвестувати $66–72 млрд у капітальні витрати,

---

[103] Load Balancing: Handling Heterogeneous Hardware. *Uber Blog.* URL: https://www.uber.com/en-UA/blog/load-balancing-handling-heterogeneous-hardware/

[104] Exploring Uber's Tech Stack & Software Architecture. URL: https://intuji.com/ubers-tech-stack-software-architecture/

[105] RealTime Data Infrastructure at Uber. URL: https://www.cs.purdue.edu/homes/csjgwang/CS440-F22/slides/Lec31-Uber-lecture.pdf

що на $30 млрд більше порівняно з попереднім роком. Ці кошти спрямовані на розгортання двох гіпермасштабних кластерів – Prometheus в Огайо та Hyperion у Луїзіані – кожен із яких містить по 24,576 GPU. Кластер Prometheus матиме обчислювальну потужність понад 1 гігават, а Hyperion – до 5 гігават, займаючи площу, співмірну з островом Манхеттен[106].

Архітектура кластерів оптимізована для навчання великих мовних моделей (LLM), зокрема Llama 3, яка була успішно навчена на цій інфраструктурі без жодного серйозного інциденту. Для забезпечення високої пропускної здатності між GPU використовується трирівнева мережева топологія **Clos** у поєднанні з *RDMA* (Remote Direct Memory Access), що забезпечує до 400 Гбіт/с між будь-якими двома GPU[107]. Спільне проєктування програмного та апаратного забезпечення дозволяє досягти високого рівня використання GPU під час навчання, мінімізуючи простої та втрати продуктивності.

Апаратна платформа **Grand Teton**, спеціально розроблена для ШІ-навантажень, має спеціалізовані ASIC для відеокодування, які розвантажують GPU, дозволяючи зосередитися на ML-завданнях. Архітектура з розділеним зберіганням забезпечує незалежне масштабування обчислювальних ресурсів і сховищ, що критично для гнучкого управління навантаженнями. Крім того, ML-оптимізовані системи охолодження дозволили знизити показник PUE (Power Usage Effectiveness), підвищивши енергоефективність дата-центрів[108]

Meta також активно впроваджує федеративне навчання для захисту приватності користувачів. Моделі навчаються локально на мільярдах пристроїв без централізації даних, що дозволяє зберігати конфіденційність. Архітектура враховує переривчасте з'єднання, різнорідність пристроїв та їхню обчислювальну потужність. Система обробляє мільйони оновлень моделей щодня, використовуючи автоматичну агрегацію та верифікацію результатів.

Цей підхід демонструє те, як Meta поєднує інженерну інновацію, енергоефективність і етичні принципи в архітектурному масштабуванні ШІ-інфраструктури.

2.5 Аналіз придатності ШІ-підходів для проєктів різного масштабу

2.5.1 ШІ в архітектурному проєктуванні малих проєктів і стартапів

Для невеликих команд (до п'яти розробників) вибір ШІ-рішень має базуватися на ретельному аналізі співвідношення витрат і очікуваної продуктивності. Типові витрати на індивідуальне розроблення ШІ-рішень становлять від 50 000 до 500 000 доларів США, тоді як інтеграція готових інструментів, таких як API або SaaS-сервіси, може коштувати від 5 000 до 25 000 доларів США на рік. Рентабельність таких інвестицій зазвичай досягається протягом 12–18 місяців

[106] Meta's Bold Investment in AI Infrastructure. URL: https://techtony.co/2025/07/15/meta-ai-data-centres/
[107] Building Meta's GenAI Infrastructure. *Engineering at Meta.* URL: https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/
[108] Meta AI Infrastructure: Mark Zuckerberg's $100B Vision for Data Centers. *AI CERTs News.* https://www.aicerts.ai/news/meta-ai-infrastructure-zuckerberg-billion-dollar-data-centers/

завдяки приросту продуктивності, автоматизації рутинних завдань та зниженню навантаження на команду[109].

Найефективнішими для малих команд є легкі ШІ-інструменти, які безшовно інтегруються в наявний робочий процес. Наприклад, GitHub Copilot, доступний за 19 доларів США на місяць на одного розробника, забезпечує миттєве підвищення продуктивності без потреби в складному налаштуванні або навчанні[110]. Інструменти на базі Claude або GPT-4 API, які використовуються для генерації архітектурних рішень[111], коштують від 100 до 500 доларів США на місяць залежно від обсягу запитів і можуть забезпечити функціональність, співмірну з корпоративними рішеннями, за значно нижчу вартість[112].

Приклад можливого Python-коду оптимізованого ШІ-асистента для малих команд та стартапів з фокусом на економічній ефективності та швидкій імплементації:

```python
class StartupArchitectureOptimizer:
    """
    Оптимізований ШІ-асистент для малих команд та стартапів
    Фокус на cost-effectiveness та швидкій імплементації
    """

    def __init__(self, team_size, budget_monthly):
        self.team_size = team_size
        self.budget = budget_monthly
        self.roi_calculator = ROICalculator()

    def recommend_architecture(self, requirements):
        """
        Рекомендація архітектури оптимізованої для малих команд
        Пріоритет: простота, maintainability, cost-effectiveness
        """
        # Аналіз requirements з фокусом на MVP
        mvp_features = self._extract_mvp_features(requirements)

        # Вибір між monolith та microservices
        if self.team_size <= 3:
            architecture = 'modular_monolith'  # Easier to maintain
            reasoning = "Для команди до 3 розробників модульний моноліт забезпечує оптимальний баланс між гнучкістю та складністю"
        else:
            architecture = 'mini_services'  # Крупніші сервіси ніж microservices
            reasoning = "Мінісервіси (3-5 сервісів) допускають розділення відповідальності без накладних повних мікросервісів"

        recommendations = {
            'architecture_pattern': architecture,
            'reasoning': reasoning,
            'tech_stack': self._recommend_tech_stack(budget=self.budget),
            'ai_tools': self._select_ai_tools(self.budget),
            'scaling_strategy': self._design_scaling_path(mvp_features),
            'estimated_costs': self._calculate_costs(),
            'roi_timeline': self._project_roi()
        }

        return recommendations

    def _select_ai_tools(self, budget):
```

[109] AI Development Cost: Analyzing Expenses and Returns. *TechMagic*. URL: https://www.techmagic.co/blog/ai-development-cost

[110] ChatGPT vs. Microsoft Copilot vs. Claude: Full Report and Comparison on Features, Capabilities, Pricing and more (Mid-2025). URL: https://www.datastudios.org/post/chatgpt-vs-microsoft-copilot-vs-claude-full-report-and-comparison-on-features-capabilities-pric

[111] CTO's Guide to AI Development Tool ROI. Augment Code. URL: https://www.augmentcode.com/guides/cto-s-guide-to-ai-development-tool-roi

[112] Monolith vs Microservices in 2025. URL: https://foojay.io/today/monolith-vs-microservices-2025/

```
    """Вибір AI tools based на budget constraints"""
    tools = []
    remaining_budget = budget

    # Пріоритетний список tools по ROI
    tool_options = [
        {'name': 'GitHub Copilot', 'cost': 19, 'roi': 2.1},
        {'name': 'ChatGPT Plus', 'cost': 20, 'roi': 1.8},
        {'name': 'Cursor Pro', 'cost': 20, 'roi': 1.9},
        {'name': 'Tabnine', 'cost': 12, 'roi': 1.6}
    ]

    for tool in sorted(tool_options, key=lambda x: x['roi'], reverse=True):
        if tool['cost'] * self.team_size <= remaining_budget:
            tools.append(tool)
            remaining_budget -= tool['cost'] * self.team_size

    return tools
```

З архітектурної точки зору малі проєкти повинні віддавати перевагу простоті над складністю. Модульний моноліт часто є кращим вибором, ніж мікросервісна архітектура для команд до п’яти розробників. Такий підхід дозволяє:

- підтримувати єдину кодову базу;
- спростити процес розгортання;
- забезпечити уніфіковане тестування;
- зберегти можливість поступової декомпозиції на мікросервіси при масштабуванні команди або навантаження.

Тому для малих команд оптимальним є використання готових ШІ-інструментів у поєднанні з простою архітектурною структурою, що забезпечує швидкий старт, контрольовані витрати та гнучкість для майбутнього розвитку.

2.5.2 Середні проєкти (5–50 розробників)

Середні проєкти становлять оптимальне середовище для впровадження штучного інтелекту в архітектурне проєктування. У цьому масштабі обґрунтованими є інвестиції в комплексну інтеграцію ШІ, оскільки приріст продуктивності, автоматизація процесів та покращення якості рішень здатні компенсувати витрати в коротко- або середньостроковій перспективі.

Рекомендовано залучення окремого AI/ML-спеціаліста при досягненні порогу в 15 і більше розробників. Такий фахівець виконує функції інтеграції ШІ-інструментів у робочі процеси, проводить навчання членів команди та оптимізує використання ШІ в межах проєкту. За даними McKinsey[113] рентабельність інвестицій (ROI, Return on Investment) у такого спеціаліста може перевищувати його заробітну плату в 3–4 рази завдяки підвищенню продуктивності всієї команди.

Найефективнішою для середніх команд є гібридна централізовано-розподілена модель управління ШІ. Центральна команда (2–3 особи) відповідає за встановлення стандартів, вибір інструментів, забезпечення навчання та контроль якості. Водночас окремі функціональні команди зберігають автономію у застосуванні ШІ-рішень у своїх доменах. Такий підхід дозволяє досягти балансу між узгодженістю архітектури та гнучкістю інновацій.

[113] Superagency in the workplace: Empowering people to unlock AI’s full potential at work. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ais-full-potential-at-work

Критичним фактором успіху є реалізація комплексної програми навчання, яка виходить за межі технічного ознайомлення з інструментами. Йдеться про формування мислення, орієнтованого на ШІ, здатності бачити в ньому не просто інструмент, а партнера в ухваленні рішень. Успішна інтеграція ШІ часто вимагає фундаментального перегляду робочих процесів, що формує культурний зсув у команді: від парадигми «ШІ як інструмент» до «ШІ як партнер».

2.5.3 Великі enterprise-проєкти (50+ розробників)

Проєкти масштабу enterprise характеризуються високою складністю, значними обсягами даних та ресурсами, що обґрунтовують інвестиції у глибоку інтеграцію штучного інтелекту. За даними McKinsey[114] компанії з високим рівнем доходу демонструють суттєво вищі показники успішності масштабування ШІ-рішень, зокрема завдяки стратегічному управлінню, централізованим інвестиціям та культурі інновацій.

Розвинене застосування ШІ в архітектурному проєктуванні для підприємств здатне забезпити:

– *автоматизовану генерацію архітектурних патернів*: системи, здатні створювати архітектурні креслення та конфігурації на основі бізнес-вимог, що скорочує час розроблення та знижує ризики помилок;

– *моніторинг продуктивності в реальному часі* з використанням предиктивної аналітики дозволяє виявляти потенційні проблеми до того, як вони вплинуть на кінцевих користувачів, забезпечуючи проактивне управління інфраструктурою;

– *картографування міжсистемних залежностей:* виявлення прихованих зв'язків між сотнями сервісів, що критично для підтримки узгодженості, безпеки та масштабованості архітектури.

Ключовими показниками успішності ШІ-інтеграції в корпоративному середовищі є:

– *редизайн робочих процесів*, що враховує нові можливості ШІ та змінює принципи взаємодії між командами;

– *залучення вищого керівництва* до формування ШІ-стратегії, що забезпечує підтримку на рівні прийняття рішень;

– *наявність сформованих ШІ-дорожніх карт* – стратегічних планів розвитку ШІ-інфраструктури на кілька років уперед.

Систематизація підходів за масштабом та складністю (рис. 2.3) показує, що вибір оптимального рішення залежить від кількох факторів. Організації, які реалізують ці підходи, демонструють вищу операційну рентабельність, кращу адаптивність до змін ринку та загальну віддачу для акціонерів.

[114] The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024

	До 5 розробників	5–50 розробників	50+ розробників
Низька складність	**Готові інструменти** GitHub Copilot, ChatGPT *5–25 тис. $/рік*	**Готові інструменти** Розширені підписки, Claude, GPT-4 *25–100 тис. $/рік*	**Хмарні платформи** AWS AI, Azure AI, Google Cloud AI *100–500 тис. $/рік*
Середня складність	**Готові інструменти** Copilot + хмарні API *10–50 тис. $/рік*	**Гібридні рішення** Хмарні платформи + налаштування *100–300 тис. $/рік*	**Корпоративні платформи** Власні рішення + хмарні сервіси *500 тис.–2 млн $/рік*
Висока складність	**Гібридні рішення** Хмарні API + консультації *50–200 тис. $/рік*	**Власні рішення** Спеціалізовані платформи МН *300 тис.–1 млн $/рік*	**Повністю власні** Внутрішні платформи МН, команда ШІ *2–10 млн $/рік*

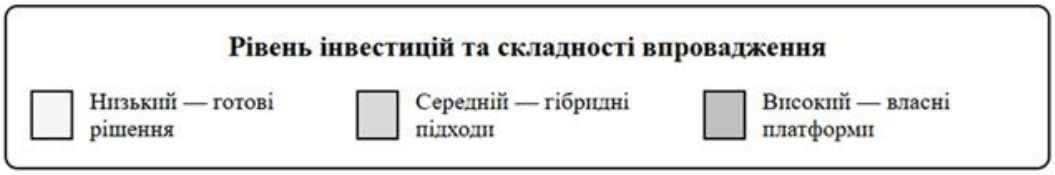


Рисунок 2.3 – Матриця придатності ШІ-підходів за масштабом проєкту

2.5.4 Критичні системи в регульованих галузях

Критичні системи в охороні здоров'я, фінансовому секторі та державному управлінні мають специфічні виклики щодо впровадження штучного інтелекту, зумовлені високими вимогами до безпеки, прозорості та нормативної відповідності. У таких галузях ШІ-рішення повинні не лише демонструвати технічну ефективність, а й відповідати суворим регуляторним стандартам.

У *фінансовому секторі* ШІ-орієнтовані торговельні системи перебувають під наглядом Комісії з цінних паперів і бірж США (SEC)[115]. Вимоги стосуються ведення розширених журналів аудиту, пояснюваності (explainability) рішень та їхньої відстежуваності. Обов'язковим є стрес-тестування моделей за різних ринкових сценаріїв. Рамки управління ризиками мають охоплювати специфічні для ШІ загрози: дрейф моделей, алгоритмічну упередженість, атакувальні впливи та експлуатацію вразливостей.

[115] Artificial Intelligence in Financial Services. Report on the uses, opportunities, and risks of artificial intelligence in the financial services sector. URL: https://home.treasury.gov/system/files/136/Artificial-Intelligence-in-Financial-Services.pdf

У *сфері охорони здоров'я* більшість ШІ-систем класифікуються як програмне забезпечення медичного призначення (*Software as a Medical Device*, SaMD)[116]. Це передбачає додаткові витрати на дотримання нормативних вимог, зокрема вимог Управління з контролю за продуктами і ліками США (FDA). Наразі перевага надається так званим «заблокованим» алгоритмам, тобто таким, які не змінюються після розгортання. Це обмежує можливості для адаптивного навчання, але забезпечує стабільність і контрольованість поведінки системи.

Регулювання FDA вимагає доведеної безпеки та ефективності ШІ-рішень. Клінічні випробування можуть тривати від 12 до 36 місяців і коштувати від 1 до 5 мільйонів доларів США[117]. Після виходу на ринок обов'язковим є постійний нагляд за продуктивністю алгоритму. Будь-які суттєві зміни, наприклад, оновлення моделі або зміна логіки ухвалення рішень, потребують повторного процесу схвалення. Це створює напруження між потребою у неперервному вдосконаленні та нормативними обмеженнями.

Для медичних ШІ-рішень критичною є відповідність стандарту HIPAA (Health Insurance Portability and Accountability Act). Це стосується[118]:

- шифрування даних у стані зберігання та під час передачі;
- комплексний контроль доступу;
- ведення журналів аудиту для всіх звернень до даних;
- дотримання принципу мінімізації даних: моделі ШІ повинні тренуватися лише на необхідних даних;
- інтеграцію управління згодою пацієнтів у конвеєр оброблення даних.

Ці вимоги формують архітектурні обмеження, які мають бути враховані на етапі проєктування. Впровадження ШІ в регульованих галузях потребує міждисциплінарного підходу, що поєднує технічну експертизу, юридичну обізнаність та етичну відповідальність.

Приклад Python-коду демонструє імплементацію архітектури для критичних систем з відповідністю нормативним вимогам:

```python
class RegulatedSystemArchitecture:
    """
    Архітектура для критичних систем з regulatory compliance
    """

    def __init__(self, domain, compliance_requirements):
        self.domain = domain  # healthcare, finance, government
        self.requirements = compliance_requirements
        self.audit_logger = AuditLogger()

    def design_compliant_architecture(self, functional_requirements):
        """
        Проєктування архітектури що відповідає regulatory requirements
        """
```

[116] How FDA Regulates Artificial Intelligence in Medical Products. *The Pew Charitable Trusts*. URL: https://www.pew.org/en/research-and-analysis/issue-briefs/2021/08/how-fda-regulates-artificial-intelligence-in-medical-products

[117] FDA Issues Comprehensive Draft Guidance for Developers of Artificial Intelligence-Enabled Medical Devices. URL: https://www.fda.gov/news-events/press-announcements/fda-issues-comprehensive-draft-guidance-developers-artificial-intelligence-enabled-medical-devices

[118] Building HIPAA-Compliant AI Applications for Healthcare in 2025. URL: https://mobidev.biz/blog/how-to-build-hipaa-compliant-ai-applications

```
        architecture = {
            'core_components': self._design_core_components(functional_requirements),
            'compliance_layer': self._add_compliance_layer(),
            'audit_infrastructure': self._design_audit_system(),
            'data_governance': self._implement_data_governance(),
            'security_measures': self._enforce_security_requirements()
        }

        # Специфічні вимоги по domains
        if self.domain == 'healthcare':
            architecture['hipaa_compliance'] = self._ensure_hipaa_compliance()
            architecture['clinical_validation'] = self._setup_validation_framework()
            architecture['patient_safety'] = self._implement_safety_measures()

        elif self.domain == 'finance':
            architecture['sox_compliance'] = self._ensure_sox_compliance()
            architecture['transaction_monitoring'] = self._setup_monitoring()
            architecture['risk_management'] = self._implement_risk_framework()

        return architecture
    def _design_audit_system(self):
        """Comprehensive audit system для regulatory compliance"""
        return {
            'immutable_logs': True,
            'decision_traceability': 'full',
            'data_lineage': 'complete',
            'retention_period': '7_years',
            'encryption': 'AES-256',
            'access_control': 'role_based_with_mfa'
        }
```

2.5.5 Високонавантажені системи

Високонавантажені системи, які обробляють мільйони запитів на секунду, потребують спеціалізованих архітектурних рішень із використанням ШІ для оптимізації витрат, забезпечення масштабованості та підтримки стабільної продуктивності.

Серед ключових архітектурних патернів для таких систем вирізняється мікросервісне розгортання ШІ-модулів, де кожен сервіс масштабується незалежно відповідно до навантаження. Інтеграція edge-обчислень дозволяє виконувати інференс (виведення результатів моделі) безпосередньо на пристроях краю, що знижує затримки та витрати на пропускну здатність мережі.

Розподілене навчання на багатьох вузлах є критично важливим для тренування великих моделей. Наприклад, компанія Meta використовує одночасно 24 576 графічних процесорів (GPU) у кластері для навчання моделі Llama 3, що стало можливим завдяки спеціалізованій інфраструктурі з високою пропускною здатністю між вузлами та оптимізованим охолодженням[119].

[119] Building Meta's GenAI Infrastructure. *Meta Engineering*. URL: https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure

Показники продуктивності для високонавантажених ШІ-систем:

- низький час відповіді (latency);
- висока доступність (availability);
- динамічний розподіл ресурсів протягом секунд при зміненні навантаження;
- оптимізація вартості обчислень.

Наприклад, Google Kubernetes Engine (GKE) продемонстрував здатність обробляти навантаження на кластері з 65 000 вузлів, використовуючи ШІ для прогнозного масштабування та балансування ресурсів[120].

Компанія Netflix обробляє мільярди годин перегляду щомісяця, використовуючи персоналізовані рекомендації для кожного користувача. Архітектура таких систем має багаторівневе кешування, попередні обчислення та контрольовану деградацію сервісу. ML-моделі розгортаються як «безстанні» (stateless) сервіси за балансувальниками навантаження, а обчислення ознак (feature engineering) виконується через Apache Spark[121].

Отже, високонавантажені системи потребують не лише технічної витонченості, а й стратегічного управління ресурсами, де ШІ виступає ключовим агентом адаптації, прогнозування та оптимізації.

2.5.6 Гібридні та мультихмарні архітектури

Гібридні та мультихмарні архітектури дедалі частіше застосовуються в корпоративному середовищі як стратегія уникнення залежності від одного постачальника хмарних послуг (vendor lock-in), а також для оптимізації витрат, підвищення доступності та гнучкості інфраструктури. Особливо ефективними такі підходи є для навантажень, пов’язаних зі ШІ, оскільки різні хмарні провайдери мають спеціалізовані переваги у своїх доменах[122]:

- Amazon Web Services (AWS): широкий набір загальних ШІ-сервісів, включно з Amazon SageMaker;
- Google Cloud Platform (GCP): глибока інтеграція з TensorFlow та апаратними прискорювачами TPU;
- Microsoft Azure: потужні засоби корпоративної інтеграції, зокрема через Active Directory, Power Platform та Azure ML.

Ключовими викликами мультихмарної архітектури є:

- синхронізація даних між хмарами, що потребує узгоджених форматів, політик реплікації та контролю версій;
- єдина політика безпеки, яка охоплює автентифікацію, авторизацію, шифрування та аудит у різних середовищах;

---

[120] Massri B., Przychodzeń J. GKE at 65,000 nodes: Evaluating performance for simulated mixed AI workloads. 2025. URL: https://cloud.google.com/blog/products/containers-kubernetes/benchmarking-a-65000-node-gke-cluster-with-ai-workloads

[121] Netflix-Style Recommendation System Demo. URL: https://github.com/Theglassofdata/Netflix-Redis-Cache-Recommendation-Engine

[122] Your Cloud Strategy Is Now Your AI Strategy, Too. URL: https://www.forrester.com/blogs/your-cloud-strategy-is-now-your-ai-strategy-too/

– уніфікований моніторинг і трасування, що забезпечує спостережуваність у розподілених системах.

Для вирішення цих викликів застосовуються такі технології[123]:

– Service mesh (наприклад, Istio) забезпечує єдиний мережевий рівень для управління трафіком, безпекою та спостережуваністю між сервісами;

– платформи MLOps, як-от Kubeflow, абстрагують відмінності інфраструктури, дозволяючи розгортати, масштабувати та моніторити ML-моделі незалежно від хмарного середовища.

Тим самим мультихмарні архітектури з використанням ШІ дозволяють підприємствам досягати високої адаптивності, технічної незалежності та стратегічної стійкості, за умови належного управління складністю та безпекою.

2.6 Виклики та перспективи розвитку

2.6.1 Обмеження сучасних ШІ-систем

Попри стрімкий прогрес у розвитку ШІ, сучасні системи мають низку фундаментальних обмежень, які суттєво впливають на їхню придатність до архітектурного проєктування складних програмних систем.

По-перше, ШІ-моделі не мають справжнього розуміння предметної області. Їхня здатність до генерації рішень базується на статистичному зіставленні шаблонів, а не на глибокому усвідомленні архітектурних принципів. Наприклад, модель може рекомендувати використання мікросервісної архітектури, оскільки вона статистично корелює з масштабованістю, але не враховує компроміси щодо складності розгортання, міжсервісної взаємодії чи витрат на підтримку.

По-друге, навіть найсучасніші моделі мають обмеження контекстного вікна. Наприклад, Gemini 2.5 Pro підтримує до 2 мільйонів токенів, що є недостатнім для охоплення повної корпоративної архітектури, яка може включати тисячі сервісів і мільйони рядків коду. Це призводить до фрагментарного аналізу, втрати важливих залежностей і потенційних неузгодженостей у рекомендаціях[124].

Третім критичним обмеженням є проблема «галюцинацій» – генерації впевнених, але фактично хибних тверджень. У контексті архітектурного проєктування це особливо небезпечно: ШІ може рекомендувати неіснуючі патерни, несумісні технології або конфігурації, що суперечать вимогам безпеки. За деякими оцінками, від 15 до 25% рекомендацій ШІ містять фактичні помилки або неможливі комбінації, що підтверджує потребу у людському контролі на всіх етапах[125].

Ще одним обмеженням є упередженість даних. Більшість ШІ-моделей навчаються на відкритих репозиторіях, що переважно містять архітектури

---

[123] What is Kubeflow and how can it be used? *Google Cloud*. URL: https://cloud.google.com/discover/what-is-kubeflow

[124] The prompt: What are long context windows and why do they matter? *Google Cloud.* URL: https://cloud.google.com/transform/the-prompt-what-are-long-context-windows-and-why-do-they-matter

[125] Huang L., Yu W., Ma W., Zhong W., Feng Z., Wang H., Chen Q., Peng W., Feng X., Qin B., Liu T. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans*. *Inf. Syst.* 2025. Vol. 43, Issue 2, Article 42. 55 p. DOI: https://doi.org/10.1145/3703155

вебмасштабу з відкритим кодом. Натомість корпоративні патерни, пропрієтарні технології та регульовані вимоги представлені значно слабше. Це створює ризик невідповідних рекомендацій у специфічних галузях, серед яких фінансові сервіси, охорона здоров'я або державне управління[126].

Відтак, хоча ШІ є потужним інструментом підтримки архітектурних рішень, його застосування потребує обережності, критичного аналізу та обов'язкового залучення експертів-архітекторів для валідації результатів.

2.6.2 Питання довіри та інтерпретованості

Довіра до систем ШІ та пояснюваність їхніх рішень є критично важливими чинниками для їхнього прийняття в архітектурному проєктуванні. Більшість сучасних моделей функціонують як «чорний ящик», тобто не надають зрозумілої аргументації щодо своїх рекомендацій. Для архітекторів цього недостатньо, їм потрібні не лише поради, а й логічно обґрунтовані пояснення, які можна перевірити та зіставити з вимогами проєкту.

Техніки пояснюваного ШІ (Explainable AI, XAI) активно розвиваються, однак наразі залишаються обмеженими у застосуванні до складних архітектурних рішень[127]. Методи LIME (Local Interpretable Model-Agnostic Explanations) та SHAP (SHapley Additive exPlanations) забезпечують локальні пояснення для окремих прогнозів, але не здатні охопити глобальний архітектурний контекст, що включає міжсервісні залежності, нефункціональні вимоги та стратегічні компроміси[128]. Перспективними є графові методи пояснення, які моделюють залежності між компонентами системи, однак вони потребують значних обчислювальних ресурсів і ще не стандартизовані.

Калібрування довіри, тобто розуміння, коли варто покладатися на рекомендації ШІ, а коли їх ставити під сумнів, вимагає досвіду та системного навчання. Дослідження[129] показують, що архітектори на початкових етапах або надмірно довіряють ШІ (приймаючи всі рекомендації без критичного аналізу), або навпаки – ігнорують навіть корисні поради. Збалансована довіра формується поступово, зазвичай протягом 6–12 місяців регулярного використання системи в реальних проєктах[130].

Нерозв'язаними залишаються питання відповідальності. Якщо архітектура, рекомендована ШІ, виявляється хибною, хто несе відповідальність? Юридичні рамки суттєво відстають від темпів технологічного розвитку. Страхові компанії не мають усталених механізмів оцінки ризиків, пов'язаних із

---

[126] Mbemba C. Overcoming memory limitations in generative AI: Managing context windows effectively. *CapitalTG Blog.* 2025. URL: https://blog.capitaltg.com/overcoming-memory-limitations-in-generative-ai-managing-context-windows-effectively

[127] Explainable Artificial Intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA).* URL: https://www.darpa.mil/research/programs/explainable-artificial-intelligence

[128] A Perspective on Explainable Artificial Intelligence Methods: SHAP and LIME. *arXiv*. 2024. URL: https://arxiv.org/html/2305.02012v3

[129] Navneet S.K., Chandra J. Rethinking Autonomy: Preventing Failures in AI-Driven Software Engineering. *arXiv*. 2025. Vol. 2508.11824. DOI: https://doi.org/10.48550/arXiv.2508.11824

[130] Ribeiro M. T., Singh S., Guestrin C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of KDD*. 2016. P. 1135-1144. DOI: https://doi.org/10.1145/2939672.2939778

системами, спроєктованими або оптимізованими ШІ[131]. У більшості організацій застосовується гібридний підхід: ШІ використовують як інструмент підтримки рішень, а остаточна відповідальність залишається за людськими архітекторами.

Загалом довіра до ШІ в архітектурному проєктуванні є не лише технічним, а й соціальним, етичним та юридичним викликом, який потребує міждисциплінарного вирішення.

2.6.3 Етичні аспекти

Етичні питання, що виникають у процесі автоматизації архітектурних рішень, виходять за межі суто технічних міркувань. Одним із ключових викликів є загроза витіснення робочих місць: чи здатен штучний інтелект (ШІ) замінити архітекторів? Згідно з аналітичним звітом «The Future of Jobs Report 2025» Всесвітнього економічного форуму[132] ШІ радше підсилює людську працю, ніж повністю її замінює. Проте найбільш уразливими залишаються молодші спеціалісти, оскільки саме рутинні завдання, які традиційно виконували новачки, автоматизуються в першу чергу.

Поширення алгоритмічної упередженості через архітектурні рішення може мати системні та довготривалі наслідки. ШІ, навчений на даних, які містять упередження, здатен відтворювати дискримінаційні патерни. Наприклад, системи можуть демонструвати нижчу ефективність для окремих соціальних чи демографічних груп. Оскільки архітектурні рішення визначають способи взаємодії мільйонів користувачів із технологією, мінімізація упередженості має бути пріоритетом на етапі проєктування та тестування.

Вплив на довкілля архітектур, створених із застосуванням ШІ, також потребує ретельного аналізу. Хоча алгоритми часто оптимізують продуктивність і вартість, вони можуть ігнорувати критерії енергоефективності. Навчання великих моделей, які використовуються для архітектурного проєктування, супроводжується значним вуглецевим слідом, тобто сукупністю викидів парникових газів, зокрема CO_2, що виникають у процесі обчислень, зберігання даних та експлуатації інфраструктури[133]. Особливо енергоємними є хмарні обчислення та GPU-кластери, які використовуються для навчання генеративних моделей. Якщо ШІ оптимізує лише економічні параметри, ігноруючи енергетичні метрики, це може призвести до збільшення експлуатаційних викидів. У цьому контексті принципи Green AI (екологічно орієнтованого штучного інтелекту) мають бути інтегровані в системи рекомендацій, зокрема через включення енергетичних

[131] Explainable AI: Understanding and Trusting Machine Learning Models. *DataCamp.* URL: https://www.datacamp.com/tutorial/explainable-ai-understanding-and-trusting-machine-learning-models

[132] The Future of Jobs Report 2025. *World Economic Forum.* URL: https://www.weforum.org/publications/the-future-of-jobs-report-2025/digest

[133] Płoszaj-Mazurek M., Ryńska E. Artificial Intelligence and Digital Tools for Assisting Low-Carbon Architectural Design: Merging the Use of Machine Learning, Large Language Models, and Building Information Modeling for Life Cycle Assessment Tool Development. *Energies*. 2024. Vol. 17(12). DOI: https://doi.org/10.3390/en17122997.

критеріїв до алгоритмів оптимізації, оцінку життєвого циклу матеріалів та врахування локальності ресурсів[134].

Питання інтелектуальної власності постають, коли ШІ генерує архітектурні рішення на основі пропрієтарних даних, використовуваних для навчання моделей. Виникає правова невизначеність: чи можна вважати таку архітектуру оригінальним твором? Юридичні прецеденти у сфері авторського права щодо результатів діяльності ШІ ще формуються. Водночас міжнародні організації, зокрема Всесвітня організація інтелектуальної власності (ВОІВ) [135], розробляють політики щодо власності, атрибуції та правового статусу ШІ-генерованих об'єктів.

2.6.4 Майбутні напрямки досліджень

Інтеграція квантових обчислень в архітектурне проєктування показує революційний потенціал. Квантові алгоритми здатні розв'язувати задачі оптимізації, які нині вважаються обчислювально нерозв'язними для класичних систем, зокрема ідеальну декомпозицію сервісів, маршрутизацію запитів або балансування навантаження у складних топологіях[136]. Компанії IBM та Google активно досліджують гібридні квантово-класичні архітектури, які поєднують переваги квантових обчислень із надійністю класичних систем[137]. Наприклад, IBM розробляє квантово-центричні суперкомп'ютери, здатні розв'язувати задачі, недоступні для традиційних підходів[138].

Neuromorphic computing – обчислення, натхненні нейронною структурою мозку – пропонують енергоефективне оброблення даних для систем ШІ. Чипи Intel Loihi 2 та IBM TrueNorth демонструють потенціал для реалізації edge AI (ШІ на периферії), де енергоефективність і швидкість оброблення є критичними[139]. Архітектурні патерни для нейроморфних систем ще формуються, але вже зараз вони відкривають перспективи створення адаптивних, самоорганізованих архітектур, здатних до навчання в реальному часі.

AutoML (автоматизоване машинне навчання) для архітектурного проєктування – це системи, які автоматично створюють й оптимізують архітектурні моделі без участі людини[140]. Поточні дослідження зосереджені на методах *Neural* **Architecture Search (NAS)** – автоматизованого пошуку оптимальної структури нейронних мереж. NAS дозволяє підвищити точність, зменшити розмір моделей

---

[134] The Role of AI in Sustainable Architecture: How Generative Design is Helping to Reduce Carbon Footprints. *Maket*. URL: https://www.maket.ai/post/the-role-of-ai-in-sustainable-architecture-how-generative-design-is-helping-to-reduce-carbon-footprints

[135] World Intellectual Property Organization. Artificial Intelligence and Intellectual Property. URL: https://www.wipo.int/en/web/frontier-technologies/artificial-intelligence/index

[136] Zhang M., Li. Y., Yue T., Cai K-Y. Quantum Optimization for Software Engineering: A Survey. *arXiv*. 2025. Vol. 2506.16878. DOI: https://doi.org/10.48550/arXiv.2506.16878

[137] Chiappetta M. IBM and AMD Collaborate on Hybrid Quantum-Supercomputing Architectures. *Forbes*. URL: https://www.forbes.com/sites/marcochiappetta/2025/08/31/ibm-and-amd-collaborate-on-new-hybrid-quantum-supercomputing-architectures

[138] Demonstrating a true realization of quantum-centric supercomputing. URL: https://www.ibm.com/quantum/blog/supercomputing-24

[139] Taking Neuromorphic Computing to the Next Level with Loihi 2. *Intel Corporation.* URL: https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf

[140] AutoML.org. Neural Architecture Search Overview. URL: https://www.automl.org/nas-overview

та скоротити час інференції, що особливо важливо для архітектур програмного забезпечення та хмарних сервісів[141].

Архітектури цифрових двійників (digital twins) для неперервної оптимізації стають потужним патерном у сучасному проєктуванні. Цифровий двійник – це віртуальна модель фізичного об'єкта або процесу, яка дозволяє проводити симуляції в реальному часі, тестувати зміни без ризику та переносити успішні рішення у продуктивне середовище. ШІ забезпечує постійну оптимізацію цифрового двійника, що сприяє підвищенню ефективності. Наприклад, компанія Mercedes-Benz застосовує цифрових двійників для архітектури заводів, досягаючи значного покращення продуктивності та гнучкості виробничих процесів[142]

Загалом майбутні напрями досліджень у сфері архітектурного проєктування із застосуванням ШІ демонструють високий потенціал трансформації галузі. Квантові та нейроморфні обчислення, AutoML і цифрові двійники не лише розширюють технічні можливості, а й формують нові парадигми архітектурного мислення: від адаптивності до енергоефективності та неперервної оптимізації. Подальші дослідження мають зосередитися на інтеграції цих технологій у практичні інженерні процеси, розробленні етичних і правових рамок їх застосування, а також на забезпеченні прозорості, відтворюваності та стійкості архітектурних рішень у складних цифрових екосистемах.

2.6.5 Прогнози розвитку

Згідно зі звітом McKinsey Technology Trends Outlook 2025[143] штучний інтелект матиме щорічний економічний ефект у межах $15,5-22,9 трлн до 2040 року. Для програмної архітектури це означає фундаментальну трансформацію – перехід від систем, спроєктованих людиною, до таких, що є ШІ-оптимізованими. Очікується, що до 2030 року понад 80% архітектурних рішень прийматиметься за участю ШІ. Це змінить роль архітектора як стратегічного координатора, а не виключно технічного розробника.

Формується нова парадигма *Software 3.0*, в якій ШІ виступає не просто інструментом, а активним учасником створення ПЗ. Традиційні системи на основі коду (*Software 1.0*) та моделі машинного навчання (*Software 2.0*) еволюціонують у ШІ-архітектурні, самозмінні системи. Ранні приклади стосуються самовідновлюваної інфраструктури, автоматично масштабованих застосунків та систем, здатних до автономної адаптації на основі змін середовища виконання[144].

Очікується також *консолідація архітектурних патернів*. Замість сотень варіацій, оптимізація за допомогою ШІ призведе до формування 20-30 стандартних патернів, які покриватимуть до 90% типових сценаріїв використання до 2028

---

[141] Neural Architecture Search Algorithm. *GeeksforGeeks.* URL: https://www.geeksforgeeks.org/deep-learning/neural-architecture-and-search-methods

[142] Gupta N. Mercedes-Benz: Revolutionizing Automotive with Digital Twins. *LinkedIn Pulse*, 2025. URL: https://www.linkedin.com/pulse/mercedes-benz-revolutionizing-automotive-digital-twins-neha-gupta-laoec

[143] McKinsey Technology Trends Outlook 2025. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-top-trends-in-tech

[144] Karpathy A. Software 2.0. *Medium*. URL: https://karpathy.medium.com/software-2-0-a64152b37c35

року[145]. Це суттєво спростить процес ухвалення рішень, зменшить ризики проєктування та підвищить узгодженість між командами.

Фах «Архітектор ШІ» формується як нова міждисциплінарна спеціалізація. Це професіонали, які поєднують знання про можливості ШІ з глибоким розумінням архітектурних принципів. Університети вже відкривають освітні програми за спеціальністю «Архітектура ШІ», а сертифікаційні курси перебувають у стадії розроблення. За прогнозами до 2027 року «Архітектор ШІ» увійде до десятки найзатребуваніших технічних професій за рівнем попиту та винагороди[146].

На рис. 2.4 показані вірогідні шляхи розвитку застосування ШІ в архітектурному проєктуванні.

Рисунок 2.4 – Дорожня карта розвитку ШІ в архітектурному проєктуванні

Прогнозовані зміни в архітектурному проєктуванні під впливом ШІ мають системний характер. Вони охоплюють як технологічні аспекти (автоматизацію, адаптивність, масштабованість), так і професійні трансформації, включно з появою нових ролей та освітніх траєкторій. Очікувана консолідація патернів, еволюція парадигм програмування та економічний ефект ШІ свідчать про потребу стратегічної підготовки до нової епохи архітектурного мислення.

2.7 Висновки до розділу

Дослідження ролі ШІ в архітектурному проєктуванні ПЗ демонструє фундаментальну трансформацію галузі. ШІ не лише автоматизує наявні процеси, а й змінює саму парадигму архітектурного проєктування, переводячи його з області мистецтва у сферу точної науки, підкріпленої даними та алгоритмами.

Ключові досягнення впровадження ШІ в архітектурне проєктування стосуються значного прискорення процесів розроблення, скорочення витрат на технічний борг та досягнення прийнятної точності в автоматичному виборі архітектурних патернів. Ці показники підтверджені реальними кейсами провідних

[145] Architectural Patterns Consolidation. *Thoughtworks*. *Technology Radar*. 2025. Vol. 32. URL: https://www.thoughtworks.com/radar/techniques/architectural-patterns-consolidation

[146] Okeseeyin T. Job Trends 2025: Top 20 Fastest Growing Jobs. *LinkedIn Pulse*, 2025. URL: https://www.linkedin.com/pulse/job-trends-2025-top-20-fastest-growing-jobs-temitope-okeseeyin-2cs5e

технологічних компаній: від 90% скорочення інфраструктурних витрат Amazon Prime Video до оброблення трильйонів подій щодня платформою Uber.

Методи та інструменти ШІ для архітектурного проєктування показали ефективність на різних масштабах проєктів: від простих ШІ-помічників для малих команд до розвинених ML-платформ для підприємств. Технологія адаптується до потреб та обмежень кожної організації. Особливо важливо, що ROI від впровадження ШІ є позитивним, навіть для малих команд з інвестиціями від $10 000 на рік.

Аналіз придатності ШІ-підходів показав, що оптимальна стратегія залежить від розміру команди, бюджету та регуляторних вимог. Малі команди отримують найбільшу вигоду від готових рішень, середні – від гібридних підходів, великі підприємства – від власних ШІ-платформ. Критичні системи потребують спеціальних підходів для відповідності регуляціям, але також виграють від можливостей ШІ у керуванні ризиками та забезпеченні якості.

Серед викликів залишаються проблеми довіри та інтерпретованості ШІ-рішень, етичні питання автоматизації та технологічні обмеження сучасних систем. Проте перспективи розвитку, включно з інтеграцією квантових обчислень, нейроморфними архітектурами та парадигмою Software 3.0, обіцяють подальшу революцію в архітектурному проєктуванні.

Практичні рекомендації для організацій, які планують впровадження ШІ в архітектуру, стосуються поступового підходу з початковим фокусом на високоефективні інструменти, інвестицій у навчання команди, створення систем управління використанням ШІ та збереження людського контролю для критичних рішень. ШІ слід розглядати не як заміну архітекторам, а як потужний інструмент для підсилення їхніх можливостей.

Майбутнє архітектурного проєктування поза сумнівом пов’язане зі штучним інтелектом. Організації, які інтегрують ШІ у свої процеси вже сьогодні, матимуть значну конкурентну перевагу завтра. Враховуючи, що лише 1% компаній досягли зрілості у впровадженні ШІ, потенціал для зростання та інновацій у цій сфері є величезним.

3 Штучний інтелект у тестуванні програмного забезпечення

3.1 Тестування за допомогою штучного інтелекту: основні завдання та інструменти

Актуальність використання ШІ для тестування програмного забезпечення (ПЗ) зумовлена стрімким зростанням складності сучасних програмних систем, високими вимогами до якості, швидкістю розроблення та обмеженими людськими ресурсами. Наразі більшість ІТ-компаній працюють за гнучкими методологіями (Agile, DevOps), де релізи відбуваються часто, а час на тестування суттєво обмежений. Це породжує потребу в автоматизації не лише виконання тестів, а й створенні, підтримці та аналізу результатів – саме ці завдання здатен ефективно виконувати ШІ.

Завдяки можливостям машинного навчання, нейромереж та оброблення природної мови, ШІ дозволяє розширити традиційні підходи до тестування. Наприклад, системи на базі ШІ можуть автоматично генерувати тест-кейси на основі технічної документації, аналізувати логи для пошуку помилок, прогнозувати дефекти ще до їх появи та адаптивно оновлювати тестові сценарії при зміні інтерфейсу. Загалом за допомогою ШІ можуть бути ефективно розв’язано різні завдання тестування ПЗ (табл. 3.1). Це значно знижує витрати часу і підвищує покриття тестами, що критично важливо в умовах високої конкуренції та швидкого розвитку технологій.

Таблиця 3.1 – Завдання тестування ПЗ, які можуть вирішуватись за допомогою інструментів ШІ

№	Завдання	Що робить ШІ	Приклади інструментів	Моделі / Технології
1	Генерація тест-кейсів	Автоматичне створення тестів з вимог або коду	TestRigor, Diffblue, Model-Based Testing Tools	NLP, Rule-based ML, GPT-моделі
2	Пріоритизація тестів	Визначає найбільш критичні тести для запуску	Testim, Launchable	Decision Trees, Logistic Regression
3	Виявлення дефектів	Пошук аномалій у поведінці системи	Sentry, DeepCode, CodeGuru	Anomaly Detection, CNN, Random Forest
4	Аналіз причин помилок	Визначає, що саме викликало збій	Microsoft Application Insights, Kibana + ML Plugins	Root Cause Analysis, Bayesian Networks
5	Прогнозування дефектів	Визначає, де найімовірніше виникнуть помилки	SonarQube (з ML-плагінами), CodeScene	Naive Bayes, Gradient Boosting
6	Аналіз впливу змін (Change Impact)	Які тести зачіпають зміни у коді	SonarQube, CodeScene, EWM, Telerik Test Studio	Graph ML, Change Propagation Models

Закінчення табл. 3.1

№	Завдання	Що робить ШІ	Приклади інструментів	Моделі / Технології
7	Візуальне тестування (UI)	Виявлення змін у зовнішньому вигляді	Applitools, Percy, Visual AI	Computer Vision, CNN, Template Matching
8	Розумна автоматизація тестування	Автоматичне створення/підтримка тестів	Test.ai, Applitools, Functionize	Reinforcement Learning, Pattern Recognition
9	Аналіз продуктивності	Визначення вузьких місць системи	Dynatrace, New Relic, LoadNinja	Time-Series ML, K-means, Autoencoders
10	Обробка зворотного зв'язку	Аналіз фідбеку, багрепортів, відгуків	MonkeyLearn, Zendesk, Lexalytics	Sentiment Analysis, Topic Modeling (LDA)

3.1.1 Використання ШІ для генерації тест-кейсів

ШІ має великий потенціал у генерації тест-кейсів для ПЗ, що є однією з найважливіших задач автоматизованого тестування. Процес генерації тестів зазвичай стосується створення набору перевірок, які дозволяють виявити можливі дефекти у системі. Традиційно це робилося вручну, але зі зростанням складності програмних систем і обсягу тестових сценаріїв, а також із потребою в швидких релізах, застосування ШІ стає дедалі важливішим.

Одним з основних способів використання ШІ для генерації тест-кейсів є застосування алгоритмів машинного навчання, зокрема, методів оброблення природної мови (NLP). Це дозволяє штучному інтелекту аналізувати технічну документацію, вимоги користувачів, специфікації та навіть код ПЗ для автоматичного створення тестових сценаріїв, які можуть перевіряти не лише основні функціональні можливості, а й більш складні випадки використання.

Наприклад, інструмент **TestRigor** (https://www.testrigor.com) використовує методи оброблення природної мови для генерації тест-кейсів з текстових описів вимог або користувацьких сценаріїв. Це дозволяє створювати тести без потреби писати код вручну, що значно спрощує процес автоматизації тестування, знижує вартість і час, необхідні для створення тестів, та дозволяє швидко адаптувати тестування до змін у вимогах.

Іншим прикладом є інструмент **Diffblue** (https://www.diffblue.com), який використовує машинне навчання для автоматичної генерації юніт-тестів на основі вихідного коду програми. Diffblue може аналізувати структуру коду, виявляти патерни і на основі цих даних генерувати тести, які покривають основні функції програми. Це дозволяє зекономити час розробників і тестувальників, оскільки більше не потрібно вручну створювати ці тестові сценарії для кожної функції.

Застосування ШІ для генерації тест-кейсів не лише підвищує ефективність, а й забезпечує більш високий рівень покриття тестами. ШІ здатен генерувати випадки тестування, які людина може пропустити, особливо коли мова йде про нестандартні сценарії або крайні випадки, які можуть стати причиною

дефектів у системі[147]. Це важливо, оскільки часто саме такі крайні випадки є джерелом найскладніших помилок у програмному забезпеченні.

Також варто зазначити, що генерація тестів за допомогою ШІ може бути адаптована до змін у програмному продукті. Це дозволяє зменшити навантаження на тестувальників, які повинні оновлювати тестові сценарії кожного разу, коли програма змінюється. Алгоритми ШІ можуть автоматично оновлювати тестові сценарії залежно від змін у коді або вимогах, що значно підвищує гнучкість і скорочує час циклів тестування.

Технології машинного навчання також дають можливість створювати більш ефективні та адаптивні методи для генерації тест-кейсів, таких як прогнозування дефектів або вибір найбільш важливих для тестування компонентів програми[148]. Такі підходи дозволяють зосередитися на найбільш ризикованих або складних частинах ПЗ, забезпечуючи високу якість продукту за менший час.

Тим самим, застосування ШІ для генерації тест-кейсів є важливим кроком до автоматизації тестування в умовах сучасних швидких циклів розроблення, який дозволяє підвищити ефективність, точність і масштабованість процесу тестування ПЗ. ШІ-інструменти, як-от TestRigor або Diffblue, допомагають компаніям скоротити витрати на тестування та збільшити швидкість виявлення дефектів.

3.1.2 Пріоритизація тестів

Пріоритизація тестів є важливою задачею в автоматизації тестування програмного забезпечення, яка дозволяє зосередити ресурси на найбільш важливих і ризикованих аспектах системи. Через обмежений час і ресурси тестування не може покривати всю функціональність програми, тому правильна пріоритизація тестів допомагає зосередити увагу на найбільш критичних частинах коду, які мають найвищу ймовірність дефектів[149]. Тут штучний інтелект, зокрема алгоритми машинного навчання, може значно покращити процес.

ШІ використовує історичні дані, щоб прогнозувати, які саме тести можуть виявити дефекти в програмі. Алгоритми машинного навчання, зокрема методи класифікації, можуть аналізувати попередні результати тестів, а також параметри коду, які змінювались під час останніх оновлень, щоб визначити, які тести є найбільш важливими для запуску. Наприклад, якщо певні модулі або функції часто спричиняли дефекти в минулому, система на основі цього аналізу пріоритизує тестування саме цих частин у майбутньому.

Один з найбільш відомих прикладів застосування ШІ для пріоритизації тестів – це використання системи **Launchable** (https://www.launchableinc.com). Вона застосовує алгоритми машинного навчання для аналізу історії виконання

---

[147] Islam M., Khan F., Alam S., Hasan M. Artificial Intelligence in Software Testing: A Systematic Review. *IEEE Region 10 Conference (TENCON'2023*). DOI: https://doi.org/10.1109/TENCON58879.2023.10322349

[148] Wang J., Suleiman B., Alibasa M. Automated Unit Test Case Generation: A Systematic Literature Review. *arXiv*. Vol. 2504.20357. DOI: https://doi.org/10.48550/arXiv.2504.20357

[149] Sharma M., Agrawal A. Test Case Design and Test Case Prioritization using Machine Learning. *International Journal of Engineering and Advanced Technology*. 2019. Vol. 9, Issue-1. P. 2742-2748. DOI: https://doi.org/10.35940/ijeat.A9762.109119

тестів і визначення, які з них найімовірніше виявлять дефекти після внесення змін до коду. Launchable прогнозує, які тести слід виконати, виходячи з даних про зміни в ПЗ, що дозволяє скоротити час на тестування і одночасно забезпечити високу якість. Система працює за принципом «test impact analysis», що дозволяє вибирати тільки ті тести, які мають найбільшу ймовірність виявлення помилок, знижуючи тим самим навантаження на систему тестування.

Ще одним прикладом є **Jenkins** (https://www.jenkins.io) у поєднанні з плагінами для аналізу впливу змін на тестування. Вони дозволяють запускати лише ті тести, які можуть бути безпосередньо пов'язані з нещодавніми змінами в коді. Завдяки використанню машинного навчання та аналізу метрик коду, Jenkins може інтелектуально визначити, які частини програми є найбільш вразливими до нових змін, і лише ці тести будуть пріоритетними. Таке рішення дає змогу суттєво зменшити час на тестування, забезпечуючи при цьому високу ймовірність виявлення критичних помилок.

Іншим прикладом є система **Testim** (https://www.testim.io), яка також використовує ШІ для оптимізації тестування. Вона аналізує результати попередніх тестів і на їх основі вирішує, які сценарії мають найбільшу ймовірність виявити помилки. Testim використовує методи машинного навчання для автоматичного виявлення найбільш важливих тестових кейсів, а також їх пріоритизації залежно від ризику.

Завдяки таким системам, як Launchable, Testim і Jenkins з плагінами для аналізу впливу змін, компанії можуть значно підвищити ефективність своїх тестових циклів, зменшуючи витрати на ресурси та час, який потрібен для виконання тестів. Замість того щоб запускати всі тести, ШІ дозволяє фокусуватися на найбільш критичних і релевантних, що особливо важливо в умовах швидкої доставки програмного забезпечення і частих змін.

3.1.3 ШІ для виявлення дефектів

Виявлення дефектів через пошук аномалій у поведінці системи є важливою задачею в тестуванні ПЗ, і ШІ може значно покращити цей процес, застосовуючи алгоритми машинного навчання та аналізу великих даних. В основі використання ШІ для виявлення дефектів лежить здатність аналізувати величезні обсяги даних, таких як логи, метрики, трасування запитів, і виявляти аномальні патерни поведінки, які можуть свідчити про потенційні помилки чи дефекти в ПЗ.

ШІ здатен виявляти дефекти не лише через наявні прямі сигнали про помилки (несправності), а й на основі відхилень від нормальної поведінки системи. Для цього використовуються методи машинного навчання, зокрема методи класифікації та кластеризації, які дозволяють виділити з масиву даних ті випадки, які не відповідають зазвичай спостережуваному поведінковому патерну[150].

[150] Natha S. A Systematic Review of Anomaly detection using Machine and Deep Learning Techniques. *Quaid-e-Awam University Research Journal of Engineering, Science & Technology*. 2022. Vol. 20. P. 83-94. DOI: https://doi.org/10.52584/QRJ.2001.11

Одним з прикладів використання ШІ для виявлення аномалій є інструмент **Sentry** (https://sentry.io), який застосовує алгоритми машинного навчання для автоматичного виявлення помилок у реальному часі. Sentry збирає дані про виконання застосунків, аналізує логи і трафік, а потім за допомогою алгоритмів виявляє аномальні патерни, які можуть свідчити про помилки в програмі. Такі аномалії можуть бути неочевидними для традиційних систем моніторингу, але ШІ здатен їх виявити, завдяки навчанню на великому масиві історичних даних. Важливою особливістю Sentry є те, що система не лише виявляє аномалії, а й дозволяє відстежити точні місця в коді, де сталася помилка, що дозволяє швидше усунути проблему.

Іншим популярним інструментом є **DeepCode** (https://www.deepcode.ai), який використовує ШІ для аналізу коду та виявлення помилок до того, як вони призведуть до збоїв у роботі програми. DeepCode застосовує нейромережі для перевірки стилю коду, логічних помилок та потенційних дефектів, пропонує рекомендації для їх виправлення. Це дозволяє тестувальникам і розробникам виявляти помилки ще на етапі написання коду, що значно зменшує ймовірність дефектів на більш пізніх етапах тестування.

Ще одним прикладом є інструмент **CodeGuru** від Amazon (https://aws.amazon.com/codeguru), який застосовує методи машинного навчання для аналізу програмного коду та виявлення можливих помилок або неефективних конструкцій. CodeGuru здатен автоматично знаходити проблеми, пов’язані з продуктивністю, безпекою та логікою, і пропонувати розробникам покращення. Це допомагає виявляти дефекти на ранніх етапах розроблення, мінімізуючи їх вплив на готовий продукт.

Використання ШІ для пошуку аномалій у поведінці системи дозволяє значно підвищити ефективність тестування, оскільки алгоритми машинного навчання здатні враховувати величезні обсяги даних та з високою точністю виявляти помилки, які можуть бути пропущені традиційними методами. Такі системи можуть працювати в реальному часі, що дозволяє оперативно реагувати на помилки і мінімізувати ризики в процесі розроблення і тестування.

Загалом застосування ШІ для виявлення дефектів через пошук аномалій дозволяє не лише автоматизувати процес тестування, а й значно покращити якість ПЗ, оскільки дозволяє виявляти і виправляти проблеми на ранніх етапах розроблення, зменшуючи кількість дефектів у кінцевому продукті.

3.1.4 Аналіз причин помилок у ПЗ

Аналіз причин помилок (root cause analysis) є важливою складовою процесу тестування ПЗ, адже без чіткої ідентифікації першопричини дефекту неможливо ефективно усунути проблему. ШІ може значно полегшити цей процес, використовуючи потужні методи аналізу даних, машинного навчання та оброблення природної мови для виявлення причин помилок і аномалій у системі.

Застосування ШІ для аналізу причин помилок дозволяє автоматично обробляти великі обсяги логів, трасувань, історії багів і навіть коментарів розробників для виявлення закономірностей і залежностей між різними частинами

коду і помилками. За допомогою таких алгоритмів, як класифікація та кластеризація, можна визначити зв'язок між конкретними змінами в коді і появою багів, що дає змогу швидко виявити джерело помилки.

Наприклад, інструмент **Rollbar** (https://rollbar.com) застосовує ШІ для аналізу багів і помилок у ПЗ. Він автоматично класифікує помилки на основі їхнього впливу і аналізує історію дефектів, щоб знайти закономірності, які можуть допомогти визначити першопричину проблеми. Rollbar використовує алгоритми машинного навчання для виявлення аномальних патернів у логах і надає детальну інформацію про те, що могло викликати конкретну помилку. Це дозволяє розробникам швидко зосередитись на вирішенні проблеми, не витрачаючи час на ручний аналіз та діагностику.

Ще одним корисним інструментом є **SonarQube** (https://www.sonarqube.org), який не лише виконує статичний аналіз коду для виявлення дефектів, а й застосовує алгоритми для аналізу причин помилок. SonarQube використовує методи машинного навчання для класифікації помилок за рівнем складності та потенційною шкодою для проєкту. Він може виявляти не тільки помилки синтаксису чи логіки, а й більш складні проблеми, наприклад, дефекти, які виникають через неправильне використання ресурсів або неточні алгоритми. У разі виявлення багів система надає рекомендації щодо їхнього виправлення, включно з аналізом причин і можливими варіантами рішень.

Ще один інструмент, який активно використовує ШІ для аналізу причин помилок, – це **Azure DevOps** (https://azure.microsoft.com/en-us/services/devops) від Microsoft. Azure DevOps вбудовує інструменти для автоматичного аналізу коду, включно з виявленням помилок і визначенням їхніх причин. Зокрема, система використовує технології машинного навчання для автоматичного виявлення кореляцій між різними компонентами програми та помилками, що дозволяє визначити, який саме модуль або частина коду є джерелом проблеми.

Аналіз причин помилок за допомогою ШІ також допомагає зменшити людські помилки в процесі тестування, оскільки системи на основі алгоритмів машинного навчання здатні аналізувати величезну кількість факторів одночасно, що є надзвичайно важким для людини. Крім того, такі системи можуть працювати в реальному часі, швидко виявляючи зміни в коді або нові дефекти, що виникають під час розроблення[151].

Застосування ШІ для аналізу причин помилок дозволяє значно скоротити час, необхідний для їхнього виявлення та усунення, а також підвищити точність діагностики. Це особливо важливо в умовах швидких релізів і постійно змінюваних вимог, коли будь-яка затримка або невиявлений дефект можуть призвести до серйозних проблем у роботі програмного забезпечення.

Крім того, ШІ допомагає долати людський фактор – суб'єктивність, втому, обмеження у знаннях. Завдяки використанню великих обсягів історичних даних, інтелектуальні системи здатні виявляти шаблони помилок, які залишаються поза увагою тестувальників. Це особливо цінно в масштабних

[151] Chen Y., Xie H., Ma M., et al. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys'24)*. 2024. P. 674-688. DOI: https://doi.org/10.1145/3627703.3629553.

розподілених системах, де комплексність перевірки ручними засобами практично неможлива.

3.1.5 Інструменти ШІ для прогнозування дефектів

Прогнозування дефектів у тестуванні ПЗ за допомогою ШІ сьогодні є важливою частиною процесу забезпечення якості ПЗ[152]. Ідея полягає в тому, щоб за допомогою аналізу великих обсягів даних – історії дефектів, коду, результатів тестів – ШІ міг прогнозувати, де ймовірно виникнуть помилки в майбутньому. Для цього застосовуються різні техніки машинного навчання, які допомагають підвищити ефективність тестування, зменшити кількість дефектів і скоротити час на їх виявлення та виправлення.

Одним із популярних інструментів з технологіями ШІ для прогнозування дефектів є **SonarQube** (https://www.sonarqube.org). Це система для статичного аналізу коду, яка може виявляти потенційні проблеми на ранніх етапах розроблення. У сучасних версіях SonarQube активно використовуються плагіни з підтримкою машинного навчання, що дозволяє не лише знаходити помилки, а й прогнозувати їх ймовірність. Використання ШІ дозволяє аналізувати не лише код, а й з'ясувати, де найбільша ймовірність виникнення дефектів у майбутньому. Зокрема, за допомогою модулів машинного навчання можна навчати SonarQube на попередніх даних для визначення патернів дефектів у новому коді. Це дає можливість автоматично зосереджувати увагу на найбільш вразливих частинах програми.

Іншим потужним інструментом для прогнозування дефектів є **CodeScene** (https://codescene.com), який активно використовує ШІ та машинне навчання для аналізу якості коду. CodeScene не тільки робить звичайний статичний аналіз коду, а й на основі даних про зміни в коді, історії комітів і тестових результатів прогнозує можливі дефекти. Він здатний виявляти «гарячі точки» в коді – ті частини, які мають найвищий ризик дефектів, і надає відповідні рекомендації для покращення якості коду. Однією з особливостей CodeScene є його здатність прогнозувати дефекти, орієнтуючись на патерни змін у коді, що можуть бути індикаторами майбутніх проблем.

Ще один інструмент, який стає все популярнішим у цій сфері, – це **DeepCode** (https://www.deepcode.ai). Цей інструмент використовує алгоритми глибокого навчання для аналізу коду та виявлення дефектів на основі їхнього контексту. DeepCode здатний працювати з різними мовами програмування та інтегрується в середовища розробки GitHub, GitLab і Bitbucket. Ідея полягає в тому, щоб навчити модель на основі великої кількості відкритих репозиторіїв та проєктів, щоб вона могла надавати інтелектуальні рекомендації по покращенню коду.

Є й інші інструменти для прогнозування дефектів, які використовують більш складні методи машинного навчання, зокрема на базі алгоритмів

[152] Тестування та забезпечення якості програмних систем : навч.-метод. посіб. [Електронне видання] / О.Г. Трофименко, А.І. Дика; Нац. ун-т «Одес. юрид. академія». Одеса: Фенікс, 2024. 140 с. URL: https://hdl.handle.net/11300/27246.

класифікації Support Vector Machines (SVM), випадковий ліс (Random Forest, RF), логістичної регресії (LR), градієнтного підвищення рівня (Boosting GB), K-найближчих сусідів (KNN), дерева рішень (DT), класифікатора XGB тощо[153]. Наприклад, робота розглядає різні моделі, які можуть бути застосовані для прогнозування дефектів, включаючи нейронні мережі, дерева рішень та інші алгоритми. Це дає можливість прогнозувати дефекти не тільки в рамках окремих модулів, а й на рівні всього проєкту, що дозволяє більш точно визначити пріоритети для тестування.

Такі підходи значно полегшують роботу команд тестувальників і розробників, оскільки дозволяють зменшити час на ручне тестування та забезпечити більш точне виявлення помилок. Використання ШІ в поєднанні з інструментами, як-от: SonarQube, CodeScene, DeepCode та іншими, дозволяє автоматизувати процеси тестування та значно покращити якість продукту.

Загалом використання ШІ для прогнозування дефектів у тестуванні ПЗ – це важливий крок до підвищення ефективності тестування та забезпечення високої якості ПЗ. Вибір інструментів залежить від специфіки проєкту, типу коду і рівня складності, однак всі ці ШІ-інструменти дають змогу значно знизити ймовірність виникнення дефектів у фінальному продукті.

3.1.6 Аналіз впливу змін

Аналіз впливу змін (Change Impact Analysis, CIA) є важливим етапом у процесі тестування ПЗ, оскільки він дозволяє оцінити, які частини системи можуть бути порушені внаслідок внесення змін у код. Цей процес критичний для оптимізації тестування, оскільки допомагає зосередити ресурси на тих частинах програми, які можуть бути найбільш вразливими до дефектів через зміни. Використання ШІ в аналізі впливу змін дозволяє автоматизувати і підвищити точність цього процесу, що знижує витрати часу і зусиль при тестуванні.

ШІ може бути використаний для аналізу кодових змін та їх потенційного впливу на інші частини програми через різні підходи, зокрема, за допомогою машинного навчання, глибокого навчання, а також методів природного мовлення для аналізу документації і коментарів. Основна ідея полягає в тому, щоб навчити моделі машинного навчання на основі історичних даних про зміни в коді, тестових результатах і дефектах, щоб вона могла передбачити, які частини програми будуть найбільш вразливими до нових змін.

Одним із таких інструментів є **SonarQube** (https://www.sonarqube.org), який аналізує код і може інтегрувати плагіни з машинним навчанням для автоматизованого виявлення потенційно вразливих місць після змін у коді. SonarQube використовує техніки статичного аналізу коду, щоб визначити, як зміни в одній частині програми можуть вплинути на інші частини[154]. Завдяки

[153] Kabir H., Rahman T., Mridul A. Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms. *Smart Wearable Technology*. 2025. P. 1-15. DOI: https://doi.org/10.47852/bonviewSWT52025645

[154] Трофименко О. Г. Інструменти статичного тестування безпеки. *Інформаційні технології: моделі, алгоритми, системи (ITMAS-2024):* матер. V Всеукр. наук.-практ. інтернет-конф. (30-31 жовтня 2024 р.) Миколаїв. С. 188-190. URL: https://itconf.nuos.edu.ua/2024/publications/static-security-testing-tools/

інтеграції з плагінами для машинного навчання «SonarQube Machine Learning Plugin» можна отримати більш точні передбачення щодо впливу змін, аналізуючи не лише структуру коду, а й історичні дані про помилки та їх залежність від типу змін. Це дозволяє фокусувати увагу на найбільш вразливих частинах програми після внесення змін.

Іншим важливим інструментом є **CodeScene** (https://codescene.com), який використовує підходи на основі ШІ для аналізу змін у коді та їхнього впливу. CodeScene аналізує історію змін у коді, визначає те, як нові коміти можуть вплинути на інші частини системи. За допомогою технік машинного навчання він здатен створювати карти «гарячих точок», які вказують на області коду з високим ризиком. Тим самим CodeScene може допомогти автоматично ідентифікувати частини коду, найбільш вразливі до дефектів після внесення змін, а також рекомендувати, де слід зосередити зусилля тестувальників.

Інший інструмент, який активно використовує ШІ для аналізу впливу змін, – це **IBM Engineering Workflow Management (EWM)** (https://www.ibm.com). Ця платформа дозволяє здійснювати комплексний моніторинг змін у коді та автоматично оцінювати, як ці зміни можуть вплинути на інші частини проєкту. Використовуючи ШІ для аналізу комітів та історії версій, EWM прогнозує ризики, пов’язані з новими змінами, і надає аналітику, яка допомагає тестувальникам та розробникам зосередити зусилля на критичних ділянках.

Ще одним прикладом є **Telerik Test Studio** (https://www.telerik.com/teststudio), який використовує ШІ для інтелектуального планування тестів на основі змін у програмному забезпеченні. Використовуючи історичні дані про дефекти та зміни в коді, інструмент автоматично генерує тестові сценарії, що з максимальною ймовірністю дозволяють виявити дефекти, що виникають після змін. Це дозволяє значно знизити кількість неактуальних тестів та підвищити ефективність процесу тестування.

Дослідження[155] показує, що на основі попереднього аналізу тестів і змін у коді, ШІ може передбачити, які тестові випадки будуть найбільш релевантними для виконання після змін. Це зменшує необхідність в ручному відборі тестів і дозволяє більш точно планувати тестування.

Застосування ШІ в аналізі впливу змін дозволяє автоматизувати процеси, які раніше вимагали значних людських ресурсів, а також забезпечити більшу точність при визначенні найбільш вразливих частин програмного забезпечення після внесення змін. Інструменти SonarQube, CodeScene, IBM Engineering Workflow Management та Telerik Test Studio використовують різні підходи для інтеграції ШІ, дозволяючи зосередити зусилля тестувальників на тих частинах коду, де зміни можуть викликати найбільший вплив.

[155] Parekh K. Next-Gen Quality Assurance: Leveraging AI, Automation, and DevOps for Scalable Software Excellence. *International Research Journal on Advanced Engineering and Management (IRJAEM)*. 2025. Vol. 3. P. 2345-2351. DOI: https://doi.org/10.47392/IRJAEM.2025.0370.

3.1.7 CV-підхід для візуального тестування

Візуальне тестування (Visual Testing) у контексті тестування ПЗ – це метод, який дозволяє перевіряти графічний інтерфейс користувача (Graphical User Interface, GUI) на наявність дефектів за допомогою зображень і відео, порівнюючи очікувані та фактичні результати. Зазвичай це важливо при тестуванні веб та мобільних застосунків, де зовнішній вигляд і інтерфейс мають великий вплив на користувацький досвід. Використання ШІ, зокрема комп'ютерного зору (Computer Vision, CV), у візуальному тестуванні дозволяє автоматизувати цей процес і робить його значно ефективнішим, особливо за умов великих і складних систем, де вручну перевіряти всі елементи інтерфейсу дуже трудомістко.

ШІ та комп'ютерний зір використовуються для порівняння візуальних елементів у тестованому програмному забезпеченні з еталонними зображеннями або шаблонами. Алгоритми машинного навчання можуть допомогти у виявленні відмінностей, навіть якщо вони є незначними, а також визначити, чи відповідає фактичний вигляд програми вимогам до дизайну.

Інструменти, які використовують комп'ютерний зір для візуального тестування, зазвичай мають функції порівняння зображень, детекції змін, а також здатність враховувати фактори зміни в розмірах елементів, масштабування тощо. Одним із таких інструментів є **Applitools Eyes** (https://applitools.com), який використовує передові технології комп'ютерного зору для автоматичного виявлення візуальних дефектів. Applitools Eyes здатний порівнювати зображення за допомогою алгоритмів глибокого навчання, що дозволяє йому розпізнавати зміни, які можуть бути не помітні для людини, але важливі для користувача. Наприклад, інтерфейс може виглядати схожим на екрані, але мати невеликі зміни у відображенні на різних пристроях або роздільних здатностях, які можуть вплинути на досвід користувача.

Іншим прикладом інструмента для візуального тестування є **Percy** (https://percy.io), який теж використовує комп'ютерний зір для автоматизації візуальних перевірок. Percy фокусується на автоматичному порівнянні вебсторінок і мобільних застосунків на основі візуальних змін. Percy використовує візуальні шаблони для перевірки, як зміни в коді (наприклад, зміни в стилях CSS або компонентах UI) впливають на вигляд застосунку чи вебсторінки. Один з основних аспектів Percy – це інтеграція з різними системами контролю версій та CI/CD, що дозволяє здійснювати автоматичні візуальні перевірки під час кожного коміту чи розгортання на етапі CI.

Ще одним популярним інструментом для автоматизованого візуального тестування є **VisualTesting.ai** (https://www.visualtesting.ai). Цей інструмент використовує комп'ютерний зір і технології машинного навчання для ідентифікації візуальних дефектів на вебсторінках або в мобільних застосунках. VisualTesting.ai здатний порівнювати екранні знімки після змін і виявляти проблеми, які можуть вплинути на зовнішній вигляд застосунку. Крім того, він дозволяє налаштовувати параметри порівняння для врахування аспектів адаптивного дизайну чи то змін у зовнішньому вигляді між різними платформами.

Науковці розглядають різні алгоритми використання комп'ютерного зору для автоматизованого тестування візуальних аспектів застосунків: детекція

контурів, шаблонне порівняння, нейронні мережі, але всі погоджуються з тим, що інструменти на базі ШІ можуть значно покращити точність і ефективність тестування, дозволяючи швидко виявляти відмінності між очікуваними і фактичними результатами візуальних тестів, навіть у складних інтерфейсах[156].

Загалом, застосування ШІ та комп'ютерного зору для візуального тестування дозволяє значно підвищити точність і швидкість тестування інтерфейсів. Інструменти, такі як Applitools Eyes, Percy та VisualTesting.ai, використовують передові технології ШІ для виявлення візуальних дефектів, що допомагає автоматизувати виявлення проблем на етапі тестування і зменшити час, витрачений на ручну перевірку інтерфейсів.

3.1.8 Розумна автоматизація тестування

Застосування ШІ до автоматизації тестування дозволяє значно підвищити ефективність та точність тестових процесів, зменшуючи час та ресурси, необхідні для виконання тестів, і дозволяючи швидше виявляти помилки. В основі цього підходу лежить використання алгоритмів машинного навчання та глибокого навчання, які допомагають створювати «розумні» тести, які адаптуються до змін у ПЗ та автоматично генерують відповідні тестові сценарії.

Основним підходом до розумної автоматизації тестування є використання ШІ для створення тестових сценаріїв, прогнозування дефектів та адаптації тестових наборів залежно від змін у ПЗ. Традиційні методи автоматизації, як-от запис і відтворення тестів, часто вимагають значних зусиль для адаптації до нових функціональних змін у програмі. Однак з використанням ШІ процес створення та підтримки тестів стає значно ефективнішим завдяки здатності ШІ розпізнавати шаблони і автоматично генерувати тестові випадки на основі виявлених змін.

Одним із популярних інструментів для реалізації розумної автоматизації тестування є **Test.ai** (https://www.test.ai), який використовує ШІ для автоматичного створення тестових сценаріїв. Test.ai застосовує машинне навчання для аналізу інтерфейсу користувача та автоматичного генерування тестових сценаріїв на основі взаємодії з елементами інтерфейсу. Цей інструмент здатний вивчати поведінку користувача і генерувати тестові сценарії, які можуть охоплювати найбільш критичні функціональні аспекти ПЗ. Важливою особливістю Test.ai є його здатність адаптуватися до змін в інтерфейсі та автоматично оновлювати тестові випадки, зменшуючи потребу в ручному налаштуванні тестів.

Іншим прикладом застосування технології ШІ для візуальної автоматизації тестування є **Applitools Ultrafast Test Cloud** (https://applitools.com). Applitools використовує алгоритми комп'ютерного зору та машинного навчання для порівняння візуальних елементів інтерфейсу з еталонними зображеннями, а також для автоматичної адаптації до змін в інтерфейсі. Завдяки цьому інструмент автоматично виявляє дефекти, які стосуються візуальних змін у інтерфейсі, і скорочує час на ручну перевірку. Applitools Ultrafast Test Cloud

[156] Wotawa F., Klampfl L., Jahaj L. A framework for the automation of testing computer vision systems. *arXiv*. 2021. DOI: https://doi.org/10.48550/arXiv.2105.04383

дозволяє знизити витрати часу на тестування, оскільки автоматично виявляє навіть дрібні зміни у графічному інтерфейсі.

Functionize (https://www.functionize.com) – ще один інструмент, який використовує ШІ для автоматизації тестування. Functionize поєднує машинне навчання з традиційними методами автоматизованого тестування для адаптивного створення тестів. Цей інструмент дозволяє автоматично генерувати тестові випадки на основі даних про змінений функціонал, розпізнаючи зміни на вебсторінці або в інтерфейсі й адаптуючи тести відповідно до цих змін. Functionize використовує алгоритми ШІ для моделювання користувацьких сценаріїв й автоматичної корекції тестових випадків, що дає змогу створювати тестові сценарії без потреби вручну прописувати кожен з них.

Крім того, інструменти Mabl (https://www.mabl.com) і Testim (https://www.testim.io) використовують моделі машинного навчання для покращення автоматизованих тестів, їхнього підтримання та інтелектуального виконання. Вони відрізняються підходом до покращення автоматизації тестів, зокрема, можуть автоматично коригувати тести та адаптувати їх на основі змін у поведінці програми чи інтерфейсі. Завдяки ML Mabl здатний коригувати тестові сценарії в разі змін у вигляді інтерфейсу або нових функцій, що є корисним при тестуванні вебзастосунків, де постійно змінюється дизайн та інтерфейс. А Testim може автоматично генерувати тести, визначати нові шляхи взаємодії з інтерфейсами, навіть якщо були внесені зміни в код. Тому ці інструменти, хоча й використовують ШІ, більше орієнтовані на автоматизацію через аналіз та адаптацію тестів у реальному часі, а не на розумну автоматизацію тестування, як це роблять Test.ai або Functionize, де основна увага приділяється створенню тестів на основі змін і прогнозуванню дефектів.

ШІ дозволяє значно підвищити точність і швидкість тестування, оскільки машини можуть швидко адаптуватися до змін у коді і автоматично генерувати нові тести, які враховують найновіші зміни в програмному забезпеченні. Крім того, використання ШІ дозволяє не лише автоматично генерувати тестові сценарії, а й зменшити кількість помилок у тестах, оскільки алгоритми можуть навчатися на основі попередніх даних про дефекти і вибирати найбільш ефективні стратегії для тестування[157].

Загалом, використання ШІ для розумної автоматизації тестування дозволяє значно підвищити ефективність тестування, зменшити час на створення і виконання тестів, а також забезпечити більш точні результати. Інструменти, як-от Test.ai, Applitools, Functionize, допомагають автоматизувати створення тестів та адаптувати їх до змін у ПЗ. Використання машинного навчання для автоматизації тестування не лише покращує ефективність, а й дозволяє знижувати ймовірність дефектів, підвищуючи якість програмного забезпечення.

[157] Thomas A. Review of AI-Driven Approaches for Automated Defect Detection and Classification in Software. Testing. *International Journal of Research and Review*. 2025. Vol. 12; Issue 6. P. 154-160. DOI: https://doi.org/10.52403/ijrr.20250619

3.1.9 Аналіз продуктивності

Аналіз продуктивності є одним із важливих етапів тестування ПЗ, який стосується перевірки ефективності системи за параметрами швидкості роботи, відгуку, масштабованості та стійкості до навантажень. Використання ШІ в аналізі продуктивності дозволяє автоматизувати цей процес, зробити його більш точним та ефективним. ШІ може бути використаний для моніторингу та прогнозування продуктивності, виявлення аномалій і навіть для автоматичного налаштування тестових сценаріїв на основі аналізу результатів попередніх тестів.

Одним з основних способів використання ШІ для аналізу продуктивності є автоматичне збирання і оброблення даних про продуктивність під час виконання тестів, зокрема виявлення аномалій і прогнозування проблем до їх виникнення. Моделі машинного навчання можуть аналізувати великі обсяги даних, щоб ідентифікувати слабкі місця в архітектурі застосунку, які можуть негативно вплинути на продуктивність при високих навантаженнях.

Інструменти, які використовують ШІ для аналізу продуктивності, можуть збирати інформацію з системи моніторингу (наприклад, зібрані дані про час відповіді, використання ресурсів, пропускну здатність), а потім застосовувати алгоритми для автоматичного виявлення патернів. Це дозволяє тестувальникам швидко виявляти потенційні проблеми без потреби виконувати аналіз вручну. Наприклад, **Dynatrace** (https://www.dynatrace.com) використовує ШІ для моніторингу продуктивності в реальному часі. Цей інструмент застосовує алгоритми машинного навчання для аналізу зібраних даних про продуктивність і виявлення аномалій, наприклад, зниження швидкості роботи або підвищене навантаження на сервери. Dynatrace автоматично визначає, які компоненти системи потребують уваги, і надає детальну інформацію про проблеми.

Іншим важливим інструментом, який також використовує ШІ для автоматизованого аналізу продуктивності вебзастосунків, є **LoadNinja** (https://www.loadninja.com). LoadNinja дозволяє створювати реалістичні тести для перевірки масштабованості системи, а ШІ допомагає оптимізувати ці тести, прогнозуючи навантаження і ефективність на основі попередніх результатів. LoadNinja використовує дані про тестування під високими навантаженнями, щоб автоматично коригувати сценарії і тестувати систему за найбільш реалістичних умов. Це дозволяє більш точно і швидко оцінити продуктивність програми в умовах навантажень.

Ще одним інструментом, який активно використовує ШІ в аналізі продуктивності, є **New Relic** (https://newrelic.com). New Relic застосовує технології машинного навчання для аналізу даних продуктивності в реальному часі, зокрема для визначення патернів та аномалій у поведінці системи. Він допомагає виявити, як різні компоненти ПЗ впливають на загальну продуктивність, і дозволяє тестувальникам швидко реагувати на потенційні проблеми. New Relic фокусується на глибокому аналізі продуктивності програм, які працюють на різних платформах і використовують мікросервісну архітектуру.

Прогнозування дефектів ПЗ на ранніх етапах розроблення не лише підвищує якість та надійність ПЗ, а й знижує вартість розроблення. Для створення моделей прогнозування дефектів ПЗ можна використовувати широкий спектр

методів машинного навчання, але ефективність та точність цих моделей часто залежать від вибору відповідної підмножини ознак. Оскільки пошук оптимальної підмножини ознак є обчислювально ресурсоємним, для визначення близьких до оптимальних рішень протягом розумних термінів зазвичай використовуються евристичні та метаевристичні підходи[158].

ШІ дозволяє значно покращити процес тестування продуктивності за допомогою автоматизації аналізу результатів тестів, виявлення аномалій і прогнозування потенційних проблем. Інструменти на кшталт Dynatrace, LoadNinja та New Relic використовують ці технології для забезпечення ефективного моніторингу і аналізу продуктивності ПЗ у реальному часі. Наукові дослідження підтверджують, що використання ШІ для аналізу продуктивності є важливим етапом у розвитку тестування ПЗ, оскільки дозволяє швидше і точніше визначати потенційні слабкі місця системи, знижуючи ризик виникнення серйозних проблем на етапі розгортання в реальних умовах.

3.1.10 Обробка зворотного зв'язку

Обробка зворотного зв'язку є важливим аспектом тестування ПЗ, оскільки дозволяє збирати, аналізувати й використовувати відгуки від користувачів, тестувальників та автоматизованих систем для покращення якості продукту. Адже відгуки користувачів стають дедалі важливішим джерелом інформації для розроблення вимог, дизайну інтерфейсів користувача та розроблення ПЗ. Наразі відгуки користувачів легкодоступні в соціальних мережах, на форумах продуктів або в магазинах застосунків.

За останнє десятиліття дослідження[159] показали, що відгуки користувачів можуть допомогти командам розробників ПЗ:

а) краще зрозуміти, як користувачі насправді використовують певні функції та компоненти продукту;

б) швидше виявляти, відтворювати та виправляти дефекти;

в) отримувати натхнення для покращень або нових функцій.

Однак, щоб повністю реалізувати потенціал відгуків, треба розв'язати дві основні проблеми. По-перше, постачальники ПЗ повинні справлятися з великою кількістю даних відгуків, якими важко керувати вручну. По-друге, постачальники також повинні справлятися з різною якістю відгуків, оскільки деякі елементи можуть бути неінформативними, повторюваними або просто неправильними.

ШІ може значно підвищити ефективність цього процесу, автоматизувати аналіз великої кількості відгуків, виділення важливих патернів і надання рекомендацій щодо подальших дій. Використання ШІ для оброблення зворотного зв'язку дозволяє швидше реагувати на проблеми, які виникають у продукті, та значно покращити процес ітераційного розвитку ПЗ.

---

[158] Mandal K., Nadim M., Chanchal R., Banani R., Schneider K. Evaluating the Performance of a D-Wave Quantum Annealing System for Feature Subset Selection in Software Defect Prediction. *arXiv*. 2024. DOI: https://doi.org/10.48550/arXiv.2410.16469.

[159] Maalej W., Biryuk V., Wei J., Panse F. On the Automated Processing of User Feedback. *arXiv*. 2024. DOI: https://doi.org/10.48550/arXiv.2407.15519

Основним підходом є застосування методів оброблення природної мови (Natural Language Processing, NLP), машинного навчання та алгоритмів глибокого навчання для автоматичного аналізу відгуків, виявлення ключових проблем та класифікації зворотного зв'язку за важливістю. Всі ці методи дозволяють заощадити значний час тестувальників, оскільки вони здатні автоматично розпізнавати теми, визначати тональність відгуків і навіть виділяти ті частини продукту, які найбільше потребують покращення.

Так, інструмент **MonkeyLearn** (https://monkeylearn.com) використовує алгоритми оброблення природної мови для автоматичного аналізу текстових відгуків користувачів. MonkeyLearn дозволяє створювати кастомізовані моделі для класифікації відгуків на основі їхнього змісту, наприклад, визначати, чи є відгук позитивним або негативним, чи вказує він на певні дефекти в продукті. Крім того, інструмент може автоматично виділяти основні теми, що дозволяє швидко зібрати інформацію про ключові проблеми в ПЗ. Застосування таких інструментів дозволяє компаніям ефективно обробляти великі обсяги зворотного зв'язку та оптимізувати процес реагування на проблеми.

Lexalytics (https://www.lexalytics.com) є потужним інструментом, який використовує технології NLP для автоматичного оброблення зворотного зв'язку. Цей інструмент дозволяє аналізувати відгуки, надані користувачами або тестувальниками, визначати настрої та класифікувати їх за типом (позитивний, негативний, нейтральний) і за темами. Використання машинного навчання для аналізу зворотного зв'язку дозволяє створити автоматизовану систему, здатну швидко реагувати на зміни у відгуках і надавати рекомендації щодо вдосконалення продукту. Lexalytics дозволяє користувачам створювати моделі для автоматичної класифікації зворотного зв'язку та виявлення важливих патернів, що дає змогу оперативно реагувати на основні питання або дефекти.

Zendesk (https://www.zendesk.com) – ще один інструмент, який застосовує ШІ для оброблення зворотного зв'язку. Основна мета Zendesk – це централізоване збирання та аналіз усіх зворотних зв'язків від користувачів через різні канали (електронну пошту, чат, соціальні мережі). Алгоритми машинного навчання, вбудовані в Zendesk, допомагають автоматично класифікувати запити та відгуки користувачів, визначати найбільш важливі питання та пріоритети, що дозволяє автоматизувати реагування на запити та пришвидшити процес виправлення помилок або вдосконалення функціоналу продукту. Це корисно при інтеграції з іншими системами тестування, позаяк дозволяє швидко отримувати й обробляти зворотний зв'язок.

ШІ активно застосовується для аналізу багатоканального зворотного зв'язку, включно з текстовими та голосовими відгуками. Системи на основі **Google Cloud Natural Language API** (https://cloud.google.com/natural-language) дозволяють автоматично аналізувати текстові дані, виділяти ключові слова, визначати тональність відгуків і з'ясовувати, чи є відгуки конструктивними або просто негативними. Інструменти Google Cloud використовують алгоритми глибокого навчання для більш точного визначення контексту та забезпечують високий рівень точності в обробленні тексту.

Використання ШІ для оброблення зворотного зв'язку в тестуванні ПЗ дозволяє значно підвищити ефективність аналізу та зменшити час на ручне

оброблення відгуків. Застосування оброблення природної мови для аналізу настроїв у відгуках користувачів і швидкого виявлення негативних або позитивних патернів у зворотному зв'язку дозволяє швидше адаптувати програму до вимог користувачів. Інструменти MonkeyLearn, Lexalytics, Zendesk і Google Cloud NLP допомагають автоматизувати процеси збору та аналізу відгуків, дозволяючи тестувальникам швидко реагувати на проблеми та покращувати якість продукту на основі отриманих відгуків.

Загалом, ШІ суттєво підвищує ефективність тестування: автоматизує рутинні та аналітичні задачі, економить ресурси, покращує якість і дозволяє сфокусуватися на стратегічних типах тестування.

3.2 Види тестування ПЗ, де застосовується ШІ

Штучний інтелект має значний потенціал для вдосконалення процесів тестування ПЗ, оскільки його застосування дозволяє автоматизувати, оптимізувати та підвищити точність тестування на різних етапах життєвого циклу програмного продукту. Завдяки можливостям машинного навчання, оброблення природної мови, комп'ютерного зору та інших технологій, ШІ сприяє не лише зниженню людської праці, а й підвищенню ефективності тестування. Системи тестування з ШІ дозволяють досягти кращих результатів у таких аспектах, як функціональне тестування, тестування інтерфейсу, продуктивності, безпеки та зворотного зв'язку (рис. 3.1).

Рисунок 3.1. Види тестування ПЗ, де можуть застосовуватися інструменти ШІ

3.2.1 Функціональне тестування

Функціональне тестування (Functional Testing) перевіряє відповідність роботи функцій ПЗ згідно з вимогами. Застосування ШІ дозволяє автоматизувати створення тестових сценаріїв, генерувати тестові випадки на основі моделювання реальних користувацьких дій і попередніх тестових результатів. Алгоритми машинного навчання здатні адаптувати тестування до змін у функціональності програми, що робить процес тестування більш гнучким та ефективним[160]. Наприклад, інструменти Test.ai використовують методи ШІ для створення та автоматичного виконання тестів, що оптимізує процес перевірки функціональних характеристик ПЗ і виявляє проблеми на ранніх етапах розроблення.

3.2.2 Тестування інтерфейсу користувача

Тестування інтерфейсу користувача (UI Testing) з використанням ШІ є важливим аспектом забезпечення якості ПЗ[161]. Завдяки технологіям комп'ютерного зору (CV), ШІ може імітувати поведінку користувача, перевіряючи правильність відображення елементів інтерфейсу, таких як кнопки, форми, меню тощо, на різних пристроях і екранах з різною роздільною здатністю. Алгоритми глибокого навчання дозволяють автоматично виявляти візуальні дефекти та розбіжності, що значно зменшує час і зусилля, необхідні для ручної перевірки інтерфейсу. Наприклад, інструменти Applitools (https://applitools.com) використовують ШІ для візуального тестування для порівняння екранних зображень та автоматичного визначення відмінностей при перевірці графічних елементів у різних версіях продукту.

3.2.3 Тестування продуктивності

ШІ активно застосовується для тестування продуктивності ПЗ (Performance Testing) за умов різних навантажень. Алгоритми машинного навчання використовують для аналізу великих обсягів даних, отриманих під час тестування навантаження, що дозволяє автоматично виявляти аномалії в показниках продуктивності та прогнозувати можливі проблеми в реальному часі, наприклад, затримки в роботі або перевантаження серверів. Крім того, ШІ може сприяти виявленню «вузьких місць» в архітектурі системи, які впливають на її продуктивність. Dynatrace (https://www.dynatrace.com), наприклад, застосовує ШІ для моніторингу та аналізу продуктивності програмних систем у реальному часі, автоматично виявляє аномалії, передбачає можливі відмови системи і навіть пропонує рішення для оптимізації, що дозволяє своєчасно реагувати на проблеми в продуктивності.

---

[160] Karhu K., Kasurinen J., Smolander K. Expectations vs Reality – A Secondary Study on AI Adoption in Software Testing. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2504.04921

[161] Тестування та забезпечення якості програмних систем : навч. посібник [Електронне видання] / О.Г. Трофименко, А.І. Дика; Нац. ун-т «Одес. юрид. академія». Одеса: Фенікс, 2024. 195 с. URL: https://hdl.handle.net/11300/27717 . ISBN 978-617-8395-88-9

3.2.4 Тестування безпеки

ШІ відіграє важливу роль у тестуванні безпеки ПЗ (Security Testing), позаяк дозволяє виявляти вразливості, які можуть бути використані для кібератак (детальніше див. підрозд. 3.3). Алгоритми машинного навчання здатні створювати інтелектуальні системи, які можуть навчатися на прикладах атак і виявляти нові, раніше невідомі вразливості, аналізувати поведінку системи та виявляти аномалії, характерні для злочинних дій, як-от: SQL-ін'єкції, XSS-атаки (Cross Site Scripting, міжсайтовий скриптинг) тощо. Застосування ШІ в цій сфері дозволяє значно знизити ризики і покращити ефективність тестування безпеки. Так, інструмент Darktrace (https://www.darktrace.com) використовує методи ШІ для моніторингу, виявлення та нейтралізації загроз у реальному часі, застосовує підходи машинного навчання для автоматичної ідентифікації підозрілих патернів та потенційних загроз.

3.2.5 Тестування на сумісність

Для тестування сумісності (Compatibility Testing) ПЗ з різними браузерами, операційними системами та платформами ШІ застосовують для автоматизації перевірки сумісності на численних пристроях і в різних середовищах. Алгоритми ШІ допомагають створювати та виконувати тестові сценарії, які покривають різноманітні комбінації платформ, тим самим підвищують якість тестування. Завдяки машинному навчанню інструменти можуть самостійно створювати тестові середовища, аналізувати їх взаємодії та виявляти можливі проблеми сумісності. BrowserStack (https://www.browserstack.com/) є прикладом інструментів, які використовують інтелектуальні алгоритми для перевірки роботи вебзастосунків у різних браузерах та операційних системах, що дозволяє зменшити час тестування сумісності й підвищити його точність.

3.2.6 Тестування навантаження та стрес-тестування

Для тестування систем під великими навантаженнями (Load and Stress Testing) ШІ здатен допомогти з прогнозуванням, плануванням та аналізом результатів тестування. Алгоритми машинного навчання використовуються для моделювання реальних умов роботи програми під різними рівнями навантаження, виявлення точок відмов та передбачення поведінки системи в критичних ситуаціях. ШІ дозволяє створювати більш реалістичні сценарії для стрес-тестування, базуючись на історичних даних та попередніх тестах. LoadNinja (https://loadninja.com/), наприклад, застосовує методи ШІ для автоматичного створення реалістичних сценаріїв і тестів під великими навантаженнями та прогнозування реакції системи. Цей інструмент аналізує систему та передбачає, як вона реагуватиме на різні рівні навантаження, що дозволяє знайти потенційно слабкі місця.

3.2.7 Обробка зворотного зв'язку

ШІ є важливим інструментом для автоматизації оброблення зворотного зв'язку (Feedback Testing) від користувачів та тестувальників. Алгоритми оброблення природної мови (NLP, Natural Language Processing) дозволяють автоматично класифікувати відгуки за темами (наприклад, баги, запити на функціональність), визначати їхню тональність і виділяти ключові проблеми, які можуть бути важливими для подальшого розроблення продукту. Моделі машинного навчання здатні прогнозувати, які відгуки найважливіші, і надавати рекомендації для подальших дій. Це дозволяє визначати пріоритети для розв'язання проблем, швидко реагувати на виявлені проблеми та оптимізувати процес виправлення помилок. MonkeyLearn (https://monkeylearn.com) і Lexalytics (https://www.lexalytics.com) є прикладами інструментів для автоматичної оброблення зворотного зв'язку, які використовують ШІ для аналізу відгуків, визначення тональності текстів та виявлення ключових проблем. Ці інструменти використовують алгоритми машинного навчання для аналізу текстових відгуків й автоматичної класифікації проблем за категоріями, що значно полегшує процес оброблення великої кількості даних.

Отже, ШІ має потужний потенціал для вдосконалення процесів тестування ПЗ, оскільки дозволяє автоматизувати, прискорювати та покращувати багато аспектів тестування. Переваги використання ШІ в тестуванні ПЗ стосуються автоматизації тестування, поліпшення ефективності та точності тестів, а також здатності до адаптації в реальному часі до змін у програмному продукті. Алгоритми машинного навчання, оброблення природної мови, комп'ютерного зору та інші технології дозволяють значно зменшити трудовитрати та прискорити виявлення дефектів, що робить тестування ПЗ більш надійним і продуктивним. Інструменти, як-от: Test.ai, Applitools, Dynatrace, BrowserStack, LoadNinja, Darktrace, активно впроваджуються в індустрії для покращення якості тестування, що підвищує ефективність та швидкість розроблення програмних продуктів.

3.3 ШІ у тестуванні безпеки

3.3.1 Роль ШІ у тестуванні безпеки вебзастосунків

Тестування безпеки вебзастосунків є важливою складовою забезпечення якості ПЗ. З розвитком кіберзагроз, з одного боку, та зростанням складності вебзастосунків, з іншого, традиційні методи тестування безпеки подекуди виявляються недостатніми для ефективного виявлення вразливостей ПЗ. Останнім часом все частіше для тестування ПЗ пропонують застосовувати ШІ. Адже він пропонує нові підходи до генерації тестових даних, що може значно підвищити ефективність і точність тестування безпеки. Основні методи ШІ охоплюють машинне, генетичні алгоритми, еволюційні стратегії, агентні моделі та інші інноваційні підходи.

Машинне навчання (МН, Machine Learning, ML) є доволі ефективною технологією для генерації тестових даних. Алгоритми МН можуть аналізувати великі обсяги даних та виявляти патерни, які можуть бути використані для створення нових тестових випадків. У дослідженні[162] оцінюються можливості генеративного ШІ для формування тестових даних у різних доменах. Сформовано три типи запитів до великих мовних моделей (LLMs), які виконують завдання генерації тестових даних на різних рівнях інтеграції: 1) генерація сирих тестових даних; 2) синтез програм конкретною мовою, які генерують корисні тестові дані; 3) створення програм, які використовують сучасні бібліотеки-фейкери. Застосування глибокого навчання для виявлення вразливостей дозволило отримати результати, значно вищої точності, аніж результати тестування, досягнуті традиційними методами.

Генетичні алгоритми (ГА, Genetic Algorithm, GA) є еволюційним методом, що використовується для пошуку оптимальних рішень за допомогою природного відбору та мутацій. ГА можуть бути ефективно застосовані для автоматичної генерації тестових даних, оскільки вони можуть створювати складні, непередбачувані сценарії тестування з урахуванням вразливостей у вебзастосунках. Системи з ГА для створення складних сценаріїв атаки виявилися ефективними для виявлення вразливостей, які не були виявлені традиційними методами[163].

Використання **агентних моделей** (Agent-based Models) є ще одним підходом, що використовується для генерації тестових даних. Ця техніка моделювання створює віртуальну екосистему агентів, взаємодія яких породжує складні структури даних, тим самим віддзеркалюючи їхні людські аналоги в реальних програмах[164]. У контексті тестування безпеки вебзастосунків агентні моделі можуть бути використані для симуляції поведінки різних типів користувачів та потенційних зловмисників. Це дозволяє створювати реалістичні сценарії взаємодії з вебзастосунком, що допомагає виявляти вразливості, які можуть бути пропущені традиційними методами.

За даними звіту IBM 2023 року[165] середня економія для організацій, які широко використовують ШІ для автоматизації тестування безпеки, становила 1,76 млн доларів США, порівняно з організаціями, які цього не робили. Адже використання ШІ для генерації тестових даних дозволяє автоматизувати процес створення складних, реалістичних сценаріїв тестування, що підвищує ефективність тестування безпеки[166].

---

[162] Baudry B., Etemadi K., Fang S., Gamage Y., Liu Y., Liu Y. Generative AI to Generate Test Data Generators. *arXiv e-prints*, arXiv-2401. 2024. DOI: https://doi.org/10.48550/arXiv.2401.17626. URL: https://arxiv.org/pdf/2401.17626

[163] Esnaashari M., Damia A. H. Automation of software test data generation using genetic algorithm and reinforcement learning. *Expert Systems with Applications*. 2021. Vol. 183. Issue C. P. 115446. DOI: https://doi.org/10.1016/j.eswa.2021.115446

[164] Clark A., Walkinshaw N., Hierons R. Test case generation for agent-based models: a systematic literature review. *Information and Software Technology*. 2021. Vol. 135. P. 106567. DOI: https://doi.org/10.1016/j.infsof.2021.106567

[165] Average duration of downtime after a ransomware attack at organizations in the United States from 1st quarter 2020 to 2nd quarter 2022. URL: https://www.statista.com/statistics/1275029/length-of-downtime-after-ransomware-attack-us/

[166] 27 Eye-Opening Statistics About User Experience, Website First Impressions, and Website Design That You Should Be Taking Very Seriously. URL: https://www.sweor.com/firstimpressions

Однією з ключових переваг впровадження технологій ШІ у процес тестування програмного забезпечення є забезпечення *високого рівня реалістичності тестових даних*. Алгоритми ШІ здатні генерувати тестові сценарії, які імітують реальні моделі поведінки користувачів, з урахуванням різноманітних варіацій їхньої взаємодії із системою. Такий підхід дозволяє імітувати широкий спектр умов експлуатації, виявляти вразливості, які залишаються невидимими при використанні шаблонних або статичних тестових наборів. Наприклад, системи на базі генеративних моделей можуть створювати послідовності дій, які відображають поведінку користувача за умов навмисного або ненавмисного порушення логіки взаємодії, що особливо актуально для виявлення логічних помилок і проблем з UX-дизайном.

Іншим суттєвим фактором є *розширення охоплення тестами (test coverage)*, що досягається за рахунок автоматичної генерації великої кількості різноманітних тестових випадків. ШІ дозволяє ефективно покривати множину гілок виконання програми, включаючи рідкісні або граничні випадки, які зазвичай не враховуються в ручному тестуванні. Це особливо важливо для складних вебзастосунків або розподілених систем, де традиційні підходи до тестування виявляються обмеженими. Згідно з практикою DevOps-платформ, застосування ШІ у тестуванні дозволяє досягти покриття понад 90% гілок коду[167], що є критично важливим для безпеки та стабільності системи.

Ще однією вагомою перевагою є *адаптивність тестових даних*, яка забезпечується можливістю машинного навчання динамічно реагувати на зміни у програмному середовищі. У разі оновлення інтерфейсу або модифікації функціоналу вебзастосунку, алгоритми ШІ можуть автоматично оновлювати тестові сценарії, виявляти потенційні точки збоїв без потреби ручного втручання. Це забезпечує *гнучкість і масштабованість процесу тестування*, що є необхідним у контексті швидких релізів, притаманних гнучким методологіям розроблення (Agile, CI/CD).

Водночас, застосування ШІ у тестуванні не позбавлене певних *викликів і обмежень*. Зокрема, якість навчальних даних суттєво впливає на результативність системи. Недостатня кількість або репрезентативність вхідних даних може призвести до генерації неточних тестових випадків, що, у свою чергу, знижує ефективність виявлення помилок. Наприклад, при навчанні моделі на застарілих або неповних даних, алгоритм може не враховувати специфіку нових функцій або типових поведінкових сценаріїв користувача. Крім того, складність моделей ШІ є ще одним бар'єром до широкого впровадження. Налаштування, оптимізація та інтерпретація результатів роботи алгоритмів вимагають глибоких знань у галузях машинного навчання, програмної інженерії та статистики. Це створює додаткове навантаження на команди розробників і тестувальників та може бути неприйнятним для невеликих компаній з обмеженими людськими або фінансовими ресурсами. Окремої уваги заслуговує проблема фальшивих позитивних результатів (false positives), які можуть виникати унаслідок

[167] Дика А.І. Роль штучного інтелекту у тестуванні безпеки вебзастосунків. *Інформаційне суспільство: проблеми та перспективи* : матер. IX всеукр. наук.-практ. конф. (24 травня 2024). Одеса, 2024. С. 153-156. https://doi.org/10.32837/11300.27842

надмірної чутливості ШІ-системи до незначних відхилень у поведінці програмного забезпечення. Такі хибні спрацьовування призводять до витрат часу на їх верифікацію, що ускладнює процес прийняття рішень щодо дійсних дефектів і негативно впливає на загальну ефективність тестування.

ШІ не здатен замінити тестувальника у тестуванні зручності використання реального користувача, перевірити та проаналізувати дані і результати тестування. ГА не може замінити роботу фахівців з розроблення та тестування, яка вимагає спілкування між зацікавленими сторонами, чуйного ставлення до потреб користувачів і розроблення функцій, орієнтованих на користувача. Саме люди є творцями та валідаторами результатів ШІ, а не навпаки. Фахівці-практики не сприймають ГА як конкурента, адже ШІ не може замінити роботу, яка вимагає участі користувачів, і співпрацю між людьми. ГА може виконувати основні, повторювані, прості завдання, але не здатні замінити спільні зусилля, які вимагають більш систематичного та абстрактного мислення[168].

Отже, методи ШІ починають відігравати важливу роль в автоматизації генерації тестових даних для тестування безпеки вебзастосунків. Використання МН, ГА та агентних моделей дозволяє покращити ефективність і точність генерації тестових даних. Попри певні виклики щодо якості даних та складності моделей, які потребують вивчення та врахування, переваги використання ШІ у тестуванні безпеки є вагомими. Тому дослідження у цій галузі сприяють створенню більш інтелектуальних та адаптивних систем захисту, які можуть ефективно протистояти сучасним кіберзагрозам. Однак, впровадження ШІ у сферу тестування програмного забезпечення потребує зваженого підходу, врахування специфіки проєкту та наявності відповідної експертизи.

3.3.2 ChatGPT для тестування безпеки

Тестування безпеки вебзастосунків є важливою складовою процесу їх розроблення, оскільки дозволяє виявити й усунити потенційні вразливості перед введенням програмного забезпечення в експлуатацію. З появою ШІ стало можливим його використання і в сфері інформаційної безпеки, наприклад, для тестування безпеки вебзастосунків.

ChatGPT може допомогти виявити вразливості у вебзастосунках, у тому числі недостатньо захищені точки входу, неправильно налаштовані права доступу, а також відомі вразливості в бібліотеках та фреймворках. Шляхом аналізу текстової інформації модель може виявити такі вразливості та надати рекомендації щодо їх виправлення. Крім того, ChatGPT може імітувати взаємодію людини з програмними системами, виявляючи вразливості, які можуть бути пропущені традиційними методами тестування[169].

На рис. 3.2 наведено приклад використання ChatGPT для автоматизації процесу сканування за допомогою OWASP ZAP (Open Web Application Security

[168] Li J., Cao H., Lin L., Hou Y., Zhu R., Ali A. User Experience Design Professionals' Perceptions of Generative Artificial Intelligence. *Arxiv*. 2023. Vol. 1. P. 1-25. DOI: https://doi.org/10.48550/arXiv.2309.15237

[169] Дика А. І. Тестування безпеки вебзастосунків за допомогою ChatGPT. *Актуальні тенденції розвитку освіти, науки та технологій*: матер. VII міжнар. наук.-практ. конф. Харків-Бахмут, 15 травня 2024 р. Т. 2. С. 28-30. URL: https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi

Project Zed Attack Proxy)[170], інструмента виявлення широкого спектра вразливостей вебзастосунків.

Код з рис. 3.3 запускає OWASP ZAP через командний рядок та виконує сканування вебзастосунку. Результати сканування зберігаються у формі HTML-звіту. Перед виконанням такого сканування варто переконатися, що OWASP ZAP встановлено та налаштовано.

```python
import subprocess

def zap_scan(target_url):
    # Запускаємо сканування OWASP ZAP з використанням zapcli
    subprocess.run(['zap-cli', 'start', '-q'])
    subprocess.run(['zap-cli', 'open-url', target_url])
    # Виконуємо активне сканування
    subprocess.run(['zap-cli', 'active-scan', '--recursive', target_url])
    # Зберігаємо результати сканування в JSON файл
    subprocess.run(['zap-cli', 'report', 'html', '--output', 'zap_report.html'])
    # Закриваємо OWASP ZAP
    subprocess.run(['zap-cli', 'shutdown'])

# Приклад використання:
target_url = 'http://your-web-application-url'
zap_scan(target_url)
```

Рисунок 3.2. Скрипт мовою Python для автоматизації процесу сканування вразливостей

Крім виявлення вразливостей, ChatGPT також може аналізувати логи активності користувачів та виявляти аномальну поведінку, що може свідчити про спроби несанкціонованого доступу або про атаки з вразливостями. Шляхом постійного моніторингу системи, модель може оперативно реагувати на потенційні загрози та сповіщати про них адміністраторів (рис. 3.3)[171].

У цьому прикладі створено клас AnomalyDetectionModel, який містить PCA (Principal Component Analysis) та моделі Isolation Forest[172]. Модель навчається на вхідних даних і може виявляти аномалії на основі цих даних. Вхідні

[170] OWASP ZAP. URL: https://www.zaproxy.org/

[171] Дика А. І., Трофименко О. Г. Аналіз потенційних загроз засобами ChatGPT для тестування безпеки вебзастосунків. *Європейські орієнтири розвитку України: науково-практичний вимір в умовах воєнних викликів*: матер. міжнар. наук.-практ. конф. Одеса, 26 квітня 2024 р. Одеса: НУ «ОЮА», 2024. С.876-878. URL: https://hdl.handle.net/11300/28085

[172] Adithya K. Anomaly Detection with Isolation Forest & Visualization. Towards Data Science. 2019. URL: https://towardsdatascience.com/anomaly-detection-with-isolation-forest-visualization-23cd75c281e2

дані у прикладі створюються штучно, але на практиці вони можуть бути реальними даними системи.

У цьому експерименті ChatGPT успішно проаналізував код на наявність вразливостей, реалізував скрипт для автоматизації процесу сканування вебзастосунків на вразливості та провів виявлення аномалій на основі моделі Isolation Forest. Це охоплює генерацію окремих фрагментів коду Python для кожного з них.

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.ensemble import IsolationForest

class AnomalyDetectionModel:
    def __init__(self, n_components=3, contamination=0.1):
        self.n_components = n_components
        self.contamination = contamination
        self.pca = PCA(n_components=self.n_components)
        self.model = IsolationForest(contamination=self.contamination)

    def fit(self, X):
        X_pca = self.pca.fit_transform(X)
        self.model.fit(X_pca)

    def predict(self, X):
        X_pca = self.pca.transform(X)
        return self.model.predict(X_pca)

# Приклад використання:
# Генеруємо випадкові дані для навчання моделі
np.random.seed(42)
normal_data = np.random.normal(loc=0, scale=1, size=(1000, 5))  # Зразки "нормальних"
anomalous_data = np.random.normal(loc=10, scale=2, size=(50, 5))  # Зразки "аномальних

# Конкатенуємо дані
data = np.concatenate((normal_data, anomalous_data), axis=0)

# Створюємо та навчаємо модель
model = AnomalyDetectionModel(n_components=3, contamination=0.1)
model.fit(data)

# Передаємо дані для передбачення
predictions = model.predict(data)

# Виводимо результати
print("Попередній розподіл аномалій:")
print(predictions)
```

Рисунок 3.3. Скрипт мовою Python для виявлення аномальної поведінки

Однак, як показав проведений у роботі експеримент та дослідження інших науковців[173], для ChatGPT характерна проблема у правильності поєднання всіх фрагментів коду без людського втручання.

[173] Al-Hawawreh M., Aljuhani A., Jararweh Y. ChatGPT for cyberse-curity: practical applications, challenges, and future directions. *ClusterComputing*. 2023. Vol. 26, № 6. P. 3421-3436. DOI: https://doi.org/10.1007/s10586-023-04124-5

Вебзастосунки нині є невіддільною частиною сучасного цифрового світу. Зростання їх популярності не в останню чергу зумовило актуалізацію вимог до їх безпеки. Потенційно вебзастосунки можуть піддаватися різноманітним загрозам: атакам з використанням вразливостей, витокам конфіденційної інформації, атакам з метою отримання несанкціонованого доступу до системи тощо.

Моделі глибокого навчання, зокрема GPT (Generative Pre-trained Transformer), дозволяють створювати системи, здатні аналізувати текстові дані, що відкриває широкі можливості їх застосування для тестування безпеки. Дослідники[174] звернули увагу на можливості застосування ChatGPT для автоматизації процесу тестування безпеки вебзастосунків, завдяки його здатності розуміти контекст та в режимі реального часу аналізувати великі обсяги текстової інформації. ChatGPT добре інтегрується з різними інструментами автоматизації процесів, що дозволяє в режимі 24/7 ефективно виявляти потенційні загрози в сценаріях тестування безпеки.

ChatGPT може бути використаний для аналізу програмного коду, логів підключень під час тестування на вразливості вебзастосунків. Він може розпізнавати патерни, характерні для найпоширеніших видів атак – SQLi та XSS. Ба більше, крім аналізу даних, модель ChatGPT може генерувати такі типи атак та перевіряти реакцію системи на них. Це допомагає виявляти потенційні вразливості та перевіряти ефективність наявних у ПЗ захисних механізмів. Крім того, ChatGPT можна використовувати для моделювання взаємодії людини з програмними системами, що дозволяє виявляти вразливості, які традиційні методи тестування можуть пропустити. Це пояснюється тим, що ChatGPT базується на обробленні природної мови (NLP, Natural Language Processing), що дає йому змогу розуміти та обробляти людську мову. Це може допомогти тестувальникам безпеки у виявленні вразливостей, характерних для атак соціальної інженерії (байтінг, вішінг, претекстінг, приманка і підміна тощо).

Як приклад моделювання атаки соціальної інженерії засобами ChatGPT було створено сторінку для клікджекінгу. Клікджекінг-уразливість полягає в тому, щоб обманом змусити користувача натиснути на нешкідливу, на перший погляд, кнопку чи то посилання, яке насправді виконуватиме небажану дію, наприклад, завантажуватиме шкідливе ПЗ або відправлятиме зібрану особисту інформацію на сторонні вебресурси. На рис. 3.4 наведено код мовою Python для виявлення вразливості вебсайту до клікджекінгу за допомогою ChatGPT.

У проведеному експерименті ChatGPT успішно проаналізував код на наявні вразливості, реалізував скрипт для автоматизації процесу сканування вебзастосунків на вразливості.

Попри значні переваги, варто зважати та враховувати певні обмеження використання ChatGPT у тестуванні безпеки. Зокрема, модель може виявляти лише ті загрози, які вона була навчена розпізнавати, тому її ефективність залежить від якості навчання та обсягу доступної інформації. Крім того, варто уникати повного автоматизованого тестування безпеки, оскільки це може

[174] Як використовувати ChatGPT для ефективного тестування безпеки: вичерпний посібник. URL: https://brainberry.ua/uk/newsroom/blog/how-to-use-chatgpt-for-effective-security-testing-a-comprehensive-guide

призвести до пропуску певних видів атак або до виникнення помилкових позитивних спрацювань.

```python
import requests

def check_clickjacking(url):
    try:
        response = requests.get(url)
        headers = response.headers

        if 'X-Frame-Options' in headers:
            x_frame_options = headers['X-Frame-Options']
            if x_frame_options.lower() != 'deny' and x_frame_options.lower() != 'sameorigin':
                print("Potential clickjacking vulnerability detected (X-Frame-Options header):", x_frame_options)
            else:
                print("No potential clickjacking vulnerability detected (X-Frame-Options header):", x_frame_options)
        else:
            print("X-Frame-Options header is not present")

        if 'Content-Security-Policy' in headers:
            csp = headers['Content-Security-Policy']
            if 'frame-ancestors' not in csp or "'none'" in csp or "'self'" in csp:
                print("Potential clickjacking vulnerability detected (Content-Security-Policy header):", csp)
            else:
                print("No potential clickjacking vulnerability detected (Content-Security-Policy header):", csp)
        else:
            print("Content-Security-Policy header is not present")

    except requests.exceptions.RequestException as e:
        print("Error occurred while checking the URL:", e)

# Example usage:
check_clickjacking("https://example.com")
```

Рисунок 3.4. Скрипт для виявлення вразливості вебсайту до клікджекінгу

Використання штучного інтелекту, зокрема моделі ChatGPT, у тестуванні безпеки вебзастосунків відкриває широкі можливості для виявлення та запобігання різного роду загроз. Аналіз текстової інформації, виявлення нових вразливостей, сканування на виявлення аномальної поведінки, а також генерація тестових атак – це лише деякі з аспектів, в яких ChatGPT може бути корисний. Проте важливо пам’ятати про обмеження використання моделі та враховувати їх при плануванні та виконанні тестування безпеки.

3.4 Ключові виклики та стратегії вдосконалення тестування ПЗ за допомогою ШІ

Тестування програмного забезпечення, яке використовує алгоритми штучного інтелекту, характеризується специфічними труднощами, пов’язаними з непередбачуваністю результатів його роботи. Оскільки машинне навчання є базовою технологією для більшості систем ШІ, воно може генерувати неочікувані

наслідки під час практичного застосування. Це зумовлює потребу розроблення тестових сценаріїв, які моделюють широкий спектр можливих ситуацій та дозволяють оцінити реакцію системи на них. Коректність функціонування значною мірою залежить від якості навчання моделі та репрезентативності навчальних даних. Для одного й того самого набору даних різні моделі ШІ можуть демонструвати різний рівень результативності, а навіть у межах однієї моделі змінення навчальних або тестових вибірок здатне призвести до варіативності результатів[175]. Це створює додаткові виклики для розробників, оскільки ускладнює процес планування тестових випадків та формування тестових даних.

Важливим аспектом є також алгоритмічна нестабільність та невизначеність, що виникають через неповну специфікацію або нечіткість вимог. Тестування систем ШІ передбачає не лише виявлення помилок, але й аналіз механізмів ухвалення рішень. Використання методів моніторингу та візуалізації роботи алгоритмів сприяє підвищенню прозорості й полегшує інтерпретацію результатів тестування.

Додатковою проблемою є стохастичний характер алгоритмів, що застосовуються під час розроблення ШІ. Програміст не задає чітко визначений алгоритм, а створює код для навчання моделі на основі даних. Важливу роль відіграє вибір параметрів ініціалізації – кількості нейронів та початкових ваг, для яких не існує універсальної формули. Цей вибір суттєво впливає на подальший процес навчання та розвиток моделі. Стохастичність алгоритмів ускладнює застосування традиційних стратегій тестування, зокрема робить проблематичним обчислення показників тестового охоплення за методологією «білої скриньки»[176].

Додатковий виклик у тестуванні ШІ створює неоднорідність архітектур програм ШІ. Кожна така архітектура вимагає розуміння її внутрішньої логіки та параметрів для ефективного тестування. Тому важливим є високий рівень компетентності експертів під час розроблення тестових стратегій для різних типів нейромереж. Забезпечення відповідності кожної архітектури вимагає глибокого розуміння її унікальних властивостей та використання адаптованих методів тестування[177].

Найпростішим методом тестування програми ШІ є тестування методом чорної скриньки, оскільки таким методом тестувальники уникають перевірки складної внутрішньої роботи алгоритмів ШІ. Одним із найпопулярніших видів тестування методом «чорної скриньки» є тестування за специфікацією вимог. Однак навіть під час тестування програм ШІ за специфікацією виникають проблеми, зумовлені наявністю фундаментального конфлікту самого поняття вимог до ШІ. Це зумовлено тим, що програми ШІ призначені для створення їхньої узагальненої поведінки, тоді як вимога призначена для документування конкретної

---

[175] Sugali K., Sprunger C., Inukollu V. Software testing: issues and challenges of artificial intelligence & machine learning. *International Journal of Artificial Intelligence and Applications (IJAIA)*. 2021. Vol.12, No.1. P. 101-112. DOI: https://doi.org/10.5121/ijaia.2021.12107

[176] Zhuy H., Liu D., Bayley I., Harrison R., Cuzzolin F. Datamorphic Testing: A Methodology for Testing AI Applications. *Technical Report OBU-ECM-AFM-2018-02.* URL: https://arxiv.org/ftp/arxiv/papers/1912/1912.04900.pdf

[177] Дика А.І. Тестування штучного інтелекту: ключові виклики, стратегії вдосконалення. *Електронні інформаційні ресурси: створення, використання, доступ:* матер. міжнар. наук.-практ. інтернет-конф. Вінниця, 20-21 листопада 2023, КЗВО “Вінницька академія безперервної освіти”. С.93-95

поведінки. Специфікація вимог до ШІ за своєю природою намагається визначити загальну поведінку, а валідація програми ШІ визначає тестування якості передбачення результату виконання алгоритму або продуктивності алгоритму[178]. Виміряти якість передбачень у спосіб, який можна перевірити, вельми складно.

Неявні дефекти, які можуть виникати в програмах ШІ, створюють додаткові виклики для тестувальників, оскільки здебільшого вони є складними та непрозорими. Подібного роду дефекти можуть стосуватися невизначеності у прийнятті рішень моделлю або впливу змінення вхідних даних на результати. Такі неочевидні проблеми вимагають ретельного аналізу та вдосконалених методів тестування, спрямованих на виявлення внутрішніх аномалій. Тому для успішного тестування програм зі ШІ доцільно поєднувати традиційні підходи з інноваційними методами, які враховують специфіку нейромереж та їх потенційні неявні дефекти.

Стратегії для вдосконалення тестування ШІ мають охоплювати застосування різноманітних тестових наборів із широким спектром сценаріїв їхнього використання. Зазвичай дані розподіляють на тренувальний та тестовий набори в певному співвідношенні, наприклад: 80% – на тренування і 20% – на тестування. Це дозволяє моделі навчатися на певному наборі даних і перевіряти свою ефективність на інших, які не входять до навчальної вибірки. Навчальний та тестовий набори мають бути репрезентативними, тобто охоплювати розмаїття сценаріїв, з якими модель може стикнутися за реальних умов. Важливо враховувати різноманітність класів та уникати вибіркового розподілу, який може спричинити перекоси у навчанні моделі[179]. Також важливо впроваджувати автоматизовані засоби тестування, які дозволяють виявляти помилки та аномалії в реальному часі.

Не менш важливу роль у тестуванні ШІ відіграють етичні аспекти. Приміром, системи розпізнавання облич можуть бути вразливими до змін у зовнішньому вигляді, що порушує приватність користувачів. Тому важливо визначити та тестувати алгоритми, які забезпечуватимуть адекватний рівень захисту особистих даних й ефективно працюватимуть для різних сценаріїв. Команди розробників і тестувальників мають активно співпрацювати з експертами у сфері етики та безпеки для визначення потенційних ризиків і розроблення відповідних заходів. Усупереч викликам тестування ШІ є ключовим етапом у безпечному та ефективному впровадженні цієї технології. Інноваційні стратегії та системний підхід до тестування дозволяють покращити надійність та безпеку систем ШІ, забезпечуючи високу якість таких програмних продуктів.

---

[178] Bousquet L., Nakamura M. Improving Testability of Software Systems that Include a Leaning Feature. *The 10th International Conference on Advances in System Testing and Validation Lifecycle.* October, 14-18, 2018, Nice, France. URL: http://www27.cs.kobe-u.ac.jp/achieve/data/pdf/1333.pdf

[179] Testing the Intelligence of Your AI. EXACTPRO, 2019. URL: https://exactpro.com/ideas/white-papers/testing-intelligence-your-ai

3.5 Висновки до розділу

У сучасному ландшафті інформаційної безпеки та тестування програмного забезпечення застосування ШІ відіграє дедалі більш стратегічну роль. Завдяки можливостям машинного навчання, генетичних алгоритмів і агентних моделей, процес тестування набуває нової глибини: системи можуть автоматично генерувати дані, моделювати взаємодію користувачів і потенційних зловмисників, а також виявляти приховані вразливості. Такі підходи суттєво розширюють тестове охоплення та покращують точність виявлення дефектів, особливо у складних вебзастосунках та розподілених системах.

Попри це, випадковість, алгоритмічна нестабільність, нечіткі вимоги до ШІ-систем, а також складність у навчанні нейромереж, створюють значні виклики для фахівців[180]. Традиційні методи тестування, зокрема стратегія білої скриньки, часто виявляються непридатними через відсутність прозорої внутрішньої логіки моделей. Водночас тестування методом чорної скриньки дає змогу уникнути технічних труднощів, проте не завжди враховує глибинні особливості алгоритмів прийняття рішень.

Однією з важливих перспектив є поєднання класичних методів з інноваційними засобами тестування, що базуються на ШІ. Це дозволяє створювати гібридні системи, які адаптуються до змін у середовищі програмного забезпечення, автоматично оновлюють тестові сценарії та виявляють нові точки збоїв. Зокрема, модель ChatGPT демонструє здатність інтегруватися з такими інструментами як OWASP ZAP, виконувати аналіз коду, генерувати Python-скрипти, виявляти аномалії поведінки користувача та моделювати атаки соціальної інженерії. Це суттєво збагачує арсенал засобів тестувальника безпеки.

Разом з тим, якість навчальних даних, ризик фальшивих позитивних спрацювань і складність в інтерпретації результатів ШІ залишаються серйозними обмеженнями. Такі аспекти потребують ретельного планування, професійної експертизи та етичного контролю, оскільки автоматизація не здатна повністю замінити людську чуйність, комунікацію між зацікавленими сторонами та вміння враховувати контекст користувацького досвіду.

З огляду на все вищезазначене, можна стверджувати, що напрямок використання ШІ у тестуванні безпеки має високий потенціал для розвитку. Інтеграція ШІ у DevSecOps-процеси, розроблення пояснювальних моделей, впровадження адаптивних платформ та створення відкритих репозиторіїв тестових сценаріїв – усе це сприятиме формуванню більш ефективних, прозорих і надійних систем захисту.

[180] Трофименко О. Г., Соколов А. В., Чикунов П. О., Ахмаметьєва Г. В., Атанасевич А. О. Аналіз ролі фахівців із тестування та забезпечення якості. *Збірник наукових праць Національного університету кораблебудування імені адмірала Макарова*. 2024. № 3 (496). С. 106-113. DOI: https://doi.org/10.15589/znp2024.3(496).16.

4 Інтеграція штучного інтелекту в сучасну веброзробку

4.1 Роль і можливості ШІ у веброзробці

ШІ поступово перетворюється на фундаментальну технологію, здатну трансформувати всі аспекти цифрової взаємодії. Насамперед це стосується сфери веброзробки через технологічні можливості та методологічні зміни, що відбуваються у процесах створення, підтримки та оптимізації вебресурсів.

Основні напрями застосування ШІ у веброзробці та вебдизайні охоплюють автоматизацію процесів розроблення, підвищення ефективності UI/UX-дизайну, генерацію та персоналізацію контенту, адаптацію сайтів під індивідуальні потреби користувача, аналіз поведінкових патернів відвідувачів та впровадження інтелектуальних асистентів[181]. Усі ці напрями формують взаємопов'язану екосистему технологічних рішень, що забезпечують якісно новий рівень функціональності та користувацького досвіду.

4.1.1 Теоретичні засади використання ШІ у вебтехнологіях

Концептуальний апарат використання ШІ у вебтехнологіях базується на поєднанні алгоритмів машинного навчання, глибокого навчання, оброблення природної мови, комп'ютерного зору та інших компонентів ШІ у традиційні вебархітектури.

Теоретичною основою цього процесу є такі принципи:

– *адаптивність систем* – здатність вебзастосунків динамічна змінювати свою поведінку відповідно до контексту використання;

– *персоналізація інтерфейсів*, яка реалізується через аналіз користувацьких даних та застосування алгоритмів рекомендаційних систем. На відміну від статичних підходів, системи на основі ШІ здійснюють адаптацію у реальному часі, враховують не лише історичні дані, а й поточний контекст взаємодії користувача;

– *автоматизація процесів* розроблення на основі генеративних моделей стосується створення програмного коду, тестових сценаріїв і документації. Принцип ґрунтується на досягненнях у галузі оброблення природної мови та формальних методах програмування, що дозволяє системам ШІ розуміти семантику програмних конструкцій та генерувати функціонально адекватні рішення;

– *інтелектуалізація вебсередовища* – створення систем, здатних самостійно ухвалювати рішення щодо оптимізації продуктивності, безпеки та користувацького досвіду. Теоретичною основою цього принципу є методи автономних агентів та багатоагентних систем, адаптовані для специфіки вебархітектур.

[181] Alenezi M, Akour M. AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions. *Applied Sciences.* 2025. Vol. 15(3). https://doi.org/10.3390/app15031344

Наведені принципи визначають наукову парадигму, в якій розглядається ШІ у веброзробці як комплексний адаптивний механізм, що впливає на всі рівні життєвого циклу вебпродукту.

4.1.2 Еволюція інтеграції ШІ у вебтехнології

Розвиток ШІ у вебтехнологіях можна умовно поділити на чотири хронологічні етапи, кожен з яких характеризується власними досягненнями та методологічними підходами.

Початковий етап (1990-2000 рр.) характеризувався домінуванням статичних HTML-сторінок з мінімальною інтерактивністю. Вебтехнології цього періоду базувалися на простих протоколах передачі гіпертексту та не передбачали інтеграції інтелектуальних компонентів. Проте вже тоді з'являлися перші спроби використання алгоритмів для аналізу вебтрафіка та базової персоналізації контенту.

Етап динамічного веб (2000-2010 рр.) ознаменувався появою динамічних вебзастосунків та першими прикладами інтеграції алгоритмів ШІ. Ключовими досягненнями цього періоду стали впровадження рекомендаційних систем Amazon, які використовували колаборативну фільтрацію для пропозиції товарів, та розроблення алгоритму PageRank компанії Google, що застосувала графовий аналіз для ранжування вебсторінок. Такі рішення заклали теоретичні та практичні основи для подальшої інтеграції ШІ у вебсистеми.

Етап машинного навчання (2010-2017 рр.) характеризувався розвитком технологій глибокого навчання та появою доступних ML-фреймворків для розробників. Випуск TensorFlow у 2015 році робив машинне навчання доступним широкому колу розробників, зокрема для вебінтерфейсів. Період відзначився розвитком мобільних технологій та стимулював використання ШІ для автоматичної адаптації інтерфейсів.

Сучасний етап (2017 р. – дотепер) визначається формуванням парадигми AI-first веброзробки, впровадженням архітектури Transformer та великих мовних моделей [182]. Поширення генеративних технологій дозволило автоматизувати не лише створення контенту, а й програмного коду, що радикально змінило процеси веброзробки.

4.1.3 Технологічні компоненти сучасної веброзробки на основі ШІ

Сучасна інтеграція ШІ у веброзробку охоплює кілька ключових напрямів.

Обробка природної мови (NLP) становить один з ключових компонентів, що використовується для автоматичної генерації текстового контенту, розпізнавання голосових команд, створення діалогових інтерфейсів та семантичного аналізу користувацьких запитів. Теоретичною основою цього напряму є методи семантичного аналізу, моделі на основі рекурентних нейронних мереж та сучасні трансформерні архітектури.

[182] Dobrescu P., Flavia D. AI First. *The New Age of Development. Contributions to Political Science.* 2022. P. 1-43. DOI: https://doi.org/10.1007/978-3-031-05664-2_1

Генеративні моделі поєднують великі мовні моделі (LLM) та дифузійні моделі (Diffusion models), забезпечують автоматичне створення програмного коду, стилів CSS, графічних елементів та мультимедійного контенту. Вони базуються на теорії ймовірнісних моделей, зокрема на методах варіаційного автокодування та генеративно-змагальних мережах. Практичне застосування генеративних моделей у веброзробці стосується автоматичного створення адаптивних лейаутів, генерації кольорових схем та типографічних рішень.

Машинне навчання у вебконтексті реалізує персоналізацію інтерфейсів, аналіз користувацького досвіду та прогнозування поведінкових патернів. Методологічною основою є контрольоване та неконтрольоване навчання, ансамблеві підходи та методи глибокого навчання з підкріпленням. Практичні застосування стосуються динамічної адаптації навігаційних структур, оптимізації розміщення контенту та прогнозування конверсійних дій користувачів.

Комп'ютерний зір у веброзробці застосовується для аналізу інтерфейсних макетів, автоматичного тестування візуальних рішень та розпізнавання графічних патернів. Технологічною базою є згорткові нейронні мережі, методи семантичної сегментації та техніки передачі навчання. Практичне використання поєднує автоматичну генерацію альтернативних описів для зображень, оптимізацію візуальної ієрархії та забезпечення доступності інтерфейсів для користувачів з особливими потребами.

Інструменти веброзробки з підтримкою ШІ, як-от: GitHub Copilot, Amazon Q Developer, автоматизують процеси програмування, тестування та рефакторингу коду. Теоретичною основою цих систем є теорія формальних мов, методи трансляції програм і моделі трансформації між поданням коду та природною мовою.

Ці компоненти формують основу сучасної інтелектуальної екосистеми веброзробки.

4.1.4 Великі мовні моделі як основа екосистеми ШІ

Великі мовні моделі (LLM), серед яких GPT, Claude, Gemini тощо, є ключовим компонентом сучасної екосистеми ШІ[183]. Теоретичною основою функціонування цих систем є архітектура трансформерів, яка використовує механізми attention для ефективного оброблення послідовностей змінної довжини[184]. На відміну від попередніх підходів, які базувалися на рекурентних нейронних мережах, трансформерна архітектура забезпечує можливість паралельного оброблення та ефективного моделювання довготривалих залежностей у тексті.

Попереднє навчання на великих текстових даних дає змогу великим мовним моделям засвоювати широкий спектр мовних патернів та знань про світ, що робить їх універсальними інструментами для різноманітних задач веброзробки.

---

[183] Brown T. et al. Language models are few-shot learners. *Advances in neural information processing systems*. 2020. Vol. 33, P. 1877–1901. https://doi.org/10.48550/arXiv.2005.14165

[184] Breckner K., Neumayr T., Mara M., Streit M, Augstein M. The Changing Nature of Human-AI Relations: A Scoping Review on Terminology and Evolvement in the Scientific Literature. *International Journal of Human–Computer Interaction*. 2025. P.1–58. DOI: https://doi.org/10.1080/10447318.2025.2482742

Процес тонкого налаштування (fine-tuning)[185] адаптує мовні патерни до специфічних задач, наприклад, генерації HTML-розмітки, CSS-стилів або JavaScript-коду. Розвиток цих технологій сприяє формуванню парадигми «no-code/low-code» розроблення, яка суттєво знижує поріг входу у веброзробку для нетехнічних спеціалістів.

Водночас поширюються спеціалізовані фреймворки ШІ для веброзробки, серед яких: TensorFlow.js – для машинного навчання у браузері, Hugging Face Transformers.js – для NLP-завдань та розширені версії графічних бібліотек з інтегрованими ШІ-компонентами.

Згідно з дослідженням McKinsey Global Institute[186] 65% організацій регулярно використовують генеративний ШІ, що майже вдвічі більше, порівняно з показниками десятимісячної давності. Це свідчить про прискорене впровадження технологій генеративного ШІ в корпоративному секторі, де довгострокова економічна оцінка потенціалу ШІ складає 4,4 трильйона доларів США додаткового зростання продуктивності на глобальному рівні[187].

Дослідження Stack Overflow Developer Survey[188] показує, що 84% розробників використовують інструменти ШІ у своєму робочому процесі. Цікаво, що позитивне ставлення до інструментів ШІ у 2025 році знизилося: 70%+ у 2023 та 2024 роках до лише 60% цього року. Професіонали мають вищий загальний позитивний настрій (61%), ніж ті, хто вчиться програмувати (53%).

4.1.5 Функціональні цілі застосування ШІ у вебтехнологіях

Використання ШІ у веброзробці та вебдизайні спрямовано на розв'язання таких взаємопов'язаних завдань, які охоплюють весь життєвий цикл вебпроєктів:

– *автоматизація створення та підтримки вебзастосунків* охоплює процеси від початкової генерації архітектурних рішень до автоматичного тестування функціональності. Методологічною основою цього напряму є принципи програмування, що дозволяють створювати програми, які генерують інші програми, та методи автоматичної генерації тестових сценаріїв на основі формальних специфікацій;

– *оптимізація користувацького досвіду* реалізується через впровадження інтелектуальних систем A/B тестування, аналіз теплових карт взаємодії та конверсійну оптимізацію на основі машинного навчання. Теоретичним підґрунтям є статистичні методи експериментального дизайну, теорія ухвалення рішень та байєсівські підходи до оптимізації, які дозволяють системам автоматично визначати найефективніші варіанти інтерфейсних рішень;

---

[185] Кудрик О.В., Бісікало О.В., Здітовецький Ю.С. Методи тонкого налаштування штучного інтелекту. *Вісник Вінницького політехнічного інституту*. 2024. № 4. С. 139–146. DOI: https://doi.org/10.31649/1997-9266-2024-175-4-139-146

[186] Quantum Black AI by McKinsey. The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024

[187] McKinsey & Company. Superagency in the workplace: Empowering people to unlock AI's full potential. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ais-full-potential-at-work

[188] Stack Overflow Developer Survey 2025. URL https://survey.stackoverflow.co/2025/

– *інтелектуалізація інтерфейсу* стосується створення адаптивних навігаційних структур, інтелектуальних систем пошуку та голосових інтерфейсів управління. Методологічною базою є теорія інформаційного пошуку, методи оброблення природної мови та технології розпізнавання мовлення. Практична реалізація може стосуватися створення чат-ботів для технічної підтримки, голосових асистентів для навігації сайтом та систем автоматичного доповнення пошукових запитів;

– *персоналізація функціональності* здійснюється на основі глибокого аналізу поведінкових патернів користувачів з використанням рекомендаційних систем та методів кластеризації. Теоретичною основою є теорія колаборативної фільтрації, методи контент-базованих рекомендацій та гібридні підходи, які поєднують різні стратегії персоналізації для досягнення максимальної ефективності;

– *прогнозування дій користувача та аналітика* базуються на методах часових рядів, марківських моделей і сучасних підходах predictive UX. Ці технології дозволяють системам передбачати потенційні дії користувача й оптимізувати інтерфейс для покращення конверсії та задоволеності користувачів.

Практична реалізація функціональності ШІ у вебсистемах здійснюється через використання як готових фреймворків, так і спеціалізованих хмарних сервісів. TensorFlow.js забезпечує можливість виконання моделей машинного навчання безпосередньо у браузері користувача, що дозволяє створювати інтерактивні застосунки без потреби серверного оброблення. Microsoft Azure AI та Google Cloud AI Platform пропонують комплексні рішення для інтеграції різноманітних сервісів ШІ у вебархітектури.

Інтеграція зовнішніх API, включно з OpenAI, Anthropic Claude, Google Vertex AI та IBM Watson, дозволяє розробникам використовувати потужні мовні моделі без потреби їх локального розгортання. Така парадигма «ШІ як сервіс»[189] значно спрощує впровадження складних інтелектуальних функцій у вебзастосунки та знижує технічні бар'єри для малих і середніх команд розробників.

У контексті вебдизайну ШІ забезпечує скорочення часу проєктування на 40-60% та суттєве зменшення кількості помилок користувацького досвіду[190]. Інструменти автоматизованого дизайну, як-от: Uizard, Figma AI або Wix ADI, створюють адаптивні макети на основі аналізу кращих практик інтерфейсного дизайну та специфічних брендингових вимог клієнтів[191].

Сучасні інструменти ШІ для вебдизайну здатні забезпечити автоматичну генерацію кольорових схем на основі психології кольору та теорії контрасту, створення типографічних рішень з урахуванням читабельності й естетичних принципів, формування компонувальних сіток відповідно до принципів візуальної ієрархії, а також автоматичну генерацію графічних елементів згідно з фірмовим стилем.

---

[189] Syed N. et al. Artificial Intelligence as a Service (AIaaS) for Cloud, Fog and the Edge: State-of-the-Art Practices. *ACM Computing Surveys*. 2025. Vol. 57. № 8. P. 1-36. DOI: https://doi.org/10.1145/3712016

[190] Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv*. 2023. Vol. 2302.06590. DOI: https://doi.org/10.48550/arXiv.2302.06590

[191] Shakked N., Whitney Z. Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. 2023. DOI: http://dx.doi.org/10.2139/ssrn.4375283

Технології генерації зображень (Adobe Firefly, Midjourney та DALL-E) поступово інтегруються у робочі процеси дизайнерів для створення унікального візуального контенту. Ці системи базуються на дифузійних моделях і технологіях text-to-image генерації, що дозволяє створювати високоякісні візуальні матеріали на основі текстових описів дизайнерських концепцій.

Загалом теоретичні засади використання ШІ у вебтехнологіях формують комплексну методологічну основу для розроблення інтелектуальних, адаптивних високоефективних вебрішень нового покоління. Інтеграція ШІ у вебдизайн не обмежується простою автоматизацією наявних процесів, а передбачає фундаментальну трансформацію підходів до створення цифрових продуктів, орієнтованих на глибоке розуміння потреб користувачів та динамічну адаптацію до змінюваного контексту їх використання.

4.2 Архітектура впровадження ШІ в сучасні вебсистеми

Ефективна інтеграція ШІ у вебсистеми потребує структурованого архітектурного підходу, що виходить за межі традиційних багаторівневих моделей. Архітектурна організація систем визначає не лише масштабованість, надійність та адаптивність технологічних рішень, а й ефективне оброблення великих обсягів даних, управління життєвим циклом моделей машинного навчання та інтеграцію із зовнішніми сервісами ШІ.

Практика провідних компаній підтверджує, що успішне впровадження ШІ можливе лише за умови тісної інтеграції інтелектуальних модулів у загальну архітектуру платформи. Компанія Google використовує ШІ у пошукових алгоритмах (BERT, MUM) для семантичного розуміння запитів та контексту, а також пропонує розробникам інструменти Google Cloud AI та Vertex AI, які дозволяють підключати ML-моделі безпосередньо до вебзастосунків[192]. У Google Ads застосовуються генеративні технології для автоматичного створення рекламних текстів і підбору зображень. У Figma тестуються функції автодизайну макетів на основі текстових описів.

Компанія реалізує ШІ через хмарні сервіси Amazon Personalize, які надають рекомендаційні алгоритми (колаборативна фільтрація, deep learning) для платформ електронної комерції. Інструмент Amazon Q Developer виконує автогенерацію коду на основі великих мовних моделей[193], а AWS AI Services (Comprehend, Rekognition) дає змогу інтегрувати NLP і комп'ютерний зір у вебсистеми без потреби у власній ML-команді[194].

Shopify робить акцент на персоналізації сторінок магазинів і прогнозуванні попиту. Shopify Magic генерує описи товарів на основі коротких підказок від продавців. Алгоритми динамічного ціноутворення та рекомендацій інтегровані безпосередньо в Shopify API, що дозволяє навіть невеликим бізнесам впроваджувати інтелектуальні рішення без значних інвестицій[195].

---

[192] Combine elements from different images, and other new ways to bring your creativity to life. URL: https://ai.google/
[193] Amazon Q Developer. URL: https://aws.amazon.com/ru/q/developer/
[194] Amazon Augmented AI Resources. URL: https://aws.amazon.com/augmented-ai/resources/
[195] Shopify API, libraries, and tools. URL: https://shopify.dev/docs/api

Ці приклади показують, що ключовим є не лише застосування алгоритмів, а й побудова цілісної архітектури, здатної ефективно інтегрувати компоненти ШІ на різних рівнях системи. Концептуальною основою таких архітектур є поєднання принципів розподілених систем, мікросервісної організації та конвеєрного оброблення даних[196]. На відміну від класичних вебрішень, системи на основі ШІ потребують додаткових рівнів для роботи з ML-моделями, управління їхнім життєвим циклом та забезпечення інтеграції із зовнішніми API. Важливим є також застосування подієво-орієнтованої (event-driven) архітектури, де взаємодія між компонентами здійснюється асинхронно через повідомлення та події [197]. Це дає змогу ефективно виконувати ресурсомісткі операції ШІ (наприклад, навчання моделей чи генерацію великих обсягів контенту) без блокування основної функціональності вебсистем.

4.2.1 П’ятирівнева архітектурна модель

Сучасна архітектура інтеграції ШІ у вебсистеми часто структурується як п’ятирівнева модель[198]. Кожен рівень виконує специфічні функції та має чітко визначені інтерфейси взаємодії із суміжними компонентами[199].

1. Рівень презентації та користувацької взаємодії (Presentation Layer)

Рівень презентації виходить за межі традиційного відображення статичного контенту та має інтерактивні елементи, які адаптуються у реальному часі до поведінки користувача. Архітектурно цей рівень реалізується через «progressive web applications» з використанням сучасних JavaScript-фреймворків, таких як React, Vue.js або Angular, доповнених спеціалізованими бібліотеками для роботи з ML.

Важливою особливістю є впровадження «client-side» ШІ – виконання інференсу моделей безпосередньо у браузері за допомогою TensorFlow.js або ONNX.js, що забезпечує миттєву реакцію інтерфейсу без серверних викликів. Рівень презентації також має адаптивні діалогові інтерфейси на базі WebRTC, голосову взаємодію через WebSpeech API та візуалізацію результатів роботи ШІ через WebGL. Для синхронізації стану між компонентами застосовуються бібліотеки керування станом (Redux та інші).

2. Рівень прикладної логіки (Application Layer)

Рівень прикладної логіки обробляє бізнес-процеси системи та координує виклики до компонентів ШІ. Реалізує сервіси генерації контенту, персоналізації досвіду користувачів, формування рекомендацій та аналізу поведінкових патернів. Архітектурними рішеннями є мікросервісна організація, RESTful API та

---

[196] Котенко М.М., Вакалюк Т.А. Впровадження мікросервісної архітектури: технічні виклики та рішення. *Вісник Херсонського національного технічного університету*. 2024. №. 4 (91). С. 299-305. DOI: 10.35546/kntu2078-4481.2024.4.39

[197] Волокита А., Зоткін К. Проблематика проєктування програмних систем на основі подійно-орієнтованих архітектур (EDA). *Measuring and computing devices in technological processes*. 2024. Vol. 4. P. 130–136. DOI: 10.31891/2219-9365-2024-80-16.

[198] Layers in software architecture. URL: https://medium.com/@sagar.hudge/layers-in-software-architecture-c8cc16329ff6

[199] Generative AI fundamentals: Exploring the 6-Layer architecture. URL: https://www.epicalgroup.com/blogs/generative-ai-fundamentals-exploring-6-layer-architecture

gRPC протоколи. Функціональні компоненти мають контролери запитів, сервіси валідації даних, модулі аутентифікації й оркестратори для координації роботи інтелектуальних сервісів.

3. Рівень штучного інтелекту (AI Processing Layer)

Рівень ШІ має спеціалізовані компоненти для різних типів інтелектуальних операцій. Архітектурно цей рівень організований як набір незалежних мікросервісів, кожен з яких відповідає за специфічний тип функціональності ШІ. Рівень містить моделі машинного навчання, великі мовні моделі (LLM), генеративні системи (GAN, Diffusion), модулі оброблення природної мови (NLP), комп'ютерного зору (Computer Vision) та рекомендаційні системи. Розгортання компонентів здійснюється через спеціалізовані платформи: TensorFlow Serving, ONNX Runtime, Google Vertex AI. Рівень інкапсулює алгоритми ШІ та надає стандартизовані інтерфейси для їх використання іншими компонентами. У сучасних реалізаціях також застосовуються LangChain та RAG-агенти для розширеної генерації контенту.

4. Рівень управління даними (Data Management Layer)

Рівень управління даними забезпечує зберігання та оброблення інформації, потрібної для навчання та функціонування моделей: користувацькі сесії, логи активності, поведінкові патерни, мультимедійний контент. Технологічний стек охоплює NoSQL бази даних (MongoDB, Cassandra), реляційні СКБД (PostgreSQL, MySQL), хмарні сховища (Google BigQuery, Amazon S3) та архітектури Data Lakes. Особлива увага приділяється швидкому доступу до неструктурованих даних для інференсу моделей у реальному часі.

5. Рівень інтеграції та оркестрації (Integration and Orchestration Layer)

Рівень координує взаємодію між усіма компонентами системи та забезпечує надійну доставку даних і результатів оброблення. Ключовими платформами є Apache Kafka для потокового оброблення даних, Apache Airflow для оркестрації конвеєрів машинного навчання, Kubernetes для контейнеризації та масштабування.

4.2.2 Практична реалізація архітектури

Практична демонстрація архітектурних принципів проілюстровано на прикладі інтеграції великої мовної моделі у вебзастосунок для генерації персоналізованого контенту. Архітектурна схема на рис. 4.1 відображає потік даних від користувацького інтерфейсу через API-шлюз до моделей ШІ та зворотне повернення результатів через усі рівні системи.

Такий підхід забезпечує розділення відповідальності між компонентами й можливість незалежного масштабування. Практичне впровадження цієї архітектури показано на прикладі інтеграції GPT у вебзастосунок. Створений вебзастосунок має форму, через яку користувач вводить запит, що надсилається до GPT через API, далі відповідь зберігається у базі даних MongoDB, адміністратор отримує повідомлення через інтегрований сервіс автоматизації робочих процесів.

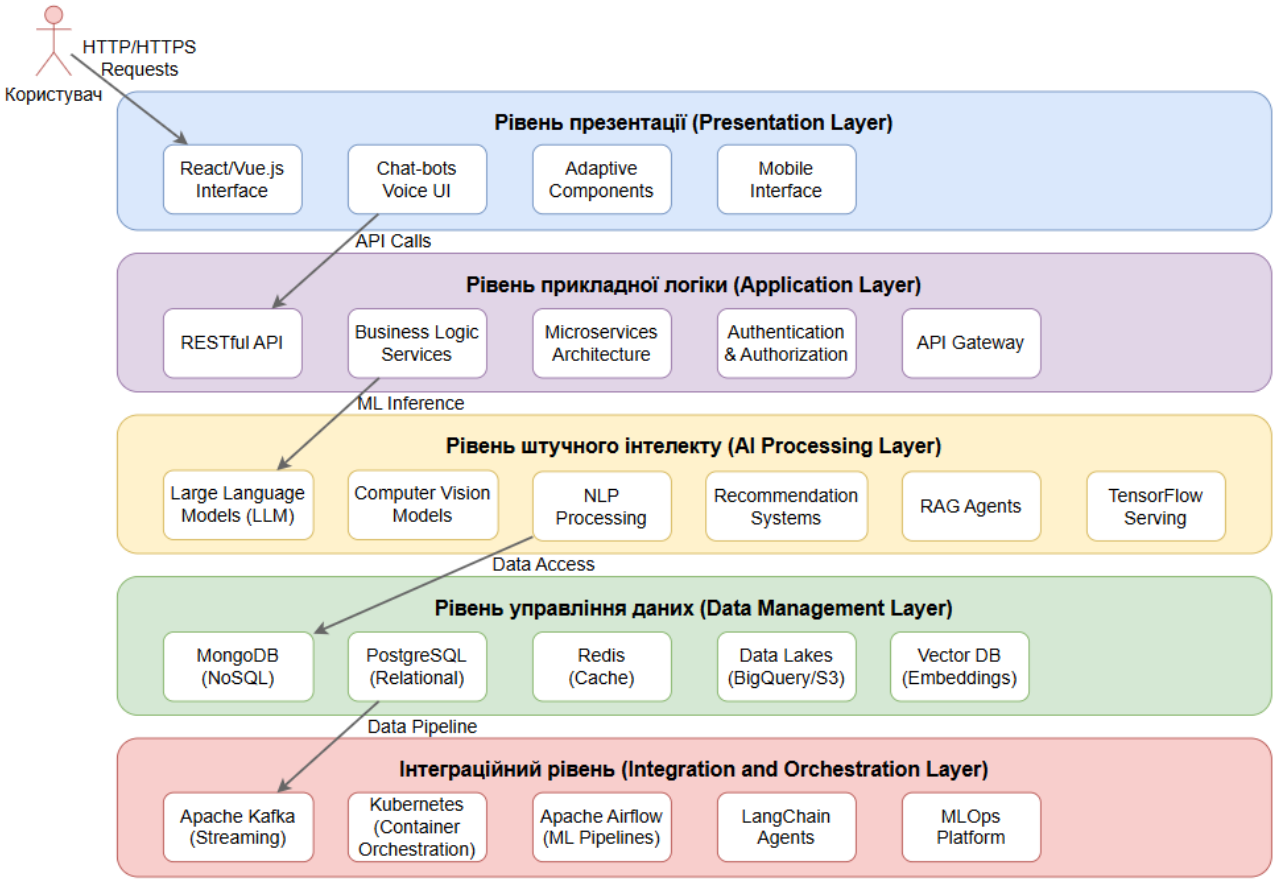


Рисунок 4.1. Архітектура схема впровадження ШІ у вебсистеми

Фрагмент запиту до OpenAI API (у форматі curl) показано на рис. 4.2.

```
curl https://api.openai.com/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gpt-4",
    "messages": [
      {"role": "user", "content": "Що таке штучний інтелект?"}
    ]
  }'
```

Рисунок 4.2. Фрагмент запиту до OpenAI API

Відповідь надходить у форматі JSON, де поле content містить згенеровану відповідь моделі (рис. 4.3).

```
{
  _id: ObjectId('68a0f0890acf66720dbe2566'),
  id: 'chatcmpl-9xSAMPLEgr9iU0B5sR2282',
  object: 'chat.completion',
  created: 1723841112,
  model: 'gpt-4-0613',
  choices: [
    {
      index: 0,
      message: {
        role: 'assistant',
        content: "Штучний інтелект (ШІ) — це галузь комп'ютерних наук,
      },
      finish_reason: 'stop'
    }
  ],
  usage: {
    prompt_tokens: 15,
    completion_tokens: 85,
    total_tokens: 100
  }
}
```


Рисунок 4.3. Згенерована відповідь моделі у форматі JSON

Як видно з прикладу, ключова інформація міститься в об’єкті message з роллю assistant. Саме цей згенерований текст (content) далі обробляється на рівні прикладної логіки, зберігається в базі даних MongoDB і повертається користувачеві на рівні презентації.

4.2.3 Забезпечення якості та надійності архітектур ШІ

Інтеграція ШІ у вебсистеми вимагає не лише коректної архітектури, а й гарантованої якості та стабільності роботи. Для цього застосовуються низка перевірених підходів. Одним із ключових елементів є *моніторинг якості* роботи компонентів ШІ, де основна увага приділяється швидкості відгуку та стабільності серверів, у випадку з ШІ важливо також оцінювати точність і релевантність результатів моделей. Якщо система рекомендацій починає пропонувати нерелевантний контент або чат-бот генерувати помилкові відповіді, це впливає на якість користувацького досвіду. Тому сучасні платформи моніторингу інтегрують механізми відстеження продуктивності моделей у середовищі production, що дозволяє оперативно реагувати на відхилення та підтримувати стабільність системи[200].

По-друге, критичну роль у безпечному впровадженні нових рішень відіграє *інфраструктура тестування та експериментального аналізу*. A/B-тестування застосовується не лише до елементів візуального дизайну, а й до поведінкових характеристик модулів ШІ. Система може поступово апробувати нові алгоритми персоналізації або інтерфейсні підказки на обмеженій частині аудиторії,

[200] Nigenda D. et al. Amazon SageMaker Model Monitor: A System for Real-Time Insights into Deployed Machine Learning Models. *arXiv*. 2022. DOI: https://doi.org/10.48550/arXiv.2111.13657

що дозволяє уникати різких змін, здатних негативно вплинути на користувацький досвід.

Третій важливий аспект – *безпека та прозорість*. Вебсистеми з інтегрованими компонентами ШІ обробляють значні обсяги персональних даних, тому принципи privacy by design та етичні стандарти мають бути закладені в архітектуру ще на етапі проєктування, що передбачає мінімізацію збору інформації, чітке інформування користувачів про мету та способи її використання, а також впровадження механізмів пояснюваності роботи алгоритмів, зокрема, у вигляді доступних і зрозумілих візуалізацій.

Не менш важливими є *масштабованість і продуктивність*. Інтеграція ШІ часто супроводжується зростанням навантаження на систему, зокрема, через генерацію контенту та оброблення великих масивів даних у реальному часі. Для забезпечення стабільної роботи вебплатформи застосовуються технології контейнеризації та хмарні обчислення, що дозволяють оперативно адаптуватися до пікових навантажень.

Нарешті, *стійкість і відмовостійкість* є фундаментальними характеристиками архітектури[201]. Вебсистема з інтегрованими модулями ШІ повинна залишатися функціональною навіть у разі виникнення помилок. Для цього застосовуються перевірені архітектурні рішення, зокрема концепція graceful degradation, яка стосується збереження базової функціональності при тимчасовому відключенні складних інтелектуальних компонентів, а також механізми автоматичного відновлення після збоїв.

Отже, архітектура впровадження ШІ у вебсистеми має комплексне технологічне рішення, яке інтегрує традиційні вебтехнології зі спеціалізованими компонентами ШІ. П’ятирівнева архітектурна модель забезпечує розділення відповідальності між компонентами, масштабованість системи та можливість незалежного розвитку різних функціональних блоків. Практична реалізація таких архітектур вимагає глибокого розуміння як традиційних принципів побудови розподілених систем, так і специфічних вимог роботи з ML-компонентами.

4.3 Веброзробка з інструментами ШІ: від дизайну до оптимізації

Інтеграція ШІ в процес UX/UI-дизайну сприяє переходу від статичних рішень до динамічних, персоналізованих систем, здатних адаптуватися в реальному часі відповідно до контексту використання та поведінкових моделей користувача. Теоретичне підґрунтя цього підходу становить синтез принципів людино-орієнтованого дизайну (Human-Centered Design) з алгоритмічними методами ШІ[202]. UX/UI-дизайн з підтримкою ШІ розглядається як неперервний процес, в якому інтелектуальні системи беруть участь на всіх етапах життєвого циклу: від аналітичного дослідження та генерації рішень до експериментального тестування, метрик ефективності та неперервної оптимізації.

---

[201] Vaibhav J, Simic M. Microservices Circuit Breaker Pattern. *Baeldung*. 2023. URL: https://www.baeldung.com/cs/microservices-circuit-breaker-pattern

[202] Zhang Yu., Chen J., Liu H., Chen Y., Xiao B., Li H. Recent advancements of human-centered design in building engineering: A comprehensive review. *Journal of Building Engineering*. 2024. Vol. 84. DOI: https://doi.org/10.1016/j.jobe.2024.108529

У цьому контексті ключовими концептами виступають: персоналізація в реальному часі, адаптивний інтерфейс, генеративний дизайн та експериментування як механізм доказової оптимізації.

4.3.1 Методологічна основа сучасного UX/UI-дизайну з підтримкою ШІ

Методологічна основа сучасного UX/UI-дизайну з підтримкою ШІ базується на інтеграції кількох міждисциплінарних підходів:

– *когнітивна психологія* надає розуміння процесів сприйняття інформації користувачем, що дозволяє алгоритмам ШІ оптимізувати візуальну інформаційну архітектуру інтерфейсу;

– *теорія інформації* визначає принципи організації контенту та навігаційних структур для оптимальної подачі інформації;

– *дизайн-мислення* адаптується для роботи зі ШІ через впровадження data-driven підходів до ітераційного проєктування. Традиційні етапи дизайн-мислення (емпатія, визначення проблеми, ідея, прототипування, тестування) доповнюються алгоритмічним аналізом користувацьких даних, автоматичною генерацією дизайн-варіантів та інтелектуальним A/B-тестуванням;

– *теорія складних адаптивних систем* дозволяє моделювати інтерфейс як динамічну систему зворотного зв'язку, що змінюється у відповідь на поведінку користувача.

Практичний процес утворює замкнений цикл безперервної оптимізації[203]. Такий підхід інтегрує збір даних, аналітику, експерименти та впровадження в єдиний автоматизований процес.

Розглянемо детально кожен з п'яти ключових етапів:

1. Збір даних. На цьому етапі збираються дані про взаємодію користувачів, наприклад: кліки, глибина скролінгу, теплові карти (heat maps) та записи сесій. Для цього використовуються інструменти аналітики, Microsoft Clarity (https://clarity.microsoft.com/), Hotjar (https://www.hotjar.com/) та Google Analytics 4 (GA4). Сучасні системи збору даних забезпечують не лише кількісні показники, а й якісні інсайти щодо поведінкових патернів користувачів.

2. Аналіз з використанням ШІ. Зібрані дані аналізуються за допомогою алгоритмів машинного навчання для виявлення закономірностей та проблем. Основні методи забезпечують кластеризацію (з використанням K-means або DBSCAN для сегментації поведінки)[204], виявлення аномалій та предиктивний аналіз (за допомогою моделей Long Short-Term Memory (LSTM) для прогнозування дій), що дозволяє класифікувати сесії як успішні чи проблемні.

3. Рекомендації та редизайн. На основі аналізу інструменти ШІ пропонують конкретні зміни в дизайні. Рекомендації можуть стосуватися покращення

---

[203] Лобода Ю.Г., Трофименко О.Г., Манаков С.Ю., Гура В.І. Штучний інтелект в сучасній веброзробці та вебдизайні: багаторівнева класифікація та систематизація. *Informatics and Mathematical Methods in Simulation*. 2025. № 15(1). С. 126-136. DOI: http://immm.op.edu.ua/files/archive/n1_v15_2025/2025_1(12).pdf

[204] Bansal M., Goyal A., Choudhary A. A comparative analysis of K-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning. *Decision analytics journal*. 2022. Vol. 3. DOI: https://doi.org/10.1016/j.dajour.2022.100071

лейаутів, контрастності, закликів до дії (calls to action, CTA) та інших елементів. Інструменти Figma AI (https://www.figma.com/ai/) та Adobe Firefly (https://firefly.adobe.com) можуть автоматично генерувати кілька варіантів оновленого дизайну на основі виявлених проблемних зон.

4. A/B тестування[205]. Згенеровані ШІ варіанти дизайну проходять автоматизоване тестування. За допомогою платформ, таких як Optimizely (https://www.optimizely.com) або Dynamic Yield (https://www.dynamicyield.com/), порівнюються ключові метрики (наприклад, конверсія CTR (Click-Through Rate)), щоб на основі даних зробити вибір найефективнішої версії інтерфейсу.

5. Оптимізація. Цей етап замикає цикл, забезпечуючи неперервне вдосконалення. Він поєднує постійне навчання (continual learning)[206] моделей на нових даних, а також поступове та контрольоване впровадження змін (roll-out). Для динамічної оптимізації в реальному часі можуть застосовуватися алгоритми, наприклад, багаторукі бандити (Bandits)[207], які автоматично спрямовують більше користувачів на найефективніший варіант.

На рис. 4.4 візуалізовано послідовність етапів від збору даних до оптимізації, які утворюють замкнений цикл «дані → аналіз → рекомендації → експерименти → оновлення».

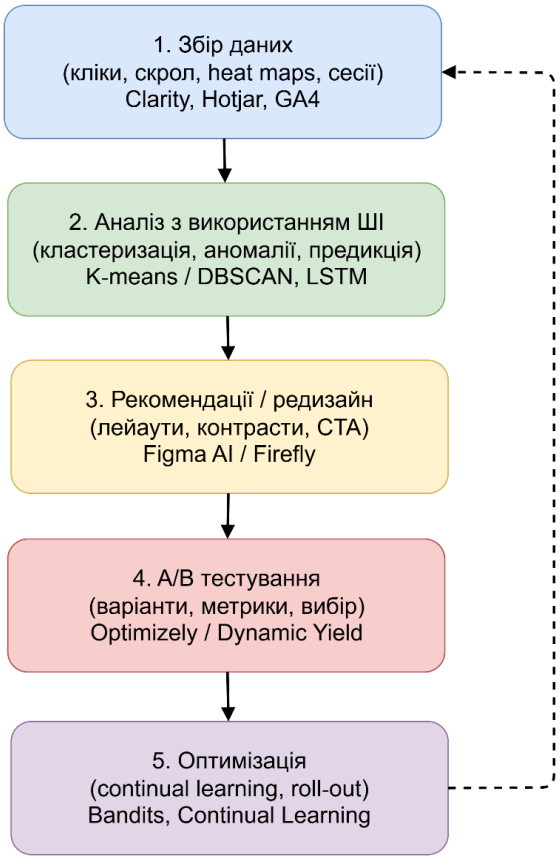


Рисунок 4.4. Схема UX-процесу з підтримкою ШІ

[205] Quin F., Weyns D., Galster M., Silva C. A/B testing: A systematic literature review. *Journal of Systems and Software*. 2024. Vol. 211. DOI: https://doi.org/10.1016/j.jss.2024.112011

[206] Wang L., Zhang X., Su H., Zhu J. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*. 2024. Vol. 46(8). P. 5362-5383. DOI: https://doi.org/10.1109/TPAMI.2024.3367329

[207] Niño-Mora J. Markovian Restless Bandits and Index Policies: A Review. *Mathematics.* 2023. Vol. 11. DOI: https://doi.org/10.3390/math11071639

Як результат формується замкнений цикл постійного вдосконалення інтерфейсу, де користувацька взаємодія стає основою для неперервної оптимізації, що забезпечує адаптивність системи до змінюваних потреб та очікувань користувачів.

4.3.2 Генерація текстового контенту

Сучасні генеративні моделі забезпечують автоматизацію процесів створення текстів для блогів, каталогів і маркетингових матеріалів та підтримують глибоку адаптацію під наміри користувача й вимоги пошукових систем[208].

Процес генерації текстового контенту поєднує три взаємопов'язані компоненти:

1) *препроцесинг та контекстуалізація* – система аналізує вхідні параметри: тип контенту (блог-пост, товарний опис, метаопис), цільову аудиторію, ключові слова та тон комунікації. На цьому етапі формується контекстуальний промпт, який направляє модель на генерацію релевантного контенту;

2) *процес генерації* – основою є моделі GPT та Claude або спеціалізовані рішення Jasper AI та Copy.ai, ці системи генерують текст поетапно, використовуючи ймовірнісний підхід до вибору наступного слова на основі контексту й навченості моделі;

3) *постпроцесинг та оптимізація* – згенерований контент проходить кілька етапів оброблення: SEO-оптимізація (вставка ключових слів, LSI-термінів, оптимізація щільності); стилістична корекція (приведення до брендового голосу та його тону); фактчекінг (перевірка достовірності фактичної інформації); читабельність (оптимізація складності речень та структури тексту).

На рис. 4.5 показано схему генерації контенту на основі ШІ.

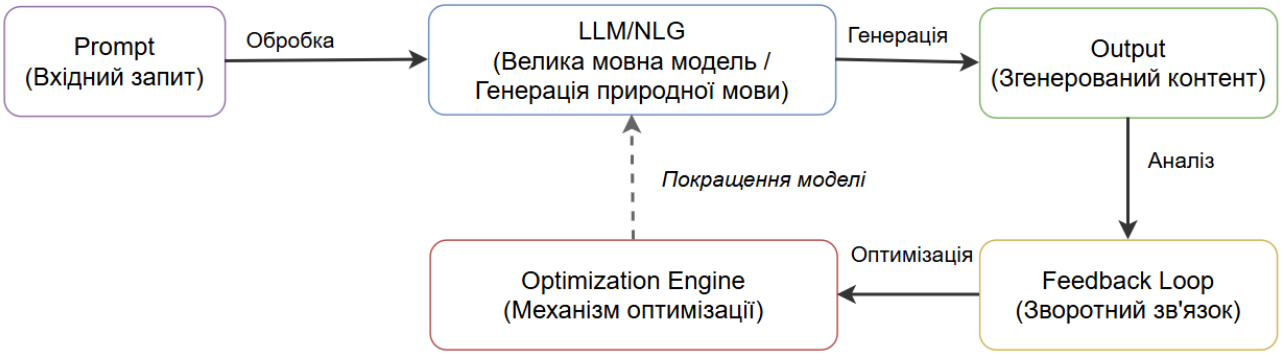


Рисунок 4.5. Схема генерації контенту на основі ШІ

Наведена схема ілюструє цикл генерації контенту через великі мовні моделі (LLM/NLG), який поєднує етапи оброблення вхідного запиту (prompt), створення контенту, аналізу результату через зворотний зв'язок (петля зворотного

[208] Floridi L. Artificial Intelligence, Deepfakes and a Future of Ectypes. *Ethics, Governance, and Policies in Artificial Intelligence*. *Philosophical Studies Series.* 2021. Vol 144. P. 307–312. DOI: https://doi.org/10.1007/978-3-030-81907-1_17

зв'язку) та подальшої оптимізації моделі. Механізм оптимізації (Optimization Engine) забезпечує покращення якості генерованого контенту на основі отриманих метрик і користувацьких відгуків.

Системи генерації дозволяють створювати великі обсяги контенту для блогів, соціальних мереж та email-кампаній. Наприклад, для інтернет-магазину можна автоматично генерувати унікальні описи товарів на основі технічних характеристик та категорій (рис. 4.6).

```
def generate_product_description(product_data):
    prompt = f"Опис для {product_data['name']}: {product_data['features']}"
    return llm_model.generate(prompt)
```

Рисунок 4.6. Автоматичне генерування описів товарів

ШІ дозволяє створювати персоналізовані версії контенту на основі поведінки користувача, демографічних даних та історії взаємодії із сайтом. Контент може стосуватися адаптації заголовків, описів продуктів, елементів із закликом до дії. Сучасні моделі здатні не лише перекладати контент, а й адаптувати його до культурних особливостей різних ринків, враховуючи локальні традиції, звички та мовні нюанси.

4.3.3 Мультимедійна генерація

Мультимедійна генерація є одним із напрямів розвитку ШІ у сфері веброзробки. Вона забезпечує створення візуального контенту (зображень, відео, інтерфейсів користувача) за текстовим описом або іншими вхідними даними. Генеративні моделі, засновані на дифузійних алгоритмах і трансформерах, забезпечують проєктування, в якій розробник або дизайнер виступає радше куратором, ніж виконавцем.

Генерація зображень. Системи DALL·E 3 (OpenAI), Midjourney, Adobe Firefly та Stable Diffusion дозволяють створювати ілюстрації високої якості, які адаптуються до стилю бренду, кольорової гами, розміру або контексту використання[209].

Приклади промптів практичного застосування генерації зображень:

– *створення тематичних банерів:* «створи банер для головної сторінки технологічної компанії у стилі мінімалізм з акцентом на синьо-білу гаму»;

– *іконки інтерфейсу:* «згенеруй іконку для мобільного застосунку на тему подорожей у стилі flat design з елементами літака та карти»;

– *ілюстрації для контенту:* «згенеруй зображення для блогу на тему сталого розвитку з використанням природних мотивів у спокійній кольоровій палітрі».

На рис. 4.7 наведено приклад API-запиту для генерації зображення.

[209] Fiala S. Comparison of Top AI Image Models: DALL·E 3, Midjourney, Stable Diffusion, Adobe Firefly, Grok, and Ideogram. URL: https://www.linkedin.com/pulse/comparison-top-ai-image-models-dalle-3-midjourney-stable-sabra-fiala-xdxvf/

```python
def generate_banner_image(description, style="modern"):
    prompt = f"Create banner: {description}, style: {style}"
    response = openai.Image.create(prompt=prompt, n=1, size="1920x1080")
    return response['data'][0]['url']
```

Рисунок 4.7. API-запит для генерації зображення

Генерація відео та анімацій. Інструменти Pika Labs (https://pika.art/), RunwayML (https://runwayml.com/), Sora (https://sora.com/) та Synthesia (https://www.synthesia.io/) дозволяють створювати короткі відео «з нуля» або на основі наданого тексту[210]. Платформи використовують комбінацію генеративних моделей та техніки синтезу руху для створення плавних анімацій.

Приклади застосування у веброзробці:

– *анімовані презентації продуктів* – створення коротких роликів, які демонструють функціональність без потреби фізичної зйомки;

– *персоналізовані рекламні ролики* – генерація відео-контенту, адаптованого під конкретні сегменти аудиторії;

– *адаптивний відеоконтент* – створення різних версій відео для різних пристроїв та контекстів використання.

Генерація інтерфейсів користувача. Galileo AI, Uizard, Framer AI та GitHub Copilot – UI-генератори, які трансформують ескізи, голосові або текстові запити на повноцінні UI-компоненти[211]. Системи використовують поєднання комп'ютерного зору та мовних моделей для розуміння дизайнерських інтентів[212]. Наприклад, запит «створи лендінг для онлайн-курсу з Python» породжує адаптивну верстку із секціями: опис курсу зі структурованою програмою навчання, інформація про викладачів з фотографіями та біографіями, форма запису з інтегрованою валідацією, відгуки студентів з рейтинговою системою.

На рис. 4.8 показано практичний приклад динамічного лендингу, адаптованого під користувача мобільного пристрою з Києва, який шукає курси Python, демонструючи персоналізацію контенту за географічними, технічними та поведінковими параметрами з урахуванням локальної мови, валюти та культурних особливостей.

HTML/CSS код-генерація. Активно використовується генерація HTML/CSS-коду на основі макетів або описів, що дозволяє швидко створювати прототипи та реалізовувати дизайн без глибоких знань front-end технологій.

На рис. 4.9 показано приклад генерації базового шаблону вебсторінки за запитом: «створи односторінковий сайт-фотогалерею з трьома зображеннями».

---

[210] Emerenini C. Top 10 AI Video Generation Tools for Creative Content. URL: https://techtraversex.hashnode.dev/top-ai-video-generation-tools-for-creative-content

[211] 5 нейромереж для створення UI та UX дизайну. URL: https://www.site2b.ua/ua/web-blog-ua/5-nejromerezh-dlya-stvorennya-ui-ta-ux-dizajnu.html

[212] 20+ найкращих AI-інструментів 2024 для дизайнера. URL: https://www.komarov.design/20-naikrashchikh-ai-instrumientiv-2024/

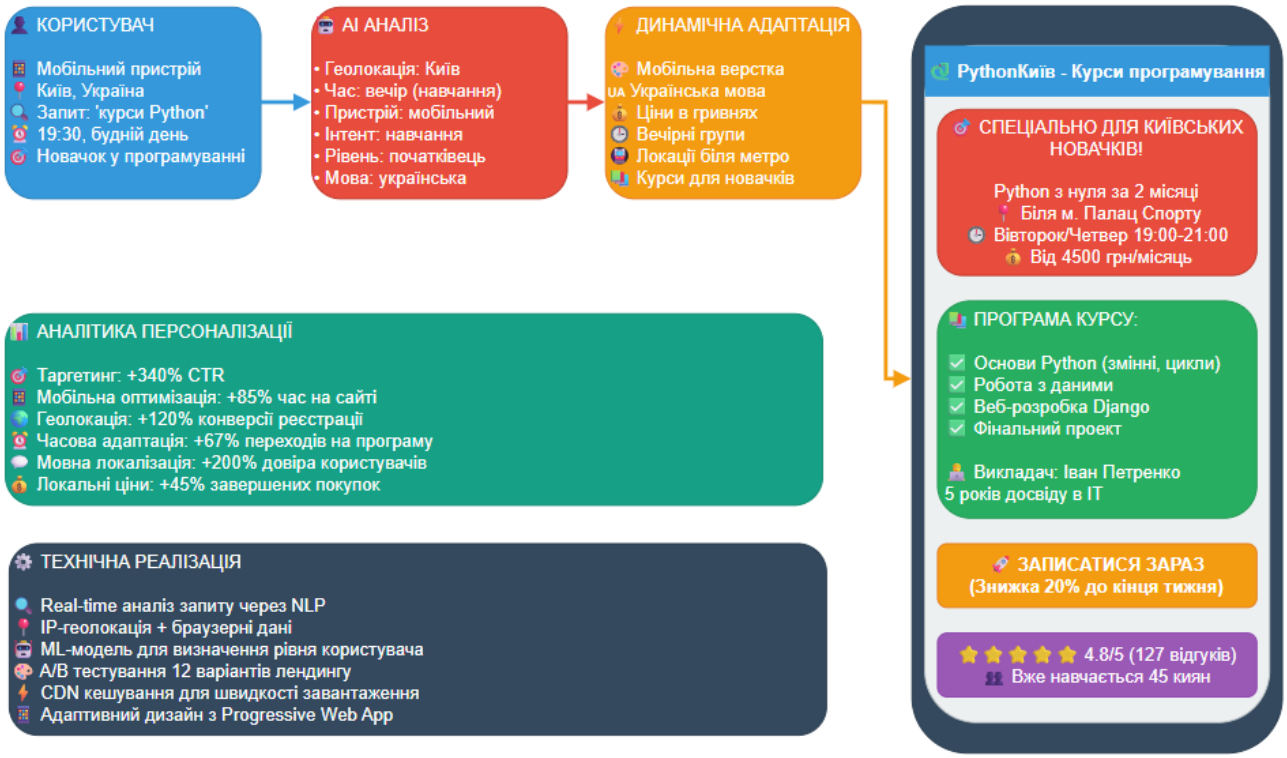


Рисунок 4.8. Приклад лендингу

```
<!-- Згенеровано за запитом: "створи фото-галерею з трьома зображеннями" -->
<!DOCTYPE html>
<html>
<head>
    <title>Моя Галерея</title>
    <style>
        body {
            font-family: sans-serif; text-align: center;
            background: #f4f4f4; margin: 0; padding: 20px;
        }
        .gallery {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
            gap: 15px; max-width: 800px; margin: 0 auto;
        }
        img {
            width: 100%; height: 200px; object-fit: cover;
            border-radius: 8px; box-shadow: 0 4px 8px rgba(0,0,0,0.1);
            transition: transform 0.3s;
        }
        img:hover { transform: scale(1.05); }
        h1 { color: #333; margin-bottom: 30px; }
    </style>
</head>
<body>
    <h1>🎨 Моя Галерея</h1>
    <div class="gallery">
        <img src="img1.jpg" alt="Фото 1">
        <img src="img2.jpg" alt="Фото 2">
        <img src="img3.jpg" alt="Фото 3">
    </div>
</body>
</html>
```

Рисунок 4.9. Генерація базового шаблону вебсторінки

На основі макета/опису інструменти формують прототипи. Приклад демонструє створення галереї з CSS Grid та hover-ефектами.

4.3.4 Інтеграція ШІ в CMS

Впровадження інструментів ШІ у системи управління контентом (CMS) стало ключовим чинником автоматизації процесів у веброзробці. Сучасні CMS вже не обмежуються статичним управлінням сторінками, а функціонують як платформи, здатні створювати, оптимізувати та персоналізувати контент у режимі реального часу.

Подальший розвиток таких систем визначається різними архітектурними підходами до інтеграції ШІ. Найпоширенішими зараз серед них є:

– *плагінний підхід* у традиційних CMS. WordPress демонструє його через Jetpack AI (https://jetpack.ai/), який автоматично генерує теги та резюме статей на основі аналізу контенту. Drupal (https://new.drupal.org/ai) застосовує модулі автокласифікації, які аналізують текст та призначають категорії. Такий підхід має низький поріг входу та просту інтеграцію, але обмежений у масштабуванні та взаємодії з ядром системи;

– *Headless-архітектура* поєднує CMS (Contentful, Strapi, Sanity) через API із зовнішніми сервісами ШІ[213]. Контент формується у headless-CMS, обробляється ШІ за допомогою вебхуків і передається у фронтенд-застосунки на Next.js або Nuxt.js з підтримкою SSR чи ISR. Такий варіант забезпечує більшу гнучкість та адаптивність для складних проєктів;

– *RAG-конвеєри* (Retrieval-Augmented Generation) для баз знань. Вони збирають контент із різних джерел, індексують у векторні сховища й генерують відповіді з цитуванням джерел[214]. Підхід застосовують для технічної документації, довідників та маркетплейсів, де важлива точність і здатність відстежувати інформацію.

Ключовими сценаріями застосування є автокласифікація та таксономія, генерація й редактура, мультимовність, модерація, автолінкування та внутрішня перелінковка, аналітика й зворотний зв’язок.

Автоматична класифікація та побудова таксономії стає можливою завдяки алгоритмам витягу сутностей (Named Entity Recognition), які розпізнають у тексті ключові поняття, персоналії, організації та географічні об’єкти. Система автоматично призначає теги та категорії, будує зв’язки між різними матеріалами й формує рекомендації «пов’язаного контенту», що значно покращує навігацію і відкриваність контенту.

Генерація та редактура контенту відбувається на кількох рівнях складності. ШІ може створювати чернетки статей на основі ключових слів або тем, автоматично генерувати метадані (title та description) для покращення SEO, створювати короткі анонси та резюме довгих матеріалів, а також розробляти детальні описи товарів для електронної комерції. Важливо, що на цьому етапі зберігається

[213] Dakowicz Ja. 5 Best Headless CMS Platforms in 2025: Reviews & Comparisons. URL: https://pagepro.co/blog/top-5-best-headless-cms-platforms/

[214] What is Retrieval-Augmented Generation (RAG)? URL: https://cloud.google.com/use-cases/retrieval-augmented-generation

людський контроль над якістю та відповідністю контенту брендовим стандартам.

Мультимовна підтримка виходить за межі простого перекладу. Сучасні системи здійснюють локальну адаптацію контенту, враховуючи культурні особливості різних ринків, синхронізують мовні версії при оновленнях та контролюють термінологічну консистентність у всіх мовах. Така підтримка особливо важлива для глобальних брендів, які потребують уніфікованого повідомлення при збереженні локальної релевантності.

Впровадження ШІ в CMS потребує переосмислення редакційних ролей. Редактор-куратор зосереджується на стратегічних рішеннях та кінцевому затвердженні контенту, контент-аналітик налаштовує метрики ефективності та KPI для оцінки якості генерованих матеріалів, а ML-інженер або адміністратор CMS забезпечує стабільну роботу ШІ-пайплайнів та моніторинг їх продуктивності.

Система контролю якості забезпечує автоматичні лінтери стилю для перевірки відповідності брендбуку, інтеграцію з фактчек-API для верифікації інформації, детекцію плагіату через сервіси як Copyscape або Turnitin, аналіз тону для забезпечення емоційної відповідності та список стоп-тем або слів для запобігання небажаному контенту.

Для забезпечення високої продуктивності застосовуються техніки Server-Side Rendering (SSR) та Incremental Static Regeneration (ISR)[215] для швидкого завантаження сторінок, використання CDN для ефективного розповсюдження медіаконтенту, організація черг на генерацію через системи як Kafka або RabbitMQ для оброблення великої кількості запитів, а також тротлінг викликів до LLM-API для контролю витрат та запобігання перевантаженню[216].

Практичні приклади впровадження демонструють Shopify Magic, який генерує описи товарів та автоматично призначає теги, використовуючи Storefront API разом з вебхуками для запуску ШІ-пайплайнів у headless-архітектурах. Google Workspace інтеграції забезпечують колаборативне редагування з підказками стилю та автоматичним резюмуванням документів, а у headless-сценаріях використовують Vertex AI для класифікації та генерації у CI/CD процесах.

4.3.5 SEO-оптимізація на основі ШІ

SEO-оптимізація з використанням ШІ має комплексний підхід, який трансформує традиційні методи просування сайтів у пошукових системах. Замість статичних ключових слів сучасні платформи дедалі частіше використовують ШІ для розуміння намірів користувача, адаптації контенту в режимі реального часу та забезпечення відповідності оновленим алгоритмам пошукових систем.

Інструменти Surfer SEO (https://surferseo.com/), MarketMuse (https://www.marketmuse.com/), Clearscope (https://www.clearscope.io/), Jasper AI (https://www.jasper.ai/) та Copy.ai (https://www.copy.ai/) разом зі спеціалізованими асистентами на базі GPT, Claude та інших LLM автоматично створюють

[215] CSR vs SSR vs ISR Rendering Patterns in Next.js. URL: https://medium.com/@sureshdotariya/csr-vs-ssr-vs-isr-patterns-in-next-js-c526757e837a

[216] Understanding Message Queues: A Guide for Developers day 40 of system design basics. URL: https://dev.to/vincenttommi/understanding-message-queues-a-guide-for-developers-day-40-of-system-design-basics-3b09

заголовки H1-H3 відповідно до пошукових запитів, добирають ключові слова на основі кластеризації запитів та оптимізують щільність ключових слів разом з LSI-термінами[217]. Системи пропонують метаописи з урахуванням CTR-аналізу, проводять глибокий аналіз конкурентів та генерують контент із високою видимістю у пошукових результатах, використовуючи технології retrieval-augmented generation для забезпечення актуальності інформації[218].

На рис. 4.10 наведено приклад Python-коду для автоматичної генерації метаданих.

```python
from openai import OpenAI

def generate_seo_metadata(title, keywords):
    client = OpenAI()
    prompt = f"""
    Згенеруй SEO-оптимізований meta description для сторінки:
    Title: {title}
    Keywords: {keywords}
    Обмежити до 160 символів. Стиль - інформативний, кликабельний.
    Враховуй E-A-T критерії Google.
    """
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.6,
        max_tokens=100
    )
    return response.choices[0].message.content
```

Рисунок 4.10. Python-код для автоматичної генерації метаданих

На рис. 4.11 показано приклад JSON-фрагмента для генерації ШІ з використанням стандартів Schema.org.

Фундаментальною складовою сучасного SEO стала інтеграція структурованих даних за стандартами Schema.org (https://schema.org/), де алгоритми ШІ автоматично розпізнають тип контенту сторінки та пропонують відповідні структуровані блоки включно з FAQ, Product, Article, Review та BreadcrumbList. Пошукові системи використовують ці дані для створення розширених сніпетів (rich snippets) у видачі, що значно покращує видимість та клікабельність результатів. Система також відстежує зміни у форматах Google та інших пошукових систем, автоматично адаптує розмітку до нових стандартів.

[217] 10 Best SEO Copywriting Tools for Founders [2025 Updated]. URL: https://founderpal.ai/best-marketing-tools/seo-copywriting-tools

[218] Best AI-Driven SEO Tools for Content Optimization in 2025. URL: https://www.outranking.io/blog/ai-driven-seo-tools-content-optimization/

```json
{
  "@context": "https://schema.org",
  "@type": "Article",
  "headline": "ШІ у веброзробці",
  "author": {
    "@type": "Person",
    "name": "Alex",
    "url": "https://example.com/author/Alex"
  },
  "datePublished": "2025-07-22",
  "dateModified": "2025-07-22",
  "publisher": {
    "@type": "Organization",
    "name": "Digital Future",
    "logo": {
      "@type": "ImageObject",
      "url": "https://example.com/logo.png"
    }
  },
  "mainEntityOfPage": {
    "@type": "WebPage",
    "@id": "https://example.com/ai-web-development"
  },
  "image": "https://example.com/ai-article-image.jpg"
}
```

Рисунок 4.11. JSON фрагмент для генерації ШІ

SEO-оптимізація на основі ШІ забезпечує неперервний моніторинг позицій, аналіз змін в алгоритмах пошукових систем та автоматичне коригування стратегій. Вони відстежують ефективність кожного елемента оптимізації: від окремих ключових слів до структурних змін сайту, надаючи детальну аналітику ROI від SEO.

4.3.6 Етичні виклики та правові аспекти

Автоматизоване створення контенту пов’язане з низкою етичних проблем. Серед них: ризики плагіату, непрозорість походження інформації, галюцинації моделей (генерування недостовірних фактів) та обмеження авторського права для творів, створених без участі людини. Відповіддю на ці виклики є впровадження гібридних підходів до генерації контенту, а саме методології, в яких ШІ виступає як виконавець або ініціатор, а людина – як критичний редактор, модератор і гарант якості. Таке поєднання автоматизації з людським контролем має назву Human-in-the-loop (HITL)[219] і має багаторівневу систему перевірок для забезпечення точності, достовірності та етичності контенту.

HITL-підходи застосовуються у створенні відповідального контенту у таких контекстах:

– *редагування та стилістична корекція*, наприклад, фахівці з контенту здійснюють перевірку на відповідність стилю бренду;

[219] What is Human-in-the-Loop (HITL) in AI & ML? URL: https://cloud.google.com/discover/human-in-the-loop

– *верифікація фактів* – журналісти або аналітики перевіряють твердження, згенеровані LLM;

– *оцінка тону та емоційного забарвлення* – маркетологи адаптують мову до цільової аудиторії;

– *контроль якості коду* – розробники переглядають фрагменти, згенеровані Copilot або CodeWhisperer.

На рис. 4.12 показано п'ятиетапний процес HITL у створенні вебконтенту:

1 етап – LLM аналіз, моделі (GPT/Claude) аналізують завдання та виконують контекстуалізацію;

2 етап – генерація контенту, яка забезпечує створення тексту, SEO-оптимізацію, структурування;

3 етап – людська перевірка, а саме: факт-чекінг, оцінка якості, етична експертиза;

4 етап – автономна корекція, тобто обробка відгуків, автоматичні правки, оптимізація;

5 етап – публікація, а саме: розміщення, моніторинг, збір аналітики.

Рисунок 4.12. Процес Human-in-the-loop у створенні вебконтенту

Цикл HITL характеризується інтегрованою системою контрольних точок (checkpoints), які розташовані на кожному критичному етапі процесу для забезпечення якості та відповідності стандартам. Центральний елемент схеми – цикл вдосконалення з неперервним навчанням, який дозволяє системі адаптуватися та покращуватися на основі накопиченого досвіду.

Технічна реалізація контролю якості подана двома ключовими компонентами: системою автоматизованої валідації та пайплайном оброблення контенту.

Перший фрагмент коду (AIContentValidator) на рис. 4.13 ілюструє клас, призначений для автоматизованої перевірки якості створеного контенту ШІ.

```python
class AIContentValidator:
    def __init__(self):
        self.fact_checker = FactCheckingAPI()
        self.plagiarism_detector = PlagiarismDetector()

    def validate_content(self, content, domain=None):
        validation_results = {
            'factual_accuracy': self.fact_checker.verify(content, domain),
            'originality': self.plagiarism_detector.check(content),
            'bias_score': self.analyze_bias(content),
            'readability': self.calculate_readability(content)
        }
        return validation_results
```

Рисунок 4.13. Система автоматизованої валідації AIContentValidator

Клас виконує кілька ключових завдань: звертається до зовнішнього API для перевірки фактів, використовує детектор плагіату, аналізує упередженість та розраховує показники читабельності. Результатом роботи є словник із комплексною оцінкою матеріалу за кількома критеріями (точність, оригінальність, відсутність упереджень, зрозумілість тексту).

Другий фрагмент (ContentQualityPipeline) (рис. 4.14) демонструє організацію процесу створення та перевірки контенту у вигляді пайплайна.

```python
class ContentQualityPipeline:
    def __init__(self):
        self.stages = [
            ('generation', self.generate_content),
            ('fact_check', self.verify_facts),
            ('style_check', self.validate_style),
            ('seo_optimization', self.optimize_seo),
            ('human_review', self.human_validation)
        ]
```

Рисунок 4.14. Пайплайн оброблення контенту ContentQualityPipeline

У класі визначено послідовність етапів: генерація контенту, фактчекінг, перевірка стилю, оптимізація під SEO та фінальна валідація людиною. Такий підхід дозволяє будувати модульні та розширювані пайплайни, де на кожному кроці застосовуються окремі алгоритми або інструменти для покращення результату.

Якість контенту, згенерованого ШІ, має оцінюватись за такими параметрами: фактична точність, релевантність ключовим запитам, стильова консистентність та читабельність і відповідність цільовій аудиторії.

Для забезпечення достовірності контенту використовуються як зовнішні сервіси, наприклад, WolframAlpha (https://www.wolframalpha.com), ClaimReview (https://www.claimreviewproject.com), Perplexity AI (https://www.perplexity.ai/), так і внутрішні моделі оцінки правдивості тверджень. GPT-моделі вбудовують

механізми фактчекінгу «на льоту» шляхом запитів до баз знань через технологію retrieval-augmented generation (RAG)[220].

На рис. 4.15 наведено приклад коду інтеграції GPT з фактчекінгом. Код демонструє практичну реалізацію такої системи з використанням векторних сховищ та ланцюгів запитів для забезпечення максимальної достовірності результатів.

У вебінтерфейсах набуває популярності використання Explainable AI[221] – систем, які пояснюють, чому була згенерована певна відповідь, текст чи зображення. Такі підходи є критично важливими для юридичних, освітніх і медичних платформ. Приклади цих підходів: виведення «шляху логіки» генерації; відображення джерел фактів; оцінка впевненості у твердженні; інтерактивне уточнення відповіді.

```python
from langchain_community.vectorstores import FAISS
from langchain_openai import OpenAI
from langchain.chains import RetrievalQA
from langchain_community.embeddings import OpenAIEmbeddings
# Завантаження індексу знань
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.load_local("faq_index", embeddings)
retriever = vectorstore.as_retriever()
# Створення ланцюга для QA
llm = OpenAI(temperature=0.1)  # Низька температура для фактичної точності
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True
)
# Запит з фактчекінгом
response = qa_chain("Чи є у ChatGPT доступ до поточних даних?")
print(f"Відповідь: {response['result']}")
print(f"Джерела: {response['source_documents']}")
```

Рисунок 4.15. Інтеграція GPT з фактчекінгом

Використання гібридних підходів дозволяє:

– уникати галюцинацій ШІ через багаторівневу перевірку;

– дотримуватися авторського права шляхом перевірки унікальності контенту (Copyscape, Turnitin);

– виконувати етичні вимоги до інклюзивності, недискримінації та культурної чутливості;

[220] Gao, Y. et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXi*. 2023. Vol. 2. DOI: https://doi.org/10.48550/arXiv.2312.10997

[221] Weiche H. et al. A Comprehensive Guide to Explainable AI: From Classical Models to LLMs. *arXiv.* 2024. DOI: https://doi.org/10.48550/arXiv.2412.00800

– ідентифікувати згенерований контент через водяні знаки, маркування AI-generated.

На рис 4.16. наведено комплексну систему етичного управління гібридною генерацією контенту, де центральним елементом є принцип етичного партнерства між людиною та ШІ для створення відповідального контенту. На цьому рисунку показано, як людський контроль забезпечує критичне мислення, стратегічні рішення та етичні судження, тоді як ШІ надає підтримку через швидкість оброблення, аналіз великих обсягів даних та креативні пропозиції. Контроль якості реалізується через багаторівневу перевірку та постійний моніторинг метрик, а управління ризиками забезпечує виявлення шкідливого контенту та превентивні заходи проти упереджень. Система зворотного зв'язку забезпечує неперервне вдосконалення через аналіз ефективності та корекцію стратегій.

Особливу увагу приділено етичним гарантіям, які стосуються заборони дискримінаційного контенту, дотримання GDPR, прозорості алгоритмів та збереження людської відповідальності за кінцеві рішення.

Рисунок 4.16. Система етичних гарантій у взаємодії людини та ШІ

Загалом гібридні підходи до генерації контенту у веброзробці не лише зменшують ризики помилок і упередженості, а й забезпечують відповідність законодавству, підвищують довіру користувачів і створюють умови для сталого використання ШІ. Ключовим принципом залишається збереження людської відповідальності за кінцеві рішення при максимальному використанні можливостей ШІ.

4.4 Інтеграція ШІ з сучасними вебфреймворками

Інтеграція ШІ із сучасними вебфреймворками, зокрема з React, Vue.js, Angular на рівні frontend та Node.js, Django, FastAPI у backend-архітектурі, відкриває нові можливості для створення інтелектуальних, адаптивних і персоналізованих вебзастосунків. Такі підходи дозволяють поєднувати алгоритми ШІ з логікою клієнтської та серверної частин, що сприяє підвищенню продуктивності, автоматизації рутинних процесів, гнучкості розроблення та покращенню користувацького досвіду. Особливу увагу приділяють застосовуванню ШІ для оптимізації завантаження компонентів, персоналізації інтерфейсів, автоматизованого тестування, а також для аналізу та візуалізації поведінки користувачів у реальному часі.

4.4.1 Теоретичні засади інтеграції ШІ у frontend-архітектуру

Клієнтська інтеграція ШІ у вебзастосунки базується на трьох основних архітектурних підходах[222]:

1) вбудований ШІ (embedded AI);
2) звернення до зовнішнього сервісу API (API-driven AI);
3) гібридна комбінація клієнтської та серверної логіки (hybrid AI).

Вбудований ШІ передбачає виконання моделі безпосередньо в браузері, що забезпечує максимальну автономність та приватність даних. Наприклад, TensorFlow.js надає гнучкий, інтуїтивний інтерфейс для побудови та розгортання моделей машинного навчання з використанням JavaScript і WebGL для високопродуктивних обчислень. Це дозволяє виконувати моделі повністю в браузері. На рис. 4.17 наведено фрагмент коду, який завантажує та запускає попередньо навчену модель безпосередньо у браузері за допомогою TensorFlow.js.

```
const model = await tf.loadLayersModel('/models/classifier.json');
const result = model.predict(preprocess(input));
```

Рисунок 4.17. Embedded AI з TensorFlow.js

Тут проілюстровано концепцію Embedded AI з TensorFlow.js, коли обчислення виконуються на клієнтському пристрої. Такий підхід забезпечує низьку затримку та гарний користувацький досвід, оскільки усуває потребу серверного оброблення, але він обмежений потужністю моделей та ресурсами браузера.

API-driven AI використовує зовнішні сервіси через API для доступу до найсучасніших моделей (наприклад, OpenAI, HuggingFace). На рис. 4.18 показано використання зовнішнього API для генерації відповіді мовною моделлю GPT.

[222] Lakshmanarao K. Different Types of Architectures in Frontend Design and Development. *International Journal for Multidisciplinary Research*. 2024. Vol. 6, Issue 5. DOI: https://doi.org/10.36948/ijfmr.2024.v06i05.28707

```
const response = await openai.createCompletion({
  model: "gpt-3.5-turbo",
  prompt: userInput
});
```

Рисунок 4.18. API-driven інтеграція

Тут код ілюструє принцип API-driven AI, коли клієнтський застосунок делегує складні обчислення сторонньому сервісу. Такий підхід розширює функціонал клієнтських застосунків, але створює залежність від мережевого з'єднання та зовнішніх провайдерів.

Гібридні рішення поєднують переваги обох підходів, де базові моделі працюють у браузері, тоді як складні обчислення делегуються серверу. Це забезпечує оптимальний баланс між швидкістю відповіді та складністю завдань.

Загалом frontend-фреймворки забезпечують швидку інтеграцію базових інтелектуальних функцій, але потребують балансування між продуктивністю та залежністю від зовнішніх сервісів.

4.4.2 Архітектурні рішення backend-інтеграції

На відміну від клієнтської частини, де обмеженням є обчислювальна потужність, на рівні backend акцент робиться на масштабованість й організацію робочих процесів. Сервери дозволяють розгортати повноцінні моделі машинного навчання, обробляти великі обсяги даних і забезпечувати інтеграцію з базами знань.

FastAPI є популярним вебфреймворком для створення API з Python, який характеризується високою продуктивністю, простим синтаксисом та вбудованою підтримкою асинхронних запитів, що робить його ідеальним для розгортання моделей машинного навчання.

На рис. 4.19 продемонстровано структуру backend-методу у FastAPI, який виконує прогноз на основі даних користувача з можливістю прямого вбудовування моделей у серверні маршрути.

```
@app.post("/predict")
async def predict(data: InputData):
    result = await model.predict_async(data.features)
    return {"prediction": result.tolist()}
```

Рисунок 4.19. Структура backend-методу у FastAPI

FastAPI використовує Pydantic для валідації запитів, автоматично перевіряє вхідні дані та надає чіткі повідомлення про помилки, що значно спрощує процес інтеграції ML-моделей і забезпечує надійність API.

Node.js також підтримує інтеграцію моделей, хоча його екосистема менш орієнтована на наукові обчислення. Завдяки бібліотеці TensorFlow.js можна

досягти сумісності з клієнтськими рішеннями, зберігаючи єдиний технологічний стек. На рис.4.20 показано приклад такої інтеграції.

```
const model = await tf.node.loadSavedModel('./model');
const y = model.predict(tf.tensor2d([features]));
```

Рисунок 4.20. Інтеграція з Node.js

Важливим напрямом розвитку backend-систем стало використання мікросервісної архітектури, де інтелектуальні модулі виділяються у самостійні сервіси, що контейнеризуються за допомогою інструментів на зразок Docker та Kubernetes. Такий підхід забезпечує гнучкість і полегшує оновлення окремих компонентів або моделей без порушення роботи основного застосунку. Основними перевагами мікросервісного підходу є технологічна незалежність (кожен сервіс може використовувати оптимальний технологічний стек) та ізоляція (відмова одного компонента ШІ не впливає на роботу системи в цілому). Мікросервісна архітектура з контейнеризацією забезпечує потужні можливості для розгортання складних ML-моделей і є стандартом для продуктивних систем.

4.4.3 Full-stack інтеграційні рішення

Фреймворки Next.js та Nuxt.js дозволяють поєднувати клієнтську і серверну логіку без чіткої межі між ними. У таких випадках функції ШІ вбудовуються безпосередньо у маршрути оброблення запитів. На рис. 4.21 наведено інтеграцію мовної моделі у маршрути Next.js. Код ілюструє full-stack підхід, за якого функції ШІ вбудовуються одночасно у клієнтську та серверну частини.

```
export async function POST(req) {
  const { prompt } = await req.json();
  const r = await fetch('https://api.openai.com/v1/completions', {
    method: 'POST',
    headers: { 'Authorization': `Bearer ${process.env.OPENAI_API_KEY}` },
    body: JSON.stringify({ model: 'gpt-3.5-turbo', prompt })
  });
  return Response.json(await r.json());
}
```

Рисунок 4.21. Інтеграція мовної моделі у маршрути Next.js

Такий підхід усуває потребу будувати окремі сервери лише для функцій ШІ, проте може створювати ризики залежності від одного фреймворку або постачальника інфраструктури. Full-stack рішення особливо ефективні для прототипування та MVP-проєктів, де швидкість розроблення є пріоритетом. Отже, Full-stack інтеграція спрощує архітектуру та прискорює розроблення, але потребує обережного планування для уникнення технологічної залежності.

Найпоширенішими сценаріями інтеграції з інтелектуальними системами є чат-боти, рекомендаційні системи та інтелектуальна UX-аналітика. Чат-боти можуть працювати в двох основних режимах: *stateless*, коли кожен запит обробляється окремо без збереження контексту діалогу, і *stateful*, коли система запам'ятовує попередні повідомлення та враховує їх при відповіді[223]. Рекомендаційні системи часто комбінують кілька методів, таких як колаборативне фільтрування та контентне фільтрування, поєднуючи результати серверних алгоритмів із локальною аналітикою користувацьких даних для надання персоналізованих рекомендацій. На рис. 4.22 показано гібридний підхід у рекомендаційних системах, а саме поєднання серверних алгоритмів і локальних моделей для персоналізації досвіду користувача.

```
const hybrid = combine([
  await fetchServerRecommendations(userId),
  generateLocalSuggestions(userHistory, localModel)
]);
```

Рисунок 4.22. Гібридний підхід у рекомендаційних системах

Подібні підходи показують, що ШІ стає не просто інструментом оброблення інформації, а частиною інтерактивної логіки взаємодії користувача із системою. Серед практичних патернів:

– *персоналізація контенту в реальному часі* – адаптація інтерфейсу на основі поведінки користувача;

– *інтелектуальне кешування* – прогнозування потреб користувача для попереднього завантаження контенту;

– *автоматизоване A/B тестування* – використання ШІ для оптимізації варіантів інтерфейсу;

– *прогнозна аналітика UX* – передбачення дій користувача для покращення досвіду.

Практичні патерни інтеграції ШІ трансформують вебзастосунки від статичних інтерфейсів до інтелектуальних систем, які адаптуються до потреб користувачів.

4.4.4 API-інтеграція та vendor ecosystem

Інтерфейси прикладного програмування є основним способом доступу до потужних мовних і візуальних моделей, розгорнутих провайдерами хмарних сервісів. Однак використання лише одного провайдера (наприклад, OpenAI чи Google) породжує проблему залежності, відому як vendor lock-in[224]. Для її

[223] Kumar A., Anand A.K., Sahoo B. From Stateless to Stateful: A Comparative Analysis of Stateful Serverless Computing Frameworks. *Proceedings of International Conference on Computational Intelligence. ICCI 2023*. Springer, Singapore. 2024. P. 223–237. DOI: https://doi.org/10.1007/978-981-97-3526-6_19

[224] Vendor Lock-in in Cloud Computing. URL: https://www.geeksforgeeks.org/mobile-computing/vendor-lock-in-in-cloud-computing/

подолання застосовуються спеціальні архітектурні патерни, наприклад проксі-шари або адаптери, які дозволяють уніфікувати виклики до різних сервісів.

Рис. 4.23 ілюструє шаблон Adapter для уніфікації роботи з різними провайдерами ШІ. Такий підхід зменшує ризик vendor lock-in і підвищує гнучкість архітектури, що дозволяє розробникам зберігати можливість переключатися між постачальниками без повної перебудови системи.

```
class AIAdapter {
  async generate(prompt, provider) { /* … */ }
}
```

Рисунок 4.23. Шаблон Adapter для роботи з різними провайдерами ШІ

Загалом ефективна API-інтеграція базується на чотирьох ключових принципах:

1) *абстракція провайдерів* забезпечується через уніфіковані інтерфейси, які дозволяють працювати з різними вебсервісами через єдиний API;

2) *фолбек-механізми* автоматично переключають на резервні сервіси при збоях, гарантуючи неперервність роботи;

3) *моніторинг витрат* контролює використання ресурсів та оптимізує навантаження, покращуючи ефективність запитів;

4) *інтелектуальне кешування* зменшує навантаження на систему, завдяки алгоритмам, які вибирають та зберігають часто використовувані відповіді.

Правильна організація API-інтеграції з множинними провайдерами дозволяє забезпечити гнучкість і надійність системи, мінімізуючи ризики технологічної залежності.

4.4.5 Продуктивність та оптимізація

Швидкодія інтегрованих моделей безпосередньо впливає на користувацький досвід. Для оптимізації застосовують техніки стискання моделей (quantization, pruning)[225], відкладене завантаження модулів (lazy loading)[226] та навіть винесення моделей на крайову інфраструктуру CDN.

TensorFlow.js рекомендує «розігрівання» моделі перед використанням, передаючи тестовий тензор такої самої форми для покращення продуктивності першого прогнозу. Це критично важливо для користувацького досвіду в реальних застосунках.

Застосовуються такі техніки: квантування моделей, периферійні обчислення, асинхронна обробка, прогресивне завантаження та кешування результатів для збереження часто використовуваних прогнозів. Ці техніки працюють

[225] Singh S., Sharma K., Karna B.K., Raj P. Pruning and Quantization for Deeper Artificial Intelligence (AI) Model Optimization. *Intelligent Control, Robotics, and Industrial Automation.* 2023. Vol. 1066. P. 933–945. DOI: https://doi.org/10.1007/978-981-99-4634-1_73

[226] Bâra R., Costin A., Cătălin T. Analysing the performance impacts of lazy loading in web applications. *Journal of Information Systems & Operations Management*. 2024. Vol. 18.1. P. 1-15. URL: http://web.rau.ro/websites/jisom/Vol.18%20No.1%20-%202024/JISOM%2018.1.pdf

узгоджено: квантування моделей зменшує їх розмір у 4-8 разів без суттєвої втрати точності, edge computing через CDN зменшує затримку мережі, асинхронна обробка забезпечує відгук інтерфейсу під час обчислень, а прогресивне завантаження дозволяє користувачеві почати роботу до повного завантаження всіх компонентів.

Інтеграція ШІ у сучасні вебфреймворки трансформує традиційні архітектури у гнучкі інтелектуальні системи, здатні до самонавчання та адаптації під користувача. Найефективніші стратегії поєднують клієнтські, серверні, full-stack та API-підходи, підтримуючи мікросервісну архітектуру та адаптери для уникнення vendor lock-in. Ключовим викликом лишається продуктивність і забезпечення прозорості роботи моделей, що напряму впливає на користувацький досвід.

4.5. Висновки до розділу

У розділі проведено комплексний аналіз теоретичних і практичних засад інтеграції штучного інтелекту у веброзробку та вебдизайн. Розглянуто концептуальні основи, технологічні компоненти та історичну еволюцію, що дало змогу виокремити ключові тенденції: перехід від статичних рішень до інтелектуальних адаптивних систем і формування парадигми «AI-first» у сучасних вебтехнологіях.

Показано, що ефективне впровадження ШІ потребує цілісної архітектури, яка поєднує традиційні вебтехнології та інтелектуальні модулі. П’ятирівнева модель забезпечує розділення відповідальності між компонентами, масштабованість, гнучкість і незалежний розвиток функціональних блоків. Практичні приклади охоплюють клієнтські, серверні та full-stack рішення, мікросервісну архітектуру і механізми оркестрації, що дозволяють інтегрувати ML-моделі на різних рівнях.

Особливу увагу приділено використанню ШІ у UX/UI-дизайні та контент-менеджменті: замкнені цикли неперервної оптимізації інтерфейсів, генерація текстового й мультимедійного контенту, автоматизація редакційних процесів у CMS, багатомовність і локалізація. Показано роль ШІ в SEO-оптимізації: від автоматичної побудови таксономій та структурованих даних Schema.org до використання retrieval-augmented generation для створення актуального контенту і розширених сніпетів.

Окремо проаналізовано аспекти забезпечення якості, безпеки та етики. Підкреслено важливість human-in-the-loop підходів, багаторівневої валідації контенту, Explainable AI (пояснюваних моделей) та дотримання правових вимог і стандартів прозорості. Такі механізми підвищують довіру користувачів, знижують ризики упередженості та забезпечують контроль за якістю автоматично створюваного контенту.

Загалом інтеграція ШІ у веброзробку виходить за межі простої автоматизації окремих процесів і трансформує методологію створення цифрових продуктів. ШІ стає базовим елементом інтелектуальних екосистем, орієнтованих на симбіоз людини і машини, динамічну адаптацію до контексту та сталий розвиток вебтехнологій.

5 Застосування штучного інтелекту в розробленні та тестуванні ігор

5.1 Сфери застосування штучного інтелекту в розробленні та тестуванні ігор

Індустрія цифрових розваг, зокрема ринок відеоігор, демонструє стрімке зростання, перетворюючись на провідний сегмент глобального медіапростору. За прогнозами аналітиків[227], обсяги світового ринку відеоігор досягнуть 503,14 млрд доларів США до 2033 року. Така динаміка зумовлюється не лише інноваціями в апаратному забезпеченні, а й активним впровадженням штучного інтелекту (ШІ) у ключові етапи життєвого циклу ігрових продуктів. Технології ШІ відіграють визначальну роль у трансформації парадигм створення, верифікації та оптимізації ігрового контенту, забезпечуючи підвищення ефективності розроблення, зниження виробничих витрат та зростання якості кінцевого продукту.

Наразі вже не виникає питання щодо доцільності застосування штучного інтелекту (ШІ) у галузі розроблення відеоігор (Game Development або GameDev), адже його використання стало звичною та інтегрованою частиною виробничого процесу. Станом на сьогодні ШІ активно впроваджується як на етапах проєктування та розроблення, так і в процесах тестування, оптимізації й підтримки тривимірних ігор (3D-ігор). Тенденції розвитку технологій ШІ свідчать про надзвичайно широкі перспективи їх застосування в індустрії GameDev, що зумовлює трансформацію підходів до створення ігрового контенту, управління геймплеєм та взаємодії з користувачем[228].

Інноваційні ШІ-інструменти вже з'явилися на ринку, їх активно використовують для автоматизації ключових процесів, підвищення якості цифрового контенту та удосконалення ігрового досвіду. Такі інструменти дають змогу розробникам значно скоротити час на створення активів гри, забезпечити більш складну й адаптивну поведінку некерованих гравцем персонажів (Non-Player Characters, NPC), а також ефективно реалізовувати автоматизоване тестування ігор. У результаті зростає ефективність команд розроблення, покращується продуктивність, а ігрові світи стають глибшими, реалістичнішими та більш динамічними.

Наукове дослідження прикладних аспектів використання ШІ в розробленні ігор є актуальним і стратегічно важливим, оскільки надає змогу ідентифікувати як потенційні переваги, так і виклики, пов'язані з впровадженням цих технологій

---

[227] Global video game market value from 2022 to 2032 (in billion U.S. dollars). *Statista*. URL: https://www.statista.com/statistics/292056/video-game-market-value-worldwide/

[228] Trofymenko O., Dyka A., Loboda Y., Tolocknov A., Bondarenko M. Artificial Intelligence in the GameDev. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*. 2025. Vol. 27. P. 109-119. DOI: https://doi.org/10.28925/2663-4023.2025.27.701

у різні етапи розроблення. Це підтверджується публікаціями[229], де аналізують як технічні, так і етичні, соціальні та творчі аспекти використання ШІ в розробленні ігор. Такий науковий інтерес засвідчує високий рівень актуальності проблематики та наголошує на потребі систематизації знань у цій сфері.

Слід зазначити, що ШІ-алгоритми можуть бути ефективно інтегровані у різні фази життєвого циклу гри, охоплюючи концептуальне проєктування, генерацію контенту, верифікацію функціональності, аналітику поведінки гравця, а також персоналізовану підтримку користувача (рис. 5.1)[230].

Рисунок 5.1. Сфери застосування ШІ в GameDev

ШІ-технології охоплюють широке коло задач у процесі розроблення відеоігор. Найпоширенішими сферами застосування інструментів ШІ в розробленні ігор є: автоматичне написання та ревізія програмного коду, генерація музики та озвучення персонажів, динамічне формування сюжетних ліній і діалогів, виявлення логічних аномалій у геймплеї через алгоритми тестування. Крім того, сторонні провайдери ігрових сервісів також дедалі частіше застосовують ШІ для забезпечення автоматизованого перекладу, модерації контенту, надання технічної підтримки та вдосконалення комунікації з гравцями.

[229] Yang D., Kleinman E., Harteveld C. GPT for Games: An Updated Scoping Review (2020-2024). *arXiv*. 2024. Vol. 2411.00308. https://doi.org/10.48550/arXiv.2411.00308

[230] Трофименко О.Г., Дика А.І., Задерейко О.В., Шевченко О.В. Сфери застосування штучного інтелекту в розробці та тестуванні ігор. *Кібербезпека: освіта, наука, техніка*. 2025. № 1(29). С. 41-59. DOI: https://doi.org/10.28925/2663-4023.2025.29.832

5.2 Інструменти штучного інтелекту в розробленні відеоігор

5.2.1 Інструменти штучного інтелекту в процесі створення графічних елементів, 3D-моделей та інших компонентів відеоігор

Розроблення 3D-ігор є складним і багатогранним процесом, який поєднує в собі технічну експертизу, художню творчість та глибоке розуміння механіки ігрового процесу. Для створення якісного продукту необхідна тісна міждисциплінарна взаємодія між програмістами, художниками, дизайнерами рівнів, аніматорами, сценаристами, геймдизайнерами та тестувальниками. Кожен з цих фахівців робить внесок у створення комплексного інтерактивного середовища, яке повинне не лише функціонувати бездоганно, а й забезпечувати гравцю емоційно захопливий досвід.

Застосування інструментів ШІ у процесі створення 3D-ігор стало вагомим чинником трансформації традиційних підходів до розроблення ігрових продуктів. Сучасні системи на базі ШІ активно використовуються для автоматизованого створення візуального та аудіовізуального контенту, моделювання тривимірних об’єктів, текстурування, генерації анімацій, а також написання сценаріїв та діалогів. Окрім цього, ШІ застосовується для синтезу музики, озвучення персонажів, розроблення логіки геймплею, автоматичного написання програмного коду та проведення інтелектуального тестування продукту (рис. 5.2)[231].

Рисунок 5.2. ШІ-інструменти в GameDev

Суттєвий вплив інструментів генеративного ШІ на виробничий процес у GameDev підтверджується емпіричними даними. Згідно з результатами

[231] Трофименко О. Г., Задерейко О. В., Логінова Н. І., Шевченко О. В. Інструменти штучного інтелекту в розробці графічних елементів, 3D-моделей та інших компонентів відеоігор. *Інформатика та математичні методи в моделюванні*. 2025. Т.15, № 2. С. 276-287. DOI: https://doi.org/10.15276/imms.v15.no2.276

аналітичного опитування[232], проведеного на початку 2025 року, близько 45% розробників відеоігор активно впроваджують генеративні ШІ-алгоритми для створення ігрового контенту, зокрема в частині 3D-моделювання, текстурування та поведінкових моделей персонажів. Учасники дослідження відзначили зростання продуктивності своєї роботи в середньому на понад 20%, завдяки використанню таких інструментів. Вказані дані свідчать про те, що технології генеративного ШІ мають значний потенціал для подальшої трансформації індустрії розроблення відеоігор. Зокрема, вони сприяють не лише підвищенню швидкості та ефективності розроблення, а й зниженню вхідного бар'єру для нових учасників ринку. Автоматизація рутинних або ресурсозатратних завдань дозволяє творчим командам зосередитись на концептуальній роботі, інноваційному дизайні та покращенні загального користувацького досвіду.

У цілому використання ШІ в генерації графічних компонентів ігрового простору створює нові можливості для масштабування процесів, зменшення витрат і підвищення конкурентоспроможності кінцевого продукту. Це відкриває новий етап у розвитку GameDev, де технології ШІ поступово стають не допоміжним інструментом, а невіддільною складовою творчого і технічного процесу розроблення.

5.2.2 Інструменти ШІ для генерації 3D-моделей та текстур в розробленні ігрових інтерфейсів

Графічний інтерфейс користувача (Graphical User Interface, GUI) є важливою складовою будь-якого програмного забезпечення, а в контексті відеоігор він відіграє визначальну роль у формуванні першого враження користувача та забезпеченні ефективної взаємодії з ігровим середовищем. Застосування інструментів ШІ у сфері UI/UX-дизайну дозволяє суттєво змінити традиційні підходи до створення інтерфейсних елементів. Завдяки можливості автоматизованої генерації високоякісної графіки, ШІ-алгоритми забезпечують значну економію часу і ресурсів, що є особливо актуальним для інді-команд та невеликих студій, які працюють в умовах обмеженого бюджету та стиснених термінів.

Однією з провідних платформ, яка надає інструменти для генерації графічного контенту з використанням ШІ, є Leonardo.Ai (https://leonardo.ai/). Цей сервіс дозволяє створювати стилістично узгоджені візуальні компоненти на основі текстових описів або попередніх ескізів, що значно спрощує процес розроблення концепт-артів, ілюстрацій та інтерфейсних рішень. Зокрема, функціонал платформи дозволяє розробникам отримати якісні зображення персонажів, предметів та елементів ігрового середовища з урахуванням заданої стилістики та настрою.

Процес створення 3D-моделей традиційно починається з етапу концептуального проєктування, який охоплює пошук референсів, аналіз візуальних аналогів та створення ескізів. ШІ-інструменти, як-от Leonardo.Ai, дають змогу автоматизувати цей етап: на основі ключових слів, які описують бажані риси

[232] AI use in video game development - statistics & facts. Statista. URL: https://statista.com/topics/13437/artificial-intelligence-ai-use-in-video-game-development/

персонажів чи об’єктів, система може згенерувати низку концептуальних варіантів, які розробники використовують як основу для подальшої деталізації.

Для генерації елементів ігрового середовища – зокрема ворогів, зброї, декоративних об’єктів, пасток, джерел світла тощо – ефективно застосовується шаблон «Game Concept». Цей шаблон орієнтований на створення зображень з високим рівнем контрастності та візуальної чіткості, що є важливим для подальшого використання згенерованого контенту в реальному ігровому просторі. Рекомендується створювати кілька варіантів одного елемента, аби обрати найбільш відповідний образ для інтеграції в загальну стилістику гри.

Зображення, створені за допомогою Leonardo.Ai, можуть бути доопрацьовані засобами інструмента Canvas, який дозволяє редагувати окремі фрагменти композиції, додаючи нові деталі або коригуючи наявні. Інструмент Realtime Canvas, доступний на платформі Leonardo.Ai, забезпечує можливість перетворення простих ескізів у деталізовані об’єкти в режимі, наближеному до реального часу, що суттєво підвищує продуктивність команди розробників.

Крім того, Leonardo.Ai можна застосовувати для розроблення елементів HUD (Heads-Up Display) – інтерфейсу, який забезпечує візуалізацію важливої інформації під час гри без порушення загального ігрового процесу. Завдяки інтеграції таких ШІ-інструментів у виробничий процес, значно прискорюється розроблення ігрового продукту, дозволяючи досягти високої якості графічного контенту при збереженні гнучкості та інноваційного підходу.

Загалом, використання платформи Leonardo.Ai в контексті розроблення графічного інтерфейсу не лише сприяє оптимізації робочих процесів, а й відіграє важливу роль у формуванні естетичної цілісності гри, що безпосередньо впливає на емоційне сприйняття продукту кінцевим користувачем.

5.2.3 Інструменти ШІ для створення 3D-моделей для відеоігор

Візуальна складова відеоігор має критичне значення для формування естетичного сприйняття користувача, оскільки вона безпосередньо впливає на занурення в ігровий процес та емоційне сприйняття продукту. За умов зростання вимог до якості графічного контенту автоматизація процесу створення 3D-моделей, текстур та анімацій стала стратегічно важливим аспектом розроблення. Традиційні методи створення високоякісних 3D-моделей вимагають значних часових витрат та високого рівня кваліфікації від художників. Водночас, для початківців освоєння складних інструментів, таких як Maya чи Blender, може бути надзвичайно складним завданням. Крім того, вартість професійного програмного забезпечення і технічна складність інструментів робить створення контенту значно дорожчим та трудомістким.

Одним із сучасних рішень для розв’язання цієї проблеми є платформа Meshy.ai, яка демонструє те, як ШІ може значно спростити та прискорити процес створення 3D-моделей та текстур. Завдяки інтуїтивно зрозумілому інтерфейсу, користувачі, навіть без досвіду в моделюванні, можуть генерувати базові 3D-структури шляхом завантаження ескізів або введення текстових описів. Алгоритми платформи аналізують введені дані, розпізнають геометричну структуру

об’єктів та властивості матеріалів, що дозволяє створювати реалістичні моделі з урахуванням освітлення, текстур та глибини. Процес моделювання має три основні етапи: оброблення текстового запиту за допомогою великих мовних моделей, генерацію початкової 3D-структури та детальне текстурування й коригування моделі.

Ключовою перевагою Meshy.ai є можливість створювати геометрично складні об’єкти з високою деталізацією та реалістичними текстурами. Використання глибоких нейронних мереж дозволяє аналізувати великі масиви зразків текстур і генерувати нові, що відповідають сучасним стандартам графіки. Оптимізований процес генерації дозволяє отримати готові 3D-моделі в різних форматах (.fbx, .obj, .glb) з інтегрованими картами текстур[233].

Meshy.ai створює об’єкти, в яких оптимально поєднуються полігональна складність і рівень деталізації, що робить їх придатними для широкого спектра застосувань: від інтер’єрного дизайну (меблі, декор) до створення персонажів, транспортних засобів, органічних структур, архітектурних елементів та інших складових віртуального середовища. Платформа також дозволяє користувачам вибирати різні матеріали (метал, дерево, тканина, скло тощо) та комбінувати їх для досягнення необхідних візуальних ефектів. Підтримка генерації текстур на основі текстових запитів (text-to-texture) та зображень (image-to-texture) додає гнучкості в процесі створення візуальних компонентів[234].

Процес створення 3D-моделей персонажів гри можна починати з концептуальних зображень, що дозволяє платформі Meshy.ai автоматично генерувати базову геометрію та текстури для об’єкта. На цьому етапі користувачі можуть перевірити результати безпосередньо в середовищі Meshy.ai або у сторонніх 3D-редакторах, таких як Blender, ZBrush, 3ds Max, Maya. Після цього виконується коригування геометрії, виправлення дефектів полігональної сітки та ретопологізація, що критично важливо для подальшої анімації.

Наступний етап – текстурування – стосується створення UV-розгортки та генерації карт нормалей, рельєфу і глобального освітлення, що поєднуються з базовими текстурами для досягнення максимальної реалістичності. Платформа Meshy.ai також дозволяє застосовувати стилістичні ефекти через функцію Stylize, що дає змогу налаштовувати зовнішній вигляд 3D-об’єкта відповідно до певних художніх вимог. Оскільки моделі використовуються в інтерактивних середовищах, наступним етапом є створення скелетної анімації (rigging). Вбудована функція Rig автоматизує прив’язку скелетної структури до геометрії моделі, що дозволяє провести тестування анімаційних деформацій і перевірку сумісності з анімаційними сценаріями.

Завершальним етапом є експорт готової 3D-моделі у формат, сумісний з основними ігровими рушіями, такими як Unity або Unreal Engine, для подальшої інтеграції в ігровий проєкт.

Моделі, створені за допомогою Meshy.ai, відзначаються високим рівнем деталізації, точною анатомічною будовою та реалістичними текстурами, що

[233] Meshy AI Review: How I Generated 3D Models in One Minute. URL: https://www.unite.ai/meshy-ai-review/
[234] Meshy AI: Transform Text into 3D Textures. URL: https://cutout.pro/learn/meshy-ai/

забезпечує їх ефективну інтеграцію в ігрові рушії та безперебійну анімацію в середовищах Unity та Unreal Engine.

Отже, Meshy.ai є ефективним і потужним інструментом на базі ШІ, що дозволяє розробникам відеоігор значно пришвидшити процес створення якісних 3D-активів, зменшуючи витрати часу на ручне моделювання та забезпечуючи високий рівень графічного контенту.

5.2.4 Інструменти ШІ для створення музики у відеоіграх

Аудіовізуальний супровід є одним з важливих аспектів, які формують ігрову атмосферу та безпосередньо впливають на емоційне сприйняття гравця. Музика у відеоіграх додає глибину візуальному й емоційному досвіду, підсилює драматургічні моменти та посилює взаємодію з ігровим середовищем. Традиційний процес створення музичних композицій для відеоігор зазвичай потребує залучення професійних композиторів, використання дорогих студійних технологій та спеціалізованого обладнання, що може вимагати значних фінансових та часових витрат. Однак впровадження новітніх технологій, зокрема штучного інтелекту, відкриває нові горизонти у розробленні музичного супроводу, робить цей процес більш доступним, швидким та адаптивним.

Одним із прикладів інноваційних ШІ-інструментів є платформа AIVA (Artificial Intelligence Virtual Artist), яка дозволяє генерувати музичні композиції в понад 250 різних стилях за лічені секунди[235]. Особливістю AIVA є використання алгоритмів глибокого навчання, які дозволяють системі аналізувати величезні обсяги музичних творів, створених класичними та сучасними композиторами. Завдяки цьому AIVA виявляє структурні закономірності та стилістичні особливості, які можуть бути використані для створення нових творів за заданими параметрами.

Принцип роботи AIVA ґрунтується на застосуванні глибинних нейронних мереж, які дозволяють системі не лише відтворювати вже наявні композиції, а й генерувати варіативні музичні фрагменти. Це дозволяє уникнути одноманітності у створюваних композиціях і забезпечити їх високий рівень емоційної виразності. Для досягнення цього результату платформа аналізує та вивчає багатий музичний матеріал, щоб синтезувати композиції, які звучать органічно та відповідають вимогам гравця, розробника чи конкретної ситуації в грі.

У контексті розроблення відеоігор AIVA значно оптимізує процес створення музики, дозволяє швидко генерувати унікальне звукове оформлення, яке підкреслює атмосферу відповідної гри. Згенеровані композиції мають динамічний, емоційно насичений характер та адаптуються до розвитку подій у грі, тим самим сприяє зануренню гравця у віртуальний світ. Наприклад, музика може змінюватися залежно від того, чи перебуває гравець у розслабленій обстановці, чи, навпаки, переживає напружені моменти битви або дослідження нових територій. Цей рівень адаптації дозволяє створювати захопливий досвід, де музика стає інтегральною частиною ігрового процесу, підсилюючи емоційний відгук та рівень занурення.

---

[235] AIVA Technology – Composing Music using AI. URL: https://d3.harvard.edu/platform-digit/submission/aiva-technology-composing-music-using-ai/

Окрім того, платформа AIVA надає можливість інтеграції у різні ігрові проєкти, що дозволяє розробникам легко створювати музичні треки без великих витрат часу та ресурсів. Вона забезпечує доступ до інструментів для налаштування композицій, дозволяє персоналізувати музичний супровід відповідно до конкретних емоційних тонів та атмосфери кожної ситуації в грі. Це не лише знижує витрати на створення звукового оформлення, а й надає розробникам більше свободи в експериментах з різними музичними стилями і настроями.

Тим самим впровадження штучного інтелекту в процес створення музики для відеоігор не лише змінює саму технологію виробництва музичних композицій, а й відкриває нові можливості для створення динамічних, адаптивних та емоційно насичених звукових середовищ. Як наслідок ігри набувають нової глибини та виразності, що робить взаємодію з ними більш захопливою та неповторною для кожного гравця.

5.2.5 Інструменти ШІ для автоматизованого озвучення персонажів відеоігор

Застосування ШІ-технологій в індустрії відеоігор відкриває нові можливості для автоматизації багатьох аспектів розроблення, серед яких особливо важливим є процес озвучення персонажів. У традиційному підході створення голосового супроводу для ігор вимагає залучення професійних акторів, оренди звукозаписних студій та ретельного постпродакшн-оброблення, що може бути як дорогою, так і часозатратною процедурою. Однак, завдяки швидкому розвитку штучного інтелекту, такі процеси зазнали суттєвих змін, зробивши озвучення персонажів більш доступним та ефективним. Сучасні технології, як-от Replica Studios та інші подібні сервіси, значно полегшують створення високоякісних голосових треків і при цьому знижують витрати часу й ресурсів[236].

Однією з найбільших переваг використання ШІ для озвучення є здатність генерувати голоси, що відповідають вимогам високої реалістичності, що робить ігровий досвід більш захоплюючим та правдоподібним. Системи, що базуються на штучному інтелекті, забезпечують широкий спектр голосових варіацій, дозволяючи розробникам вибирати з численних голосових шаблонів, а також налаштовувати інтонацію, темп і емоційний колорит голосу. Це дозволяє створювати індивідуальні голоси для кожного персонажа гри, що є особливо важливим для створення глибоких і багатих наративних складових. Завдяки такій технології розробники можуть повністю контролювати аудіовізуальний аспект гри, гарантуючи, що кожен персонаж отримує унікальне озвучення, яке відповідає його характеру, ролі в сюжеті та емоційній палітрі гри.

ШІ-технології зробили надали інді-розробникам доступ до готових платформ для синтезу мовлення. Такі платформи мають інтеграційні інструменти та SDK (Software Development Kit), а також відкриті бібліотеки та API-інтерфейси, які значно спрощують процес інтеграції голосового супроводу в ігри. Це дозволяє зменшити фінансові витрати та прискорити розроблення, що є особливо

[236] Replicastudios: AI Voice Actors for Games, Films, and Animation I. URL: https://hunts-ai.onrender.com/ai/replicastudios/

важливим для невеликих студій з обмеженими ресурсами. Застосування таких технологій дозволяє створювати професійне озвучення без потреби залучення великої команди акторів чи звукорежисерів.

Яскравим прикладом успішного використання ШІ в озвученні персонажів є проєкт «The Ascent», розроблений шведською компанією Neon Giant. У цьому проєкті разом з партнерами Sweet Justice Sound та Altered Studio використана інноваційна технологія для генерації голосів персонажів за допомогою ШІ[237]. Завдяки використанню попередньо записаного голосу актора Сема Х'юза система згенерувала понад 800 нових діалогових реплік для 40 NPC[238]. Це дозволило значно покращити рівень залучення до гри, надавши персонажам голоси, які відповідають їхній індивідуальності та ролі в сюжеті, навіть за обмежених ресурсів.

Крім того, застосування інструментів ШІ для озвучення персонажів має важливі переваги в контексті локалізації контенту. Технології синтезу мовлення на базі ШІ у поєднанні з машинним перекладом дозволяють значно спростити і автоматизувати процес перекладу та озвучення діалогів для різних мовних груп. Це важливо для глобалізації відеоігор, оскільки дозволяє зберігати інтонаційні особливості, емоційний фон та культурні аспекти мови при перекладі. Такі системи можуть автоматично адаптувати голоси та мовний стиль для кожної мови, що забезпечує високу якість локалізації і доступність гри для міжнародної аудиторії. Це розширює потенціал ігор на світових ринках, дозволяє досягти ширшої аудиторії та забезпечити гравців із різних країн доступом до гри на їхній рідній мові.

Отже, впровадження ШІ в процес озвучення персонажів у відеоіграх не лише значно знижує витрати часу та коштів на виробництво аудіоконтенту, а й забезпечує високий рівень реалістичності та інтерактивності, що підвищує емоційне занурення гравця в ігровий процес. Враховуючи можливість локалізації та персоналізації голосових варіацій, ШІ стає незамінним інструментом у сучасній розробленні відеоігор, забезпечуючи нові можливості для створення інтерактивних, багатокультурних та глибоких наративних досвідів.

5.2.6 Інструменти ШІ для створення анімації у відеоігровій індустрії

У традиційній практиці створення анімаційного контенту для відеоігор потребує покадрового ручного моделювання, яке вимагає значних часових і трудових ресурсів. Проте стрімкий розвиток технологій ШІ, зокрема алгоритмів глибокого навчання, трансформує цей процес і дозволяє автоматизувати анімаційне моделювання на основі аналізу рухів реальних людей та їх подальшої адаптації до віртуального середовища.

Одним із ключових напрямів застосування ШІ в анімації є автоматична генерація рухів персонажів. Глибокі нейронні мережі здатні опрацьовувати відеодані з рухами людини, виділяти кінематичні патерни та генерувати послідовності анімацій з високим ступенем плавності й достовірності. Такий підхід не лише скорочує тривалість виробництва, а й підвищує якість результатів, завдяки здатності

[237] Furkan Ö. How AI Voice Technology is Transforming the Gaming Industry. URL: https://speaktor.com/ai-voice-in-gaming/
[238] Neon Giant teams up with Altered for voice in action RPG game The Ascent. URL: https://www.altered.ai/blog/neon-giant-teams-up-with-altered-for-voice-in-action-rpg-game-the-ascent/

моделі передбачати наступні стани руху та формувати реалістичні інтерполяції між позами. Це особливо актуально за потреби відтворення складних динамічних рухів, наприклад, у бойових іграх або спортивних симуляторах.

Показовим прикладом ефективного використання таких технологій є платформа DeepMotion (https://www.deepmotion.com). Вона надає можливість користувачам завантажувати відеозаписи з реальними рухами, які потім обробляються за допомогою ШІ-модулів. Алгоритми DeepMotion на основі системи Motion Intelligence, яка функціонує у хмарному середовищі, автоматично ідентифікують опорні точки скелетної структури людини та трансформують їх у тривимірну анімацію персонажа. При цьому підтримуються складні рухові активності, включно з елементами бойових мистецтв, атлетики, паркуру та хореографії[239]. Важливою перевагою платформи є її інтеграційна сумісність з поширеними ігровими рушіями Unity та Unreal Engine, що забезпечує неперервність розроблення та зменшує потребу у додатковій ручній адаптації анімацій.

Процес створення цифрової анімації розпочинається з аналізу завантаженого відео, під час якого система автоматично визначає скелетні маркери (ключові точки). Користувач може налаштувати параметри трекінгу — наприклад, відслідковування окремих частин тіла або обмеження руху лише до верхнього сегмента тулуба. На основі цих даних ШІ формує тривимірну скелетну структуру з відповідною кінематикою, здатною достовірно відтворювати динаміку руху суб'єкта. Результати відображаються в інтерактивному 3D-переглядачі, де розробник може оцінити точність трекінгу та за потреби внести корекції. Після завершення процесу оброблення фінальний результат експортується у формат, підтримуваний цільовим середовищем розроблення гри.

Загалом впровадження інструментів ШІ у виробництво анімації для відеоігор сприяє значному підвищенню ефективності та якості розроблення, відкриває нові можливості для творчої реалізації складних рухових сценаріїв без потреби у використанні дорогих систем захоплення руху (motion capture).

5.2.7 Інструменти ШІ для автоматизації програмування у відеоігровій розробленні

Програмування відеоігор традиційно характеризується високою складністю та вимогливістю до точності, оскільки кожна помилка у коді може призвести до критичних збоїв у роботі програмного продукту. Особливо це стосується складних інтерактивних систем, які поєднують численні модулі: фізику, штучний інтелект персонажів, графіку та мережеву взаємодію. Проте останні досягнення у сфері ШІ значною мірою трансформують процес програмування, позаяк пропонують інструменти для автоматизованої генерації, перевірки та оптимізації коду.

Одним із найпомітніших прикладів таких інструментів є GitHub Copilot, створений у співпраці GitHub та OpenAI[240]. Цей інструмент функціонує на базі

[239] Tokarev K. DeepMotion: AI Driven Motion for Games. URL: https://80.lv/articles/deepmotion-ai-driven-motion-for-games

[240] What is GitHub Copilot? URL: https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot

моделей машинного навчання, здатних аналізувати контекст введеного коду, пропонувати завершення фрагментів, генерувати шаблони функцій і навіть знаходити оптимальні способи реалізації певних логічних структур. Наприклад, при створенні ігрової механіки – умовного бойового алгоритму або обчислення траєкторії руху об'єкта – Copilot може надати кілька варіантів реалізації з урахуванням попереднього коду, тим самим пришвидшуючи процес розроблення.

Крім генерації коду, інтелектуальні системи здатні здійснювати аналіз програмного забезпечення на відповідність найкращим практикам інженерії. Це стосується виявлення логічних помилок, можливих вразливостей у безпеці, недоцільного використання пам'яті чи перевантаження обчислювальних ресурсів. Такий підхід дозволяє виявляти дефекти ще на ранніх етапах життєвого циклу програмного забезпечення, що знижує витрати на подальше тестування та усунення багів. Наприклад, якщо у грі використовується складна система штучного інтелекту NPC (неконтрольованих персонажів), ШІ-асистент може попередити про неефективну рекурсію або відсутність оброблення виняткових ситуацій, які могли б спричинити нестабільність гри під час виконання.

Практичні результати свідчать про високу ефективність подібних інструментів. Згідно з аналітичним звітом[241] понад 1 мільйон розробників вже активно використовують GitHub Copilot, а понад 20 000 організацій інтегрували його у свої робочі процеси. Дослідження показують, що Copilot дозволяє скоротити час виконання завдань приблизно на 55%, а продуктивність розробників зростає на 30%. Це потенційно сприятиме зростанню світового ВВП на понад 1,5 трильйона доларів США до 2030 року[242], що вказує на потужний економічний ефект та підтримує тенденцію до цифрової трансформації індустрії програмного забезпечення.

Тим самим, впровадження інструментів ШІ у розроблення відеоігор дозволяє значно підвищити ефективність роботи розробників, покращити якість програмного коду, зменшити кількість помилок, а також зосередитися на креативних та інноваційних аспектах ігрового дизайну. У перспективі це сприятиме створенню стабільних, безпечних і конкурентоспроможних продуктів у висококонкурентному середовищі ігрової індустрії.

5.2.8 Інструменти ШІ для створення діалогів у відеоігрових проєктах

Процес написання сценаріїв та діалогів для відеоігор є важливою складовою наративного дизайну й вимагає не лише граматичної та стилістичної точності, а й відповідності сюжетним вимогам, жанровим характеристикам та інтонаційній відповідності персонажів. Особливу складність становить необхідність створення варіативних реплік для нелінійних ігор із гнучкою структурою діалогів та можливістю численних сюжетних відгалужень. У цьому контексті вагомим інструментом стають сучасні мовні моделі на базі ШІ, зокрема великі моделі оброблення природної мови (Large Language Models, LLMs), наприклад, GPT. Ці

[241] What is GitHub Copilot? URL: https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot

[242] The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot. URL: https://github.blog/news-insights/research/the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-github-copilot/

моделі здатні генерувати лексично та синтаксично коректні тексти, які враховують попередній контекст, жанрову стилістику, емоційне забарвлення реплік та логічну послідовність сюжетних подій. Так, використання GPT-моделей дозволяє сценаристам отримувати кілька альтернативних варіантів діалогу, що спрощує вибір оптимального варіанту за тоном, ритмом або глибиною емоційної виразності. Це значно скорочує часові витрати на ручне редагування та дає змогу зосередитися на створенні більш складних сюжетних ліній.

Також інтелектуальні системи ефективно виявляють логічні та фактологічні неузгодженості у сценаріях, пропонують відповідні виправлення. Така функціональність корисна для забезпечення сюжетної цілісності, зокрема у великих ігрових проєктах, де взаємодіють сотні діалогових вузлів та гравець може впливати на перебіг подій. Окрім цього, ШІ-технології підтримують автоматизовану локалізацію діалогів, завдяки системам нейронного машинного перекладу (Neural Machine Translation, NMT). На відміну від традиційних методів, NMT здатна враховувати культурно-мовні контексти, ідіоматичні вирази та жанрову стилістику, що є критично важливим для збереження ігрової автентичності та передачі емоційної атмосфери в усіх мовних версіях гри[243].

Яскравим прикладом практичного застосування ШІ у створенні динамічних діалогів є проєкт AI Dungeon (https://aidungeon.com/) від компанії Latitude. Ця текстова пригодницька гра функціонує на базі мовних моделей GPT і власної інфраструктури та генерує сюжетні лінії та репліки в режимі реального часу відповідно до дій гравця. Така інтерактивна структура дозволяє формувати унікальний ігровий досвід із високим рівнем адаптивності. Для розробників це означає можливість створення контенту, який реагує на дії користувача, має багатоваріантні розгалуження та забезпечує гнучку локалізацію діалогів.

Інтеграція ШІ в процес сценарного дизайну не лише підвищує продуктивність творчих команд, а й сприяє створенню більш реалістичних, емоційно насичених і сюжетно логічних наративів. У довгостроковій перспективі такі інструменти сприятимуть переходу до процедурно генерованих сценаріїв, де діалоги не лише створюються динамічно, а й адаптуються до стилю гри окремого користувача, що підвищує рівень занурення та персоналізації геймплейного досвіду.

5.2.9 Інструменти ШІ для тестування відеоігор

Процес тестування є невіддільною складовою життєвого циклу розроблення відеоігор і безпосередньо впливає на якість, стабільність та комерційну успішність кінцевого продукту. Традиційні підходи до тестування, засновані на ручній перевірці функціоналу, є надзвичайно трудомісткими, залежними від людського фактора й малоефективними при роботі з великими обсягами ігрового контенту[244]. Саме тому дедалі більшого поширення набувають автоматизовані

[243] Game On: How AI is Revolutionizing Game Localization! URL: https://linkedin.com/pulse/game-how-ai-revolutionizing-localization-midlocalize-wq6of

[244] Трофименко О.Г., Пастернак Ю.Ю., Манаков С.Ю., Лобода Ю.Г. Автоматизація тестування вебсайтів електронної комерції. *Сучасна спеціальна техніка*. 2021. № 2(65). С. 46-59. DOI: https://doi.org/10.36486/mst2411–3816.2021.2(65).5

системи на базі ШІ, які дозволяють значно підвищити продуктивність та точність процесу тестування.

Алгоритми ШІ здатні моделювати тисячі сценаріїв взаємодії в ігровому середовищі у реальному часі, виявляти як технічні, так і логічні дефекти, наприклад, неочікувану поведінку NPC, некоректну взаємодію об'єктів або збої в анімаціях[245]. На відміну від ручного тестування ШІ-тестери можуть неперервно обробляти велику кількість ситуацій за короткий проміжок часу. Це особливо важливо для складних ігор з відкритим світом, де обсяг можливих взаємодій надзвичайно великий.

Прикладами ефективного впровадження подібних рішень є Razer Game Assistant та Razer QA Companion, які частково інтегровані в систему WYVRN[246]. Ці інструменти здійснюють моніторинг ігрового процесу в режимі реального часу, фіксують послідовність дій користувача та автоматично виявляють аномалії. Завдяки цьому розробники мають змогу відстежувати джерела помилок, включно зі складними сценаріями, які важко виявити вручну.

Ще одним перспективним інструментом є AI QA Copilot – хмарний плагін, сумісний із рушіями Unreal Engine та Unity. Він здійснює автоматизовану діагностику коду, виявляє помилки, аналізує продуктивність, створює детальні звіти та надає рекомендації щодо оптимізації. За результатами досліджень[247] цей інструмент забезпечує виявлення на 20-25% більше помилок порівняно з традиційними методами тестування, що дозволяє скоротити час перевірки до 50%, а також зменшує витрати на 40%, що в сукупності суттєво підвищує ефективність QA-процесу.

Крім того, у рамках ігрового рушія Unreal Engine 4 (UE4) активно використовуються поведінкові дерева (Behavior Trees) як засіб для тестування та налагодження дій NPC та ботів. Поведінкові дерева дозволяють створювати складні патерни поведінки, які адаптуються до змін у навколишньому середовищі гри. Це забезпечує перевірку цілісності ігрової логіки та достовірності симуляції віртуальної поведінки.

Інтеграція ШІ в процес тестування відеоігор демонструє вагомі переваги: зниження витрат, підвищення точності, пришвидшення зворотного зв'язку та здатність до масштабованості. Водночас слід враховувати низку технічних і методологічних обмежень. Наприклад, сучасні моделі ШІ ще не здатні повною мірою замінити людське розуміння контексту гри, естетичних параметрів або непередбачуваних сценаріїв, що потребують емпатії та інтерпретації. Надалі розвиток ШІ-систем у сфері геймдеву залежатиме від поєднання автоматизованого аналізу та креативного втручання людини, що забезпечить ефективний і збалансований підхід до забезпечення якості в ігровій індустрії.

---

[245] Трофименко О.Г., Дика А.І., Лобода Ю.Г. Аналіз інструментів тестування вебзастосунків. *Кібербезпека: освіта, наука, техніка.* 2023. № 4(20). С. 62-71. DOI: https://doi.org/10.28925/2663-4023.2023.20.6271

[246] Razer Reveals WYVRN: AI-Powered Gaming Ecosystem Set to Transform Play and Development. URL: https://press.razer.com/company-news/razer-reveals-wyvrn-ai-powered-gaming-ecosystem-for-game-developers/

[247] Razer launches new Wyvrn game dev platform with automated AI bug tester. URL: https://www.theverge.com/news/632121/razer-wyvrn-developer-ai-qa-gamer-copilot-sensa-haptics-thx

5.3 Трансформація геймдев-процесів під впливом штучного інтелекту: практичні аспекти та виклики

Попри значний прогрес у розвитку ШІ, на сьогодні немає універсального інструмента, здатного повноцінно охопити всі етапи розроблення відеоігор. Це зумовлено як високою складністю сучасних ігрових продуктів, так і розмаїттям механік, жанрів та типів контенту, які потребують різних підходів до автоматизації. Сучасні ШІ-інструменти зазвичай спеціалізуються на вирішенні окремих завдань у межах виробничого циклу, що підтверджується практичним досвідом провідних компаній.

Зокрема, сервіси Leonardo.Ai та Adobe Substance 3D активно використовуються для автоматизованої генерації візуального контенту, зокрема текстур, матеріалів і художніх концептів. Це дозволяє суттєво скоротити час створення унікальних ігрових середовищ та швидко адаптувати графіку під стилістичні вимоги конкретного проєкту. Такі технології підвищують гнучкість процесу розроблення і водночас зменшують навантаження на художників, даючи їм змогу зосередитися на креативних завданнях.

Подібно до цього, платформи Meshy.ai та Luma AI використовують генеративні алгоритми для створення фотореалістичних 3D-моделей на основі текстових описів. Цей підхід значно спрощує pipeline створення ігрових об’єктів, що раніше вимагав ручного моделювання. Для генерації анімацій реалістичних рухів у реальному часі застосовуються рішення на кшталт DeepMotion, які аналізують відеодані та синтезують рухи персонажів із правильною кінематикою. У такий спосіб роль дизайнерів змінюється: вони перетворюються на кураторів, які контролюють результати автоматизованих систем, зосереджуючись на естетиці, цілісності стилю та наративній відповідності.

У контексті створення поведінкових моделей неігрових персонажів (NPC) актуальним інструментом залишається Behavior Tree в рушії Unreal Engine 4. Хоча цей інструмент не є повноцінною ШІ-системою, він забезпечує формування складної ієрархічної логіки дій NPC, що дозволяє створювати динамічні реакції на зміну ігрового середовища. Для глибокого машинного навчання агентів, які адаптуються до поведінки гравця, використовується фреймворк Unity ML-Agents, який дає змогу реалізовувати автономні моделі навчання на основі підкріплення (reinforcement learning). Це сприяє побудові більш реалістичних, гнучких і адаптивних моделей поведінки в ігрових середовищах.

Інтеграція зазначених інструментів також має безпосередній вплив на системи оброблення та аналізу ігрових даних. Завдяки машинному аналізу дій гравців розробники можуть отримувати зворотний зв’язок щодо геймплейної привабливості, балансу механік та інтерфейсної зручності, що дозволяє здійснювати адаптацію контенту у реальному часі. Це, своєю чергою, підвищує безпеку та стабільність системи, знижує ризики критичних помилок і запобігає небажаним сценаріям поведінки гравців.

Важливо зазначити, що зазначені технології вже активно застосовуються і в українському геймдев-середовищі, де більшість розробників розглядають ШІ

не як загрозу своїм робочим місцям, а як ефективний інструмент, який покращує якість продукції та оптимізує робочі процеси.

Разом з тим, попри численні переваги, використання ШІ у розробленні відеоігор не є універсальним чи беззаперечним рішенням. Поточні генеративні моделі мають технічні обмеження, зокрема недостатню здатність до узагальнення у складних багатоконтекстних ситуаціях, а також концептуальні межі, пов'язані з творчою природою процесу розроблення. Тому варто поєднувати автоматизовані підходи з людським експертним контролем для забезпечення якості, безпеки та естетичної довершеності кінцевого продукту.

5.4 Складнощі та обмеження використання ШІ у розробленні та тестуванні відеоігор

Генеративні моделі ШІ оперують на основі текстових або зображених підказок, а тому не завжди володіють комплексним розумінням сюжету, ігрової механіки або внутрішнього лору (Game Lore). Це може призводити до нелогічного вмісту, наприклад, NPC у фантазійному світі може демонструвати поведінку, не відповідну встановленим законам всесвіту гри. Тим самим, якість генерованого контенту не завжди відповідає заданим стандартам: нерівномірна складність рівнів, поверхневі сюжетні лінії виникають через обмежений контекст аналізу з боку ШІ. Це може призвести до ситуацій, коли гра здається незбалансованою чи нецікавою.

Хоча ШІ здатен автоматизувати створення тестових сценаріїв і виявляти технічні баги, він не має когнітивного уявлення гравця та "інтуїції", необхідної для розпізнавання тонких помилок. Наприклад, система може пропустити порушення балансу при певній комбінації навичок або неприродну реакцію NPC, що очевидна людині через досвід у тестуванні[248]. Генеративні моделі не здатні запам'ятовувати тривалі події або логічно зв'язані ланцюги дій протягом часу, що зменшує їхню здатність виявляти баги на великих часових проміжках.

Генеративні моделі навчаються на загальнодоступних даних, які не завжди відповідають вимогам геймдизайну або комерційним стандартам. Це може спричинити повторюваність контенту, порушення авторського права (наприклад, через копіювання захищених творів без дозволу), а також створення токсичних або небажаних елементів[249]. Для комерційних продуктів такі юридичні ризики можуть мати серйозні наслідки.

Надмірне використання промислових моделей може призвести до зменшення ролі творчих спеціалістів – художників, сценаристів та дизайнерів ігрових рівнів. Однак, навіть у разі використання ШІ, отримані результати вимагають дослідницького аналізу і доопрацювання (корекція структури, стилістики, емоційного забарвлення), що вилучає частину передбаченої економії часу та ресурсів.

---

[248] Трофименко О.Г., Дика А.І., Лобода Ю.Г. Аналіз інструментів тестування вебзастосунків. *Кібербезпека: освіта, наука, техніка*. 2023. № 4(20). С. 62-71. DOI: https://doi.org/10.28925/2663-4023.2023.20.6271.

[249] Трофименко О. Г. Складнощі та недоліки використання генеративного штучного інтелекту в розробленні та тестуванні ігор. *Інформаційне суспільство: проблеми та перспективи* : матер. X всеукр. наук.-практ. конф. (23 травня 2025). Одеса, 2025. С. 28-31. DOI: https://doi.org/10.32837/11300.29842

Так, сюжет, згенерований ШІ, може бути граматично правильним, але емоційно порожнім, не відповідати загальному тону гри. Генеративний ШІ не здатен повноцінно замінити цю командну динаміку, бо не володіє емоційною, креативною чи стратегічною гнучкістю людини. Навіть найкращі ШІ-моделі залишаються лише зручними інструментами, а не повноцінними учасниками процесу.

Генеративні моделі, зокрема нейронний рендеринг, потребують значних обчислювальних ресурсів. Для інді-студій або малих команд витрати на серверні обчислення можуть суттєво перевищувати бюджет, необхідний на закупівлю або оренду обладнання високого класу[250].

Сучасні методи генерації рівнів (наприклад, у rogue-like жанрі) формуються алгоритмами випадкової комбінації елементів (коридорів, кімнат, ворогів), що забезпечує кількісне різноманіття, але позбавляє глибинного дизайну: сюжетного контексту, емоційної напруги, поступової еволюції через рівень. З огляду на це, гра, хоча й різниться в структурі, але відчувається подібною, оскільки не має стилістично узгодженого наративу чи емоційних акцентів.

Отже, аналіз виявив цілу низку можливих складнощів та обмежень використання ШІ у розробленні та тестуванні відеоігор (рис. 5.3).

Рисунок 5.3. Складнощі та обмеження використання ШІ у GameDev

Вказані проблеми зумовлені глибинними технічними обмеженнями моделей – відсутністю здатності моделювати контекст, когнітивного досвіду, належної кількості релевантних даних, а також неможливістю замінити людську креативність. Тому для забезпечення якісного, безпечного та унікального продукту необхідне збалансоване поєднання ШІ-інструментів та людського контролю, де

[250] Inzoi's 'Smart Zoi' AI system sounds great on paper but seeing it in a live demo didn't exactly wow me. URL: https://www.pcgamer.com/games/life-sim/inzois-smart-zoi-ai-system-sounds-great-on-paper-but-seeing-it-in-a-live-demo-didnt-exactly-wow-me/

генеративні підходи – це лише допоміжні інструменти, що доповнюють, а не замінюють, талановитих розробників.

5.5 Висновки до розділу

Сучасні технології ШІ радикально трансформують всі етапи розроблення комп'ютерних ігор, відкриваючи нові можливості для оптимізації робочих процесів та підвищення якості кінцевого продукту. Інтеграція в ігрову індустрію інструментів на базі ШІ, таких як Meshy.ai, Leonardo.Ai, AIVA, DeepMotion, GitHub Copilot, Replica Studios та AudioCraft, сприяє суттєвому скороченню трудомістких і затратних процесів, таких як рутинне створення ігрових об'єктів, перевірка відповідності логіки геймплею або локалізація інтерфейсу. Як наслідок, спостерігається прискорення виробничих циклів, підвищення стабільності програмного забезпечення та зростання загальної якості кінцевого продукту, що, в свою чергу, позитивно впливає на користувацький досвід і комерційну успішність ігрових проєктів.

Інтеграція ШІ-технологій у розроблення відеоігор відкриває нові горизонти автоматизації, підвищення продуктивності та креативного самовираження. Оптимальним підходом є синергія між інструментами ШІ та творчістю розробників, що дозволяє забезпечити баланс між інноваційністю та унікальністю геймдизайну. Подальші дослідження мають бути зосереджені на адаптивності алгоритмів ШІ до складних ігрових сценаріїв, удосконаленні етичних рамок їх використання та розширенні доступності інструментів для широкого кола розробників.

6 Комп'ютерний зір

6.1 Теоретичні основи комп'ютерного зору

Наразі можливості комп'ютерного зору широко застосовують у розробленні та тестуванні програмного забезпечення. Завдяки здатності автоматично аналізувати візуальні дані, технології комп'ютерного зору значно підвищують ефективність перевірки інтерфейсів користувача, розпізнавання помилок відображення, перевірки відповідності дизайну, а також тестування застосунків у режимі реального часу. У поєднанні з методами машинного навчання комп'ютерний зір дозволяє автоматизувати рутинні етапи тестування, знижуючи навантаження на команду розробників і скорочуючи час виходу продукту на ринок. Його застосування охоплює як традиційні десктопні та мобільні застосунки, так і програмне забезпечення для вбудованих систем, де важлива точність візуального розпізнавання та контроль якості відображення[251].

Комп'ютерний зір (Computer Vision, CV) – це галузь ШІ, спрямована на розроблення алгоритмів та систем, здатних автоматично аналізувати, інтерпретувати та осмислювати візуальну інформацію з цифрових зображень або відео. Завдяки застосуванню методів глибокого навчання та оброблення зображень, сучасні комп'ютерні системи можуть із високою точністю здійснювати виявлення, класифікацію та розпізнавання об'єктів у реальному середовищі[252].

Сучасні технології комп'ютерного зору забезпечують можливість ідентифікації різноманітних об'єктів, облич, жестів і рухів, а також аналізу їхніх просторових взаємозв'язків і прогнозування подальших дій. Зокрема, такі системи реалізують функції розпізнавання об'єктів, класифікації зображень, аналізу динамічних сцен і реконструкції тривимірної структури простору.

Основу комп'ютерного зору становить поєднання методів машинного навчання, цифрового оброблення зображень і геометричного моделювання. Це дає змогу отримувати релевантну інформацію з візуальних даних з метою автоматизації процесів прийняття рішень або розширення когнітивних можливостей людини[253].

Комп'ютерний зір надає комп'ютерам здатність «бачити» та інтерпретувати навколишнє середовище, аналогічно до людського зору, але із використанням цифрових сенсорів і алгоритмів. На відміну від людини, яка формує зорові навички впродовж життя, системи комп'ютерного зору досягають необхідного рівня точності через навчання на великих обсягах даних, що дозволяє здійснювати оброблення інформації швидко та ефективно.

---

[251] Minimol Anil Job. Automating and Optimizing Software Testing using Artificial Intelligence Techniques. *International Journal of Advanced Computer Science and Applications (IJACSA).* 2021. Vol. 12, No. 5. P. 594-602. DOI: http://dx.doi.org/10.14569/IJACSA.2021.0120571.

[252] Research Areas in Computer Vision: Trends and Challenges. URL: https://surl.lu/dkcbei

[253] What's the scope of computer vision in AI? URL: https://surl.lu/cqwyou

Сьогодні галузь комп’ютерного зору поєднує різні алгоритми та архітектури ШІ, такі як згорткові нейронні мережі (Convolutional Neural Networks, CNN) та трансформатори зору (vision transformers, ViT), для оброблення, аналізу та вилучення відповідних шаблонів із візуальних даних[254]:

– генеративний ШІ;
– периферійний комп’ютерний зір;
– комп’ютерний зір у реальному часі;
– доповнена та віртуальна реальності;
– 3D-візуалізація;
– поодиноке навчання.

Генеративний ШІ – це клас ШІ-систем, здатних автономно створювати новий контент, зокрема текст, зображення, відео, аудіо або програмний код, у відповідь на запити користувачів. Такі системи ґрунтуються на високорівневих моделях машинного навчання, зокрема на архітектурах глибокого навчання, які моделюють когнітивні функції, подібні до процесів навчання та прийняття рішень у людському мозку.

Генеративні моделі функціонують шляхом виявлення, кодування та узагальнення прихованих закономірностей у великих масивах даних. На основі вивчених статистичних зв’язків вони формують відповідний вихідний контент, релевантний до запиту, сформульованого природною мовою. Здатність до генерації нових даних на основі наявного контексту робить такі системи потужним інструментом для автоматизованого оброблення інформації, креативних завдань і взаємодії людини з комп’ютером на семантичному рівні.

Периферійний комп’ютерний зір. Відомо, що комп’ютерний зір – це процес навчання комп’ютерів аналізувати візуальні дані подібно до людських. Але, на відміну від людського ока, комп’ютерний зір може знаходити закономірності у візуальних даних, які люди не можуть виявити, що робить його цінним інструментом у багатьох галузях. З іншого боку, периферійні обчислення – це парадигма розподілених обчислень, яка стосується переміщення обчислювального оброблення ближче до джерела даних. Це означає, що дані обробляються безпосередньо там, де вони генеруються, уникаючи надсилання великих обсягів даних у хмару для оброблення, аналізу та зберігання. Ця техніка удосконалює комп’ютерний зір, шляхом швидкого оброблення та аналізу на пристроях (як-от: камери, датчики, мобільні телефони) без залежності від хмарних серверів. Результатом є ухвалення рішень у режимі реального часу, підвищена безпека, знижені вимоги до пропускної здатності та менша затримка.

Ключовою перевагою периферійного комп’ютерного зору є його здатність працювати в режимі реального часу, що дозволяє його застосовувати у сфері безпеки, спостереження та робототехніки. Крім того, позаяк він працює локально, такий комп’ютерний зір забезпечує високий ступінь конфіденційності та безпеки, оскільки немає потреби надсилати конфіденційні дані на віддалений сервер, де вони більш вразливі до зовнішнього втручання. Тому периферійні пристрої,

[254] Computer Vision Tasks (Comprehensive 2025 Guide). URL: https://viso.ai/deep-learning/computer-vision-tasks/

наприклад, сервери або комп'ютери, підключаються до камер та запускають моделі ШІ в програмах реального часу[255].

Комп'ютерний зір у реальному часі (Real-Time Computer Vision, RCV) – це галузь ШІ та машинного навчання. Він «бачить» та розуміє вміст цифрових зображень, таких як фотографії та відео. Такий комп'ютерний зір дозволяє комп'ютерам миттєво аналізувати візуальні дані та реагувати на них. На відміну від пакетного оброблення, яке має справу з накопиченими даними, системи реального часу постійно обробляють потоки даних. Ключові показники продуктивності для таких систем:

– *затримка* – час, витрачений від введення до практичного виведення;

– *пропускна здатність* – обсяг даних, оброблених за одиницю часу;

– *точність* – точність ідентифікації та аналізу об'єктів у динамічних середовищах[256].

Доповнена та віртуальна реальності (Augmented reality, AR and virtual reality, VR) змінюють нашу взаємодію із зовнішнім світом. Комп'ютерний зір є головним рушієм, який керує плавним переходом між віртуальним та реальним світами. У доповненій реальності комп'ютерний зір використовується для виявлення та розпізнавання об'єктів у візуальних даних, а також для відстеження об'єктів для розуміння руху, підрахунку об'єктів. У віртуальній реальності комп'ютерний зір використовується для оцінки поз рук та відстеження жестів та розпізнавання погляду[257].

3D-візуалізація. Досягнення в моделюванні комп'ютерного зору допомагають експертам аналізувати 3D-зображення, точно фіксуючи інформацію про глибину та відстань. Системи 3D-комп'ютерного зору витягують, обробляють та аналізують 2D-візуальні дані для створення 3D-моделей за допомогою різних алгоритмів і методів збору даних, які дозволяють моделям комп'ютерного зору визначати розміри, контури та просторові відносини об'єктів у заданому візуальному середовищі[258].

Поодиноке навчання – це піднапрямок машинного навчання, який стосується класифікації нових даних, коли є лише кілька навчальних вибірок з контрольованою інформацією. Поодиноке навчання можна використовувати в завданнях комп'ютерного зору, оскільки модель комп'ютерного зору може доволі добре працювати з відносно невеликою кількістю навчальних вибірок[259].

Отже, галузь комп'ютерного зору стрімко розвивається, завдяки впровадженню передових алгоритмів ШІ, що забезпечує ефективне оброблення та аналіз візуальної інформації. Це свідчить про багатогранність наявних підходів у сфері комп'ютерного зору.

Комп'ютерний зір стрімко трансформує майбутнє технологій, оскільки дедалі більше галузей інтегрують його інструменти у власні процеси, досліджуючи широкі можливості його застосування. Значний прогрес у сфері ШІ, зокрема в

---

[255] Earney Sh. What is Edge Computer Vision, and How Does it Work? URL: https://surl.li/ldnyrd

[256] Real-Time Vision: Accelerating Innovations in Computer Processing. URL: https://surl.li/sybliy

[257] Computer Vision in AR and VR – The Complete Guide. URL: https://surl.li/dcsgse

[258] 3D Computer Vision: A Comprehensive Guide. URL: https://viso.ai/computer-vision/3d-computer-vision/

[259] Lyashenko V. Understanding Few-Shot Learning in Computer Vision: What You Need to Know. URL: https://surl.li/luvrhj

галузях машинного навчання та глибокого навчання, суттєво підвищив ефективність і функціональність систем комп'ютерного зору.

Окрему увагу слід приділити зростанню попиту на технології розпізнавання облич, які стають невіддільною частиною сучасних систем безпеки, аналітики поведінки користувачів, а також механізмів персоналізації обслуговування. Зокрема, у сферах роздрібної торгівлі та охорони здоров'я фіксується значне зростання впровадження рішень на основі комп'ютерного зору, що зумовлено потребою в безконтактній взаємодії, підвищенні точності обслуговування, ефективності процесів та створенні індивідуалізованого клієнтського досвіду.

У свою чергу, поширення технологій периферійних (edge) обчислень і стрімке зростання кількості пристроїв, що належать до Інтернету речей (IoT), відкриває нові горизонти для інтеграції комп'ютерного зору в децентралізовані системи. Це дозволяє обробляти візуальну інформацію безпосередньо на пристроях у режимі реального часу, знижуючи затримки, зменшуючи навантаження на мережу та покращуючи безпеку даних.

Згідно з актуальними ринковими прогнозами, глобальний ринок комп'ютерного зору оцінюється у 30,25 мільярда доларів США станом на 2025 рік і, за очікуваннями, досягне 50,98 мільярдів доларів США до 2030 року[260]. Прогнозується, що середньорічний темп зростання у період з 2025 по 2030 рік становитиме 19,8%, що свідчить про стабільне зростання інтересу до цієї технології та її комерціалізації у найближчі роки. Така динаміка підтверджує стратегічну важливість комп'ютерного зору як ключового напряму в інноваційній екосистемі майбутнього (рис. 6.1)[261].

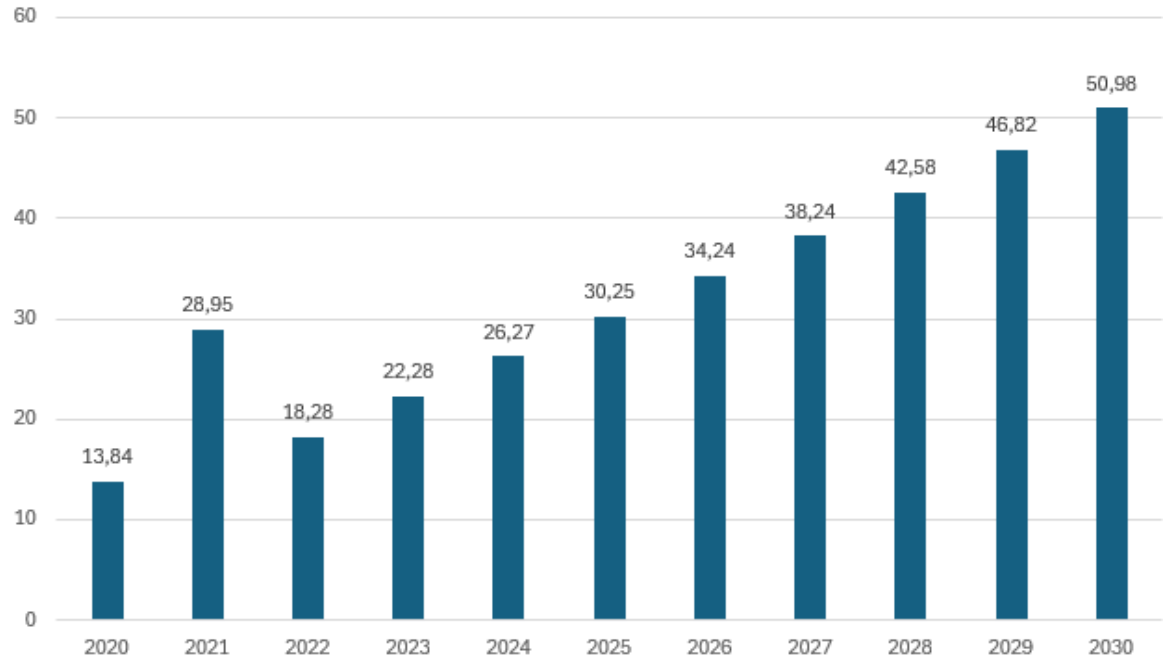


Рисунок 6.1. Розмір світового ринку комп'ютерного зору

[260] Computer Vision Market Summary. URL: https://surl.lu/geeazf
[261] Computer Vision – Worldwide. URL: https://surl.lu/qmmvkd

Зростання ринку комп'ютерного зору зумовлене низкою ключових факторів, серед яких – ріст попиту на автоматизацію в різних галузях, стрімкий розвиток технологій ШІ та машинного навчання, удосконалення апаратного забезпечення й сенсорних систем, а також активне впровадження автономних транспортних засобів. Технології комп'ютерного зору знаходять широке застосування в секторах промислового виробництва, роздрібної торгівлі, автомобілебудування та охорони здоров'я, зокрема для розв'язання завдань контролю якості, управління складськими запасами, медичної візуалізації тощо.

Інтеграція комп'ютерного зору в автоматизовані системи дозволяє підвищити продуктивність, забезпечити високу точність операцій та зменшити витрати на виконання рутинних процесів. Тому за умов зростання використання рішень на базі ШІ у багатьох галузях, комп'ютерний зір відіграє важливу роль для підвищення ефективності бізнес-процесів, оптимізації ухвалення рішень та забезпечення конкурентоспроможності підприємств.

6.2 Основні завдання комп'ютерного зору

Попри активне впровадження технологій комп'ютерного зору в різноманітні галузі, ефективне функціонування таких систем ґрунтується на розв'язанні низки базових завдань, серед яких:

1) класифікація об'єктів;
2) виявлення або детектування об'єктів;
3) сегментація об'єктів;
4) розпізнавання об'єктів.

Зазначені завдання відіграють ключову роль у забезпеченні точного аналізу візуальних даних та їх інтерпретації ШІ.

1) Класифікація об'єктів є одним із базових процесів комп'ютерного зору та передбачає визначення загального класу або категорії зображення залежно від його змісту (рис. 6.2). Основною метою класифікації зображень є віднесення їх до одного з попередньо визначених класів або категорій. Це завдання зазвичай реалізується за допомогою алгоритмів машинного навчання, зокрема CNN, які відзначаються високою здатністю до вилучення релевантних ознак із візуальних даних.

Проте ефективність класифікації може залежати від низки факторів, які знижують точність розпізнавання. Оскільки комп'ютерний зір часто працює з неоднозначними або неповними вхідними даними, навіть добре навчена модель може допускати помилки класифікації. Серед основних чинників, що впливають на результати:

– *оклюзія* – ситуація, коли цільовий об'єкт частково перекритий іншими елементами сцени. Наприклад, якщо кішка частково прихована за рослинністю, навіть при видимості її голови модель може не розпізнати об'єкт через обмежену кількість візуальної інформації;

Рисунок 6.2. Класифікація об’єктів

– *фоновий шум* – наявність складних текстур або подібності кольорів між об’єктом і фоном може ускладнити процес розпізнавання. Наприклад, червоне яблуко на фоні стільниці схожого кольору або транспортний засіб, який повільно рухається серед інших, – усе це створює труднощі для точного визначення меж і класу об’єкта.

Успішна реалізація класифікації вимагає врахування як технічних характеристик моделей, так і контексту зображень, із якими вони працюють[262].

2) Виявлення або детектування об’єктів ґрунтується на застосуванні методів класифікації зображень з метою не лише розпізнавання об’єктів, а й визначення їхнього просторового розташування в межах зображення або відеопотоку. Цей підхід дає змогу точно ідентифікувати об’єкти, встановлювати їхні координати та позначати відповідними класовими мітками.

На практиці детектування об’єктів дозволяє системі виконувати підрахунок об’єктів у кадрі, що є важливим для задач аналізу середовища, моніторингу або автономного управління. Наприклад, алгоритм виявлення може одночасно розпізнати та класифікувати кота і собаку на зображенні, точно визначивши їхні межі та класи (рис. 6.3).

Виявлення об’єктів підвищує точність систем ШІ завдяки їхній здатності отримувати доступ до більших обсягів інформації. Якщо система ШІ була навчена виявленню об’єктів, вона зможе вивчати нові функції та можливості з даних. Це означає, що її не потрібно перепроєктувати для розроблення нових можливостей. Завдяки виявленню об’єктів системи ШІ можуть обробляти великі та складні набори даних, які були б нездійсненними або занадто трудомісткими для традиційних методів машинного навчання.

[262]Image Classification vs. Object Detection: Everything You Need to Know. URL: https://surl.lt/vyvmrr

Рисунок 6.3. Виявлення об’єктів

При виявленні об’єктів потрібно враховувати деякі властивості зображень:

– *варіація точки зору* – різні перспективи можуть надати об’єкту абсолютно новий вигляд. Саме тому навчальні набори даних повинні мати зображення з різних перспектив;

– *деформація* – багато об’єктів мають надзвичайні можливості деформації та не є твердими тілами. Детектор об’єктів може не ідентифікувати людей на фотографіях, якщо його навчили ідентифікувати лише об’єкти, які сиділи, стояли або йшли. Це пояснюється тим, що ознаки на цих зображеннях не відповідають ознакам, які детектор об’єктів навчили ідентифікувати під час навчання;

– *різні умови освітлення* призводять до появи різних відтінків на об’єктах. За таких різноманітних умов освітлення зображення пішохода виглядає по-різному. Це впливає на надійну здатність детектора виявляти об’єкти[263].

3) Сегментація об’єкта на зображенні – це ключовий процес у комп’ютерному зорі, який розділяє зображення на змістовні сегменти. Вона призначає мітки пікселям, спрощує складні візуальні дані та дозволяє комп’ютерам ефективніше розуміти та аналізувати зображення. Сегментація спрощує подання зображення на щось більш значуще та просте для аналізу. Усі пікселі, що належать до однієї категорії, мають спільну мітку, призначену їм.

Є два способи сегментації:

– *подібність* – сегменти формуються шляхом виявлення подібності між пікселями зображення. На даному способі базуються алгоритми машинного навчання (наприклад, кластеризація);

– *розривність* – тут сегменти формуються на основі зміни значень інтенсивності пікселів на зображенні. Цей спосіб використовується методами

[263] Image Classification vs. Object Detection: Everything You Need to Know. URL: https://mindy-support.com/news-post/image-classification-vs-object-detection-everything-you-need-to-know/

виявлення ліній, точок та країв для отримання проміжних результатів сегментації, які можна обробити для отримання кінцевого сегментованого зображення.

Для розуміння режимів сегментації потрібно визначити поняття «об'єктів» і «фонів». Об'єкти – це ідентифіковані сутності на зображенні, які можна відрізнити один від одного, присвоївши їм унікальні ідентифікатори, тоді як фон стосується частин зображення, які неможливо порахувати, як-от: небо, водойми та інші подібні елементи.

Розрізняючи об'єкти та фони, можна визначити різні режими сегментації зображень та їх застосування (табл. 6.1)[264].

Таблиця 6.1 – Типи сегментації зображень

Тип сегментації	Визначення	Характеристики	Застосування
Екземплярна	Виявляє та сегментує кожен об'єкт як окрему сутність з унікальними межами	– ідентифікує окремі об'єкти (наприклад, автомобілі, людей); – розділює об'єкти, що перекриваються; – не вимагає інформації про клас	Відстеження об'єктів, AR/VR, розширене виявлення об'єктів
Семантична	Позначає кожен піксель категорією класу (наприклад, «небо», «дорога», «автомобіль»)	– призначає щільні мітки класів кожному пікселю; – не розрізняє різні екземпляри одного класу; – фон та об'єкти згруповані за класом	Аналіз дороги, медична візуалізація, розуміння місця події
Паноптична	Поєднує семантичну та екземплярну сегментацію для детального маркування	– забезпечує попіксельне маркування за класами та екземплярами; – розрізняє об'єкти та їхні екземпляри; – пропонує більш деталізовану та найякіснішу інформацію	Автономні транспортні засоби, робототехніка, інтерактивні системи ШІ

Екземплярна сегментація – це тип сегментації зображення, який поєднує виявлення та сегментацію кожного об'єкта на зображенні. Вона схожа на виявлення об'єктів, але з додатковим завданням сегментації меж об'єкта. Алгоритм не має уявлення про клас області, але він розділяє об'єкти, які перекриваються. Сегментація екземплярів корисна в застосунках, де потрібно ідентифікувати та відстежувати окремі об'єкти (рис. 6.4).

Семантична сегментація – це сегментація зображення, яка передбачає позначення кожного пікселя на зображенні відповідною міткою класу без урахування іншої інформації чи контексту (рис. 6.5). Мета полягає в тому, щоб призначити мітку кожному пікселю на зображенні, що забезпечує щільне маркування зображення. Алгоритм приймає зображення як вхідні дані та генерує карту сегментації, де значення пікселя (0, 1, ..., 255) зображення перетворюється на мітки класів (0, 1, ..., *n*). Це корисно в застосунках, де важлива ідентифікація різних класів об'єктів.

[264] Guide to Image Segmentation in Computer Vision: Best Practices. URL: https://encord.com/blog/image-segmentation-for-computer-vision-best-practice-guide/

Рисунок 6.4. Екземплярна сегментація

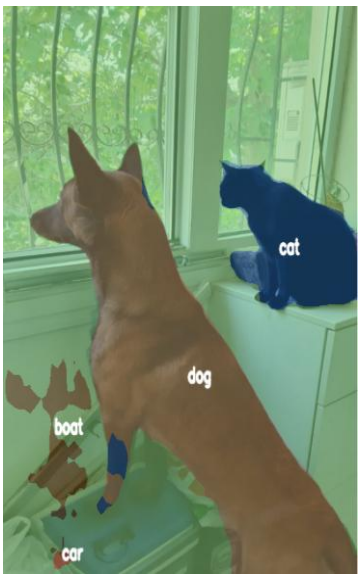


Рисунок 6.5. Семантична сегментація

Паноптична сегментація – це поєднання семантичної та екземплярної сегментацій (рис. 6.6). Вона передбачає позначення кожного пікселя міткою класу та ідентифікацію кожного екземпляра об’єкта на зображенні. Цей режим сегментації зображення забезпечує максимальну кількість високоякісної детальної інформації від алгоритмів машинного навчання. Він корисний у застосунках, де модель комп’ютерного зору має виявляти та взаємодіяти з різними об’єктами у своєму середовищі.

Рисунок 6.6. Паноптична сегментація

Отже, сегментація об’єктів на зображеннях є важливим завданням аналізу в комп’ютерному зорі, що дозволяє виділяти значущі області та полегшує машинне сприйняття складних візуальних сцен.

4) Розпізнавання об’єктів – це завдання, яке використовується для виявлення закономірностей у даних. У комп’ютерному зорі розпізнавання використовується для визначення розташування об’єктів на зображеннях та відео. Для виконання задач розпізнавання важливими є аспекти цифрового оброблення

зображень, зокрема: покращення якості зображень, визначення їхніх характеристик, спектральний аналіз багатовимірних сигналів, а також безпосереднє розпізнавання зображень[265].

Обробка зображень – це методи та алгоритми, які використовуються із зображеннями з метою їх покращення або вилучення з них потрібної інформації. Це важливо для комп'ютерного зору, де попередня обробка зображення може значно підвищити результат розпізнавання. Традиційними методами оброблення зображень є: регулювання контрастності та яскравості зображення, вирівнювання гістограми, зменшення шуму, усунення розмиття, виявлення країв тощо.

Розпізнавання зображень за допомогою методів машинного навчання ґрунтується на здатності алгоритмів виявляти приховані закономірності в даних. Одним із найефективніших підходів є глибоке навчання, яке використовує багаторівневі нейронні мережі для автоматичного вилучення інформативних ознак із вхідних зображень[266].

Розпізнавання об'єктів є ключовим завданням комп'ютерного зору, що полягає в ідентифікації об'єктів на зображенні та визначенні їхніх меж. Досягнення високої точності в цьому процесі значною мірою залежить від етапу попередньої оброблення зображень, яке забезпечує нормалізацію, фільтрацію шумів, змінення розмірів та інші трансформації. Сучасні методи глибокого навчання демонструють високі результати завдяки здатності моделей вивчати великі обсяги даних і адаптуватися до складних варіацій у візуальній інформації.

У контексті комп'ютерного зору основними задачами є класифікація, детекція (виявлення), сегментація та розпізнавання об'єктів. Кожна з них виконує окрему функцію. Сукупне використання цих підходів дає змогу системам штучного інтелекту точно інтерпретувати візуальні дані. Реалізація зазначених задач досягається за допомогою як традиційних методів оброблення зображень, так і сучасних технологій машинного та глибокого навчання.

6.3 Методи комп'ютерного зору

6.3.1 Класичні методи комп'ютерного зору

Класичні методи комп'ютерного зору є основою багатьох алгоритмів аналізу зображень і відео. Вони використовувалися ще до широкого впровадження глибокого навчання. Ці методи не потребують великих обсягів навчальних даних і зазвичай базуються на математичних і геометричних підходах до оброблення візуальної інформації. Вони дозволяють розв'язувати задачі, пов'язані з попереднім обробленням зображень, виявленням об'єктів, визначенням ключових точок, аналізом кольору тощо. Однією з ключових особливостей таких методів є те, що вони покладаються на аналітичні ознаки зображення: градієнти, контури,

[265] Логінова Н.І. Використання мови програмування Python для вирішення завдань розпізнавання образів. *Інформаційне суспільство: проблеми та перспективи:* матер. VIII Всеукр. наук.-практ. конф. (м. Одеса, 2 червня 2023 р.). С. 56-60. https://doi.org/10.32837/ 11300.25263

[266] Gaudenz Boesch. Image Recognition: The Basics and Use Cases. URL: https://viso.ai/computer-vision/image-recognition/

кольорові характеристики, текстуру, геометричні особливості тощо. Такі методи добре працюють у контрольованих умовах, де середовище та об'єкти не надто змінюються.

До поширених напрямів класичних методів комп'ютерного зору належать:

1) *аналіз світла та кольору*, зокрема побудова гістограм кольорів, робота з різними колірними просторами (RGB, HSV тощо), що дає змогу виділяти об'єкти на основі їхніх кольорових характеристик;

2) *детектори країв* (наприклад, фільтри Sobel або Canny) і виявлення контурів, які дозволяють локалізувати межі об'єктів;

3) *фільтрація зображень* використовується для покращення якості та виділення важливих структур;

4) *виявлення особливих точок і побудова дескрипторів* (наприклад, SIFT, SURF, ORB), які дозволяють ідентифікувати унікальні локальні області на зображенні для задач розпізнавання, зіставлення або трекінгу;

5) *хаф-перетворення* (Hough Transform) є ефективним інструментом для виявлення геометричних фігур (прямих, кіл тощо) у зображенні;

6) *ректифікація зображень* – виправлення геометричних викривлень для отримання більш точних вимірювань та перспективного відображення;

7) *стиснення зображень і відеоданих* – дозволяє оптимізувати зберігання та передавання візуальної інформації без суттєвої втрати якості.

Зручним і потужним інструментом для реалізації класичних методів комп'ютерного зору є мова програмування Python, завдяки її гнучкості та широкому набору спеціалізованих бібліотек[267]. Зокрема, бібліотека OpenCV (Open Source Computer Vision Library) надає доступ до більшості зазначених алгоритмів, дозволяючи ефективно працювати із зображеннями та відео. Завдяки зручному інтерфейсу та високій продуктивності OpenCV дозволяє реалізовувати різні алгоритми комп'ютерного зору. Бібліотека підтримує роботу з відео в реальному часі, що робить її зручною для створення застосунків зі зчитуванням потокового відео з камер. Завдяки кросплатформеності, високій продуктивності та підтримці різних мов програмування OpenCV стала однією з найпопулярніших бібліотек у сфері комп'ютерного зору, що пояснюється відкритим вихідним кодом, активною спільнотою та великою кількістю навчальних матеріалів[268].

1) Аналіз світла та кольору

Кольори відіграють життєво важливу роль у тому, як люди сприймають світ, допомагають їм розпізнавати, розуміти та взаємодіяти з навколишнім середовищем. Однак, на відміну від людини, машини не бачать кольори – вони обробляють їх як дані, перетворюючи відтінки на числові значення. Здатність розпізнавати та інтерпретувати кольори допомагає подолати розрив між людським зором та комп'ютерним зором.

Найефективнішим методом пошуку об'єкта вважається визначення його за кольором. Це робиться за допомогою маски кольору. Кольорова маска при накладанні на вихідне зображення дозволяє відфільтрувати кольори відповідно до

---

[267] Логінова Н.І., Толокнов А.А. Візуалізація даних у Python. *Європейські орієнтири розвитку України: науково-практичний вимір в умовах воєнних викликів* : матер. міжнар.наук.-практ. конф. (Одеса, 26 квітня 2024 р.) Одеса: Фенікс, 2024. С. 854-856.

[268] OpenCV is the world's biggest computer vision library. URL: https://opencv.org

вибраного діапазону кольорів. Області зображення, які не відповідають вказаному діапазону кольорів, не відображатимуться. Після застосування колірної маски ці області будуть пофарбовані в чорний колір, а все, що відповідає колірному діапазону, – у білий колір. Це дуже полегшує завдання пошуку об'єктів. Отже, добираючи потрібний діапазон кольорів, можна вивести на екран лише шуканий об'єкт або групу об'єктів. На рис. 6.7 показаний код, який знаходить за кольором собаку та кота (рис. 6.8).

```
image = cv2.imread("cat_dog.jpg")
small_image = cv2.resize(image,  dsize: (0, 0), fx=0.5, fy=0.5)
cv2.imshow( winname: "Original", small_image)

low_color = (0, 0, 0)
high_color = (175, 255, 55)
only_object=cv2.inRange(small_image,low_color,high_color)

cv2.imshow( winname: 'only_object', only_object)
```

Рисунок 6.7. Фрагмент код пошуку об'єктів за кольором

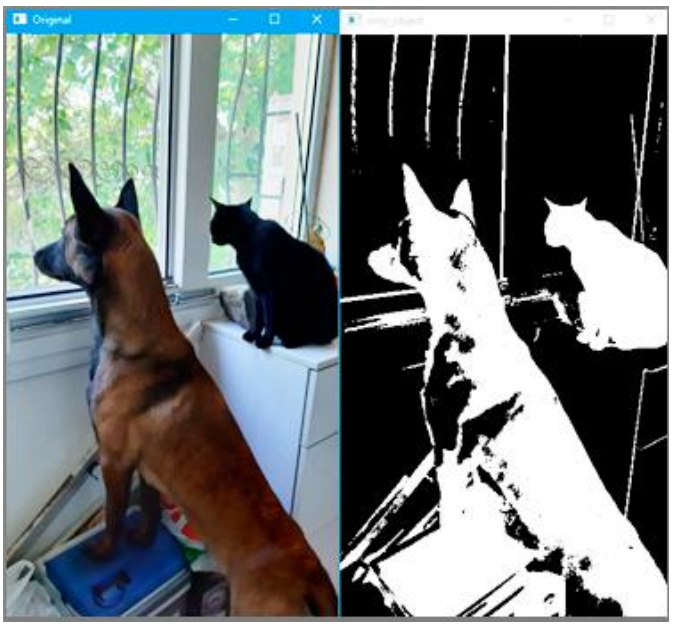


Рисунок 6.8. Результат пошуку об'єктів за кольором

Наведений приклад із пошуком об'єктів за кольором показав, що не завжди вдається виділити лише необхідний об'єкт. Часто разом з об'єктом на зображенні виділяється безліч зайвих точок, так званні шуми, що ускладнює виконання подальшого аналізу зображення. Тому треба максимально позбавитися шумів, визначивши причини їх виникнення. Для цього розглянемо докладніше поняття колірного простору.

RGB-простір є тривимірним кубом, у якому кожен колір задається координатами трьох базових компонентів: червоного, зеленого та синього (рис. 6.9).

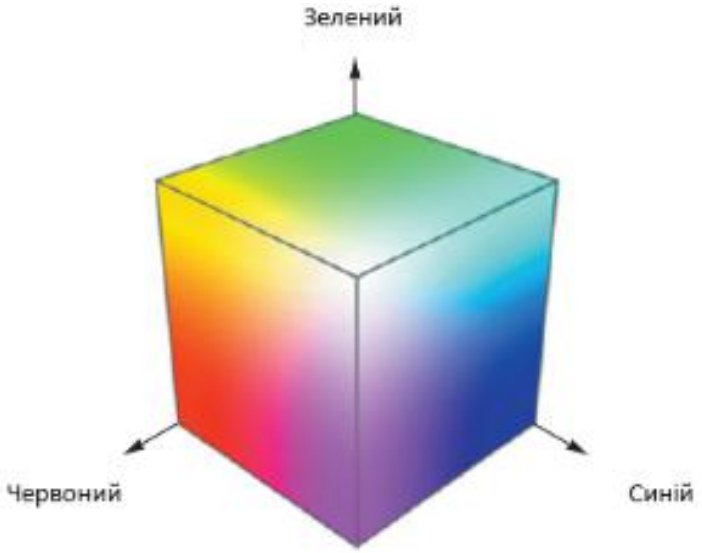


Рисунок 6.9. RGB-простір

Функція *cv2.inRange()* дозволяє виділяти кольори в межах певного діапазону, що утворює в цьому просторі прямокутний паралелепіпед (рис. 6.10). Однак такий підхід не завжди є точним, оскільки через фіксовану орієнтацію граней паралелепіпеда у виділену область можуть потрапляти небажані кольори, які частково відповідають критеріям, але не належать до шуканого об'єкта.

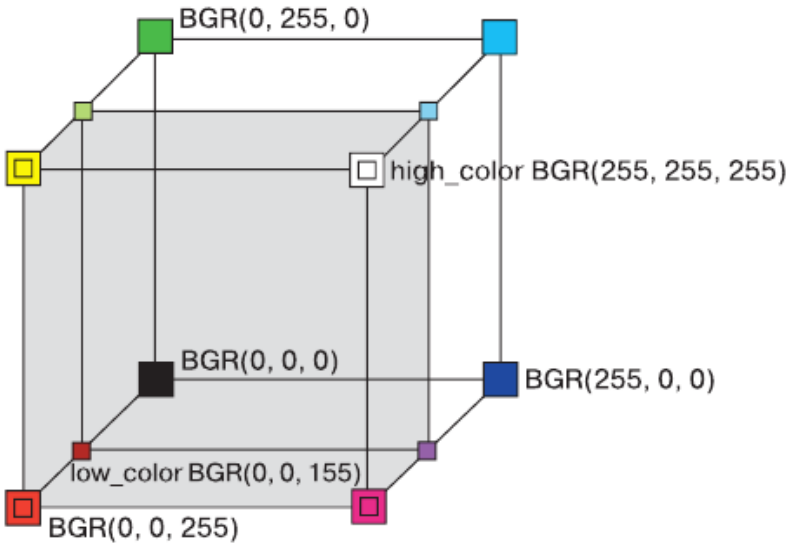


Рисунок 6.10. RGB-паралелепіпед

Для точнішого виділення об'єктів доцільно використовувати HSV-простір (рис. 6.11), який описує колір через колірний тон (Hue), насиченість (Saturation) і яскравість (Value або Brightness). Це дозволяє більш гнучко налаштовувати діапазони, враховувати не лише колір, а й ступінь його насиченості та яскравість. HSV-модель краще відображає особливості людського сприйняття кольору, тому робота з кольорами у цьому просторі є ефективною, особливо коли потрібно ізолювати відтінки, близькі за кольором, але різні за світловими характеристиками.

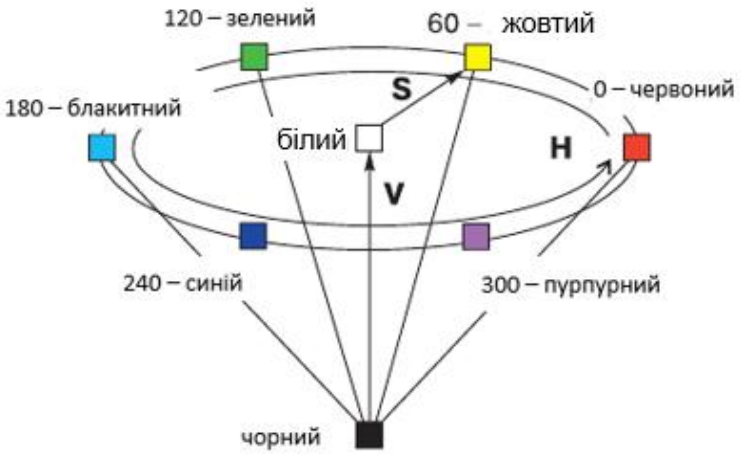


Рисунок 6.11. HSV-модель

Завдяки функції *cv2.cvtColor()* зображення легко переводиться з RGB у HSV-простір, а далі можна застосовувати маскування за вибраним діапазоном значень HSV, що значно покращує якість виділення кольорових об'єктів.

Фрагмент коду програми, який перетворює зображення на HSV-простір, виділяє кольори в діапазоні потрібних відтінків та створює маску, що застосовується до зображення, наведено на рис. 6.12, а результат – на рис. 6.13.

```
hsv = cv2.cvtColor(small_image, cv2.COLOR_BGR2HSV)

low_color = (0, 0, 0)
high_color = (212, 255, 65)

mask = cv2.inRange(hsv, low_color, high_color)
cv2.imshow( winname: "Object", mask)
```

Рисунок 6.12. Фрагмент коду вибір об'єктів за допомогою HSV-моделі

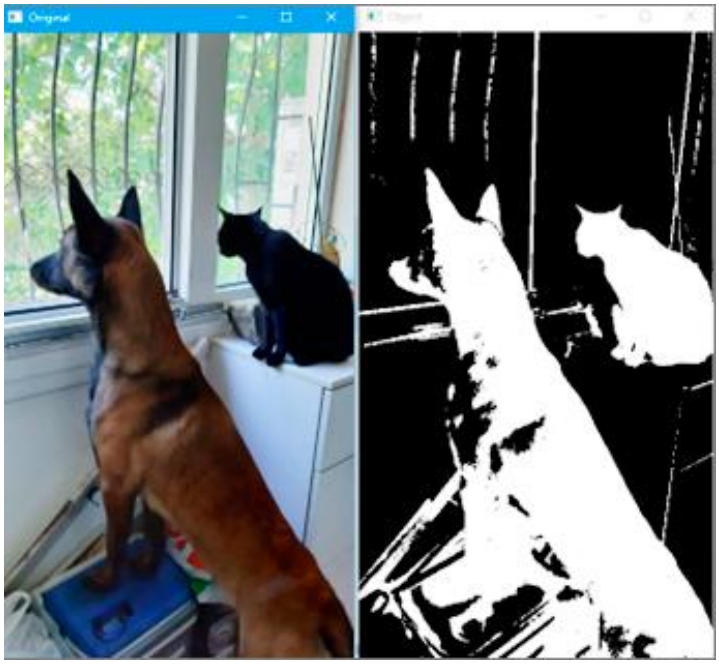


Рисунок 6.13. Результат виділення

Змінення значення *low_color* і *high_color* може краще підібрати відтінки об'єктів, які потрібно виділити. Але для більш точного виділення, краще використати моделі глибокого навчання, які будуть розглянути далі.

У комп'ютерному зорі використовують *гістограми кольорів* для аналізу та опису колірного вмісту зображення. Вони показують, як часто зустрічаються певні значення кольору або яскравості в усьому зображенні або його частині [269].

Комп'ютери переглядають зображення у тривимірному масиві чисел, який називають масивом зображень. Число задає пікселі, а три виміри відповідають основним кольорам: червоному, синьому та зеленому. Різні значення пікселів складають інтенсивність кольору, контрастність та яскравість і формують зображення, яке ми бачимо. Багато програм для оброблення зображень покладаються на аналіз цих значень пікселів. Гістограма кольору – це популярний метод візуалізації побудови зображення, який відображає кількість та візуалізує появу кожного значення пікселя на зображенні. Більша кількість означає велику кількість інтенсивності пікселів і навпаки. Гістограма кольору допомагає аналізувати елементи зображення, як-от: інтенсивність кольору, контрастність та тіні.

Зображення складається з пікселів різних значень. Гістограма – це таблиця, яка показує кількість пікселів, які мають задане значення на зображенні. Так, гістограма зображення з градаціями сірого матиме 256 записів (entries або бункерів, контейнерів – bin). Bin 0 дає кількість пікселів на зображенні, які мають значення 0, bin 1 – кількість пікселів, які мають значення 1, і так далі. Тому, якщо підрахувати всі записи гістограми, можна отримати загальну кількість пікселів. Гістограми також можуть бути нормалізовані так, щоб сума бункерів дорівнювала 1. У цьому разі кожен бункер дає відсоток пікселів, які мають це конкретне значення на зображенні[270].

Обчислення гістограми в Python за допомогою OpenCV здійснюється за допомогою функції *cv::calcHist*, яка дозволяє обчислити гістограму декількох каналів зображення будь-якого типу і діапазону значень пікселів (рис. 6.14).

Гістограма кольорів зображення собаки та кота (рис. 6.15).

На графіку видно, що найбільші значення гістограми (піки) припадають на ліву частину (низькі значення інтенсивності), що свідчить про велику кількість темних або затемнених пікселів в усіх трьох каналах (Red, Green, Blue). Усі три канали мають свої піки в різних діапазонах, що свідчить про кольорове зображення з різними відтінками. Зелений і синій мають вищі значення, ніж червоний, отже, у зображенні переважають холодні відтінки. У правій частині гістограми (високі значення інтенсивності) значення майже нульові для всіх каналів, що вказує на відсутність або мінімальну присутність дуже світлих (білих) пікселів. Гістограми кожного з каналів займають широкий діапазон значень, тобто в зображенні є велика колірна варіативність, і воно не є одноманітним або сірим.

---

[269] Фургала Ю., Вельгош А., Вельгош С., Русин Б. Використання гістограм кольору для ідентифікації об'єктів при масштабуванні та обертанні зображень *Електроніка та інформаційні технології*. 2020. Вип. 13. С. 28–37. DOI: https://doi.org/10.30970/eli.13.3

[270] Розпізнавання образів та обробка зображень: метод. вказівки до виконання лабораторних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення». Чернігів: НУ «Чернігівська політехніка», 2021. 146 с. URL: https://surl.li/qfczfa

```
# Розділення каналів
channels = cv2.split(small_image)
colors = ('b', 'g', 'r')
hist_img = np.full( shape: (400, 512, 3),  fill_value: 255, dtype=np.uint8)

# Побудова гістограми для кожного каналу
for chan, color in zip(channels, colors):
    hist = cv2.calcHist( images: [chan],  channels: [0],  mask: None,  histSize: [256],  ranges: [0, 256])
    hist = cv2.normalize(hist, hist).flatten()

    bin_width = int(hist_img.shape[1] / 256)

    for i in range(1, 256):
        cv2.line(hist_img,
                 pt1: (bin_width * (i - 1), 400 - int(hist[i - 1] * 400)),
                 pt2: (bin_width * i, 400 - int(hist[i] * 400)),
                 color: (255 if color == 'b' else 0,
                  255 if color == 'g' else 0,
                  255 if color == 'r' else 0),
                 thickness: 1)
```

Рисунок 6.14. Фрагмент коду обчислення гістограми зображення

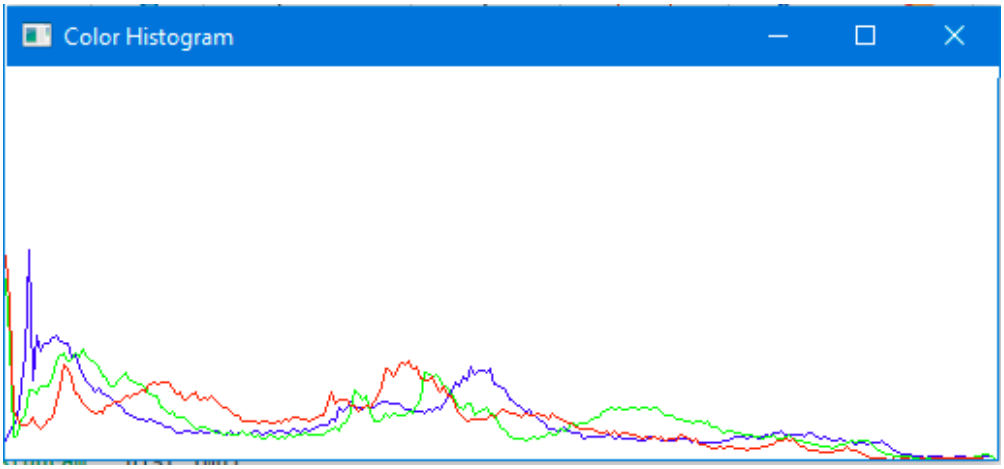


Рисунок 6.15. Гістограма зображення

Отже, зображення є темним, з насиченими кольорами, з переважанням синьо-зелених відтінків. Це може бути нічне або затінене зображення з різноманітними кольоровими об'єктами.

2) Детектори країв

Детектори країв – це алгоритми, які визначають різкі зміни яскравості на зображенні, тобто знаходять межі об'єктів, де один об'єкт переходить в інший або у фон. Це важливий етап у багатьох задачах комп'ютерного зору.

Є велика кількість таких детекторів, але для їх розуміння розглянемо декілька з них, а саме детектор Собеля та детектор Канні.

Детектор Sobel використовується в цифровому обробленні зображень та комп'ютерному зорі. Названий він на честь Ірвіна Собеля та Гарі Фельдмана – фахівців Стенфордської лабораторії штучного інтелекту (SAIL). Основа детектора базується на зв'язуванні зображення з невеликим знімним фільтром та

цілим числом у горизонтальному та вертикальному напрямках, і цей процес простий з точки зору обчислень. Він працює, обчислюючи градієнт інтенсивності зображення на кожному пікселі зображення, і тим самим, наскільки ймовірно, що цей піксель позначає край.

Є два типи фільтрів виявлення країв. Перший тип використовує колірну гаму, яка обчислює похідну першого порядку цифрового зображення, таку як оператор (Sobel, Prewitt, Robert). Другий тип називається гаусовим і залежить у своїх обчисленнях від похідної другого порядку в цифровому зображенні, такої як детектор Canny та лапласіан гаусового детектора. Особливістю оператора є його простота та надання приблизних значень для величини градієнта, а також його здатність виявляти краї і напрямок. У галузі цифрового оброблення зображень згорткова матриця або маска – це невелика матриця, яка використовується для виявлення країв, що досягається шляхом згортки між ядром і зображенням[271].

Функція *cv2.Sobel* бібліотеки OpenCV реалізує детектор Sobel (рис. 6.16).

Синтаксис функції:

cv2.Sobel(src, ddepth, dx, dy, ksize),

де *src* – вхідне зображення;

ddepth – глибина вихідного зображення;

dx, dy – порядок похідної по кожній осі;

ksize – розмір ядра Собеля (3, 5, 7…).

```
# Застосування оператора Собеля
sobel_x = cv2.Sobel(small_image, cv2.CV_64F, dx: 1, dy: 0, ksize=3)  # похідна по X
sobel_y = cv2.Sobel(small_image, cv2.CV_64F, dx: 0, dy: 1, ksize=3)  # похідна по Y

# Обчислення градієнта
sobel_magnitude = cv2.magnitude(sobel_x, sobel_y)

# Перетворення до 8-бітного формату
sobel_x = cv2.convertScaleAbs(sobel_x)
sobel_y = cv2.convertScaleAbs(sobel_y)
sobel_magnitude = cv2.convertScaleAbs(sobel_magnitude)
```

Рисунок 6.16. Фрагмент коду із застосування детектора Sobel

Результат виконання коду показаний на рис. 6.17.

Модуль градієнта – це значення, яке описує інтенсивність зміни яскравості в зображенні. У контексті детектора Sobel, модуль градієнта дозволяє об'єднати похідні по X та Y в єдину карту меж.

Детектор Canny – один з ефективних алгоритмів виявлення країв, метою якого є ідентифікація та виділення меж об'єктів на зображенні.

[271] Elham M., Rawaa T., Howraa M. Design and Fundamentals of Sobel Edge Detection of an Image. 2022. Vol. 9. P. 2458-9403. URL: https://www.researchgate.net/publication/359894954_Design_and_Fundamentals_of_Sobel_Edge_Detection_of_an_Image

а) *б)* *в)*

Рисунок 6.17. Результат виділення країв детектором Sobel
а – виділення по осі x; б – виділення по осі y; в – модуль градієнта

Виявлення країв – це метод, який використовується в обробленні зображень для пошуку меж об’єктів на зображенні. Край визначається як раптова зміна інтенсивності пікселів на зображенні. Краї є межами між окремими об’єктами або областями з різним рівнем інтенсивності.

Алгоритм виявлення країв Канні – це багатоетапний процес, призначений для ідентифікації країв на зображенні, який мінімізує хибнопозитивні результати та точно локалізує виявлені краї[272].

В OpenCV детектор Canny реалізується функцією *cv2.Canny()*.

Синтаксис функції:

cv2.Canny(image, threshold1, threshold2),

де *threshold1* – нижній поріг (мінімальна сила градієнта);

threshold2 – верхній поріг (максимальна сила градієнта).

Перед застосуванням *cv2.Canny()* зображення має бути перетворено на відтінки сірого.

Фрагмент коду програми, реалізації детектора зображено на рис. 6.18.

```
# Застосування детектора Канні
edges = cv2.Canny(small_image, threshold1=100, threshold2=200)
```

Рисунок 6.18. Фрагмент коду із застосування детектора Canny

Результат виконання коду показано на рис. 6.19.

[272] Himangi Agrawal, Krish Desai. Canny edge detection: a comprehensive review. *International Journal of Technical Research & Science.* 2024. Vol. 9. P. 27-35. DOI: 10.30780/specialissue-ISET-2024/023

Рисунок 6.19. Результат виділення країв детектором Canny

Отже, детектори країв і виявлення контурів є ключовими інструментами для локалізації та аналізу об'єктів на зображенні. Вони спрощують структуру сцени та забезпечують основу для подальшого оброблення або розпізнавання.

3) Фільтрація зображень

Фільтрація зображень – це процес модифікації або покращення зображення. Вона може бути використана для покращення деяких особливостей (країв) зображення або навіть для видалення деяких особливостей зображення. Фільтрація зображень охоплює різні кроки, як-от: згладжування, підвищення різкості, покращення країв, виявлення країв, видалення шуму тощо. Фільтрація зображень може бути останнім кроком в обробленні зображень, де результатом фільтрації зображення є очікуване зображення, або навіть проміжним кроком, де відфільтроване зображення може бути використане іншою технологією, наприклад, машинним навчанням.

Методи фільтрації використовуються для видалення шуму або покращення певних особливостей зображень. Є два поширені типи фільтрації зображень – низькочастотна та високочастотна (рис. 6.20)[273].

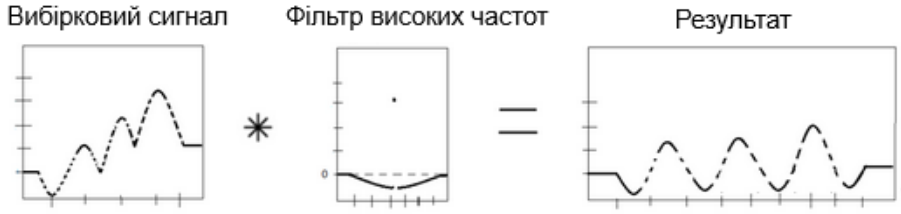
Низькочастотна фільтрація розглядається як згладжувальний фільтр, який використовується для послаблення високих частот та збереження нижчих. Вона має лише додатні значення (1 напрямок). Низькочастотні фільтри зменшують високочастотний шум, зберігаючи при цьому низькочастотну інформацію, ефективно згладжуючи зображення. Це допомагає усунути зернистість.

Високочастотна фільтрація використовується для послаблення низьких частот і збереження високих. Ідея полягає у виділенні частин сигналу, де він найбільше змінюється. Їх можна ідентифікувати як фільтри високих частот, оскільки вони мають як додатні, так і від'ємні значення (2 напрямки). Такі фільтри покращують краї та дрібні деталі, збільшуючи високочастотні компоненти. Вони

[273] Brito P. Computer Vision – The Importance of Filtering. URL: https://surl.lt/nlnkyo

корисні для виявлення країв та виділення ознак, допомагаючи визначити межі об'єктів на зображенні[274].

а)



б)

Рисунок 6.20. Типи фільтрації зображень
а – низькочастотна; б – високочастотна

OpenCV використовується для застосування різних фільтрів, які можуть значно покращити якість зображення, зменшити шум, підвищити контрастність або виділити контури. До основних фільтрів відносять:

– фільтри згладжування: фільтр Гауса, медіанний та двосторонній фільтри;

– фільтри покращення контрасту та різкості: фільтр високих частот, оператора Лапласа.

Розглянемо спочатку **фільтри згладжування** (розмиття або шумозаглушення), серед яких поширеними є фільтр Гауса, медіанний та двосторонній фільтри.

Фільтр Гауса – це лінійний фільтр, який зазвичай використовується для зменшення шуму або розмиття зображення. По суті краї зображення розмиваються, а контрастність зменшується[275]. Синтаксис функції для цього фільтра:

cv2.GaussianBlur(image, shapeOfTheKernel, sigmaX),

де *image* – зображення, яке потрібно розмити;

shapeOfTheKernel – форма матриці 3 на 3/5 на 5;

sigmaX – стандартне відхилення ядра Гауса, яке за умовчанням дорівнює 0.

Медіанний фільтр – це нелінійний метод цифрової фільтрації, який часто використовується для видалення шуму із зображення. Синтаксис функції для цього фільтра:

cv2.medianBlur(src, dst, ksize),

де *src* – вхідне зображення;

dst – вихідне зображення;

ksize – розмір ядра, висота і ширина якого мають бути додатними та непарними значеннями.

[274] Image Filtering in Python Using Pillow. URL: https://surl.li/dtampd
[275] Open Source Computer Vision. URL: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

Двосторонній фільтр – це нелінійний фільтр згладжування зображень, що зберігає краї та зменшує шум. Він замінює інтенсивність кожного пікселя середньозваженим значенням інтенсивності сусідніх пікселів. Синтаксис функції:

cv2.bilateralFilter(src, dst, d, sigmaColor, sigmaSpace, borderType),

де *src* − об'єкт Mat, який подає вхідне зображення для цієї операції;
dst − об'єкт Mat, який подає вихідне зображення для цієї операції;
d – числова змінна цілого типу, яка задає діаметр околиці пікселя;
sigmaColor − змінна цілочислового типу, яка задає сигму фільтра в колірному просторі;
sigmaSpace − змінна цілочислового типу, яка задає сигму фільтра в координатному просторі;
borderType − цілочисловий об'єкт, який задає тип використаної межі.

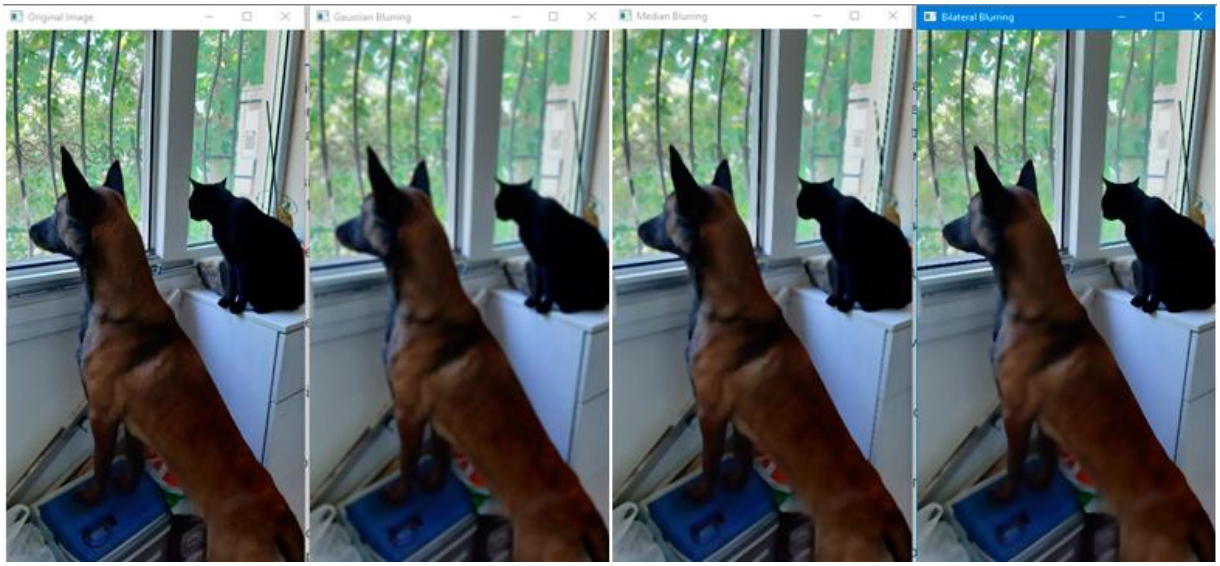
На рис. 6.21 наведено приклад коду програми використання фільтрів згладжування, а на рис. 6.22 – результат її виконання.

```
# Фільтр Гауса
Gaussian = cv2.GaussianBlur(small_image, ksize: (7, 7), sigmaX: 0)
cv2.imshow( winname: 'Gaussian Blurring', Gaussian)

# Медіанний фільтр
median = cv2.medianBlur(small_image, ksize: 5)
cv2.imshow( winname: 'Median Blurring', median)

# Двосторонній фільтр
bilateral = cv2.bilateralFilter(small_image, d: 9, sigmaColor: 75, sigmaSpace: 75)
cv2.imshow( winname: 'Bilateral Blurring', bilateral)
```

Рисунок 6.21. Фрагмент коду програми фільтрів згладжування



а) *б)* *в)* *г)*

Рисунок 6.22. Результат фільтрації зображення
а – оригінал; б – Gaussian Blurring; в – Median Blurring; г – Bilateral Blurring

Тепер розглянемо **фільтри покращення контрасту та різкості**. Підвищення різкості зображення – це процес підкреслення країв зображення, щоб зробити його чіткішим та чіткішим. В OpenCV є різні методи підвищення різкості зображення, наприклад, використання фільтра високих частот або оператора Лапласа.

Одним із найпоширеніших методів підвищення різкості зображення є використання *фільтра високих частот*. Цей фільтр працює шляхом підсилення високочастотних компонентів зображення, таких як краї, та одночасного придушення низькочастотних компонентів, таких як плоскі області. Для цього використовується функція *cv2.filter2D()*[276]. Її синтаксис:

resulting_image = cv2.filter2D(src, ddepth, kernel),

де *src* – вхідне зображення, до якого застосовується фільтрація. По суті це матриця, яка подає зображення у значеннях інтенсивності пікселів;
ddepth – це глибина цільового зображення. Значення -1 означає, що *результуюче* зображення матиме таку саму глибину, як і вихідне зображення;
kernel – це матриця фільтра, застосована до зображення.

Функція *filter2D()* згортає зображення з ядром, що призводить до розмиття або підвищення різкості зображення та покращення його характеристик.

Приклад коду, який показує, як виконати фільтрацію високих частот на зображенні, наведено на рис. 6.23. Результат виконання коду наведено на рис. 6.24.

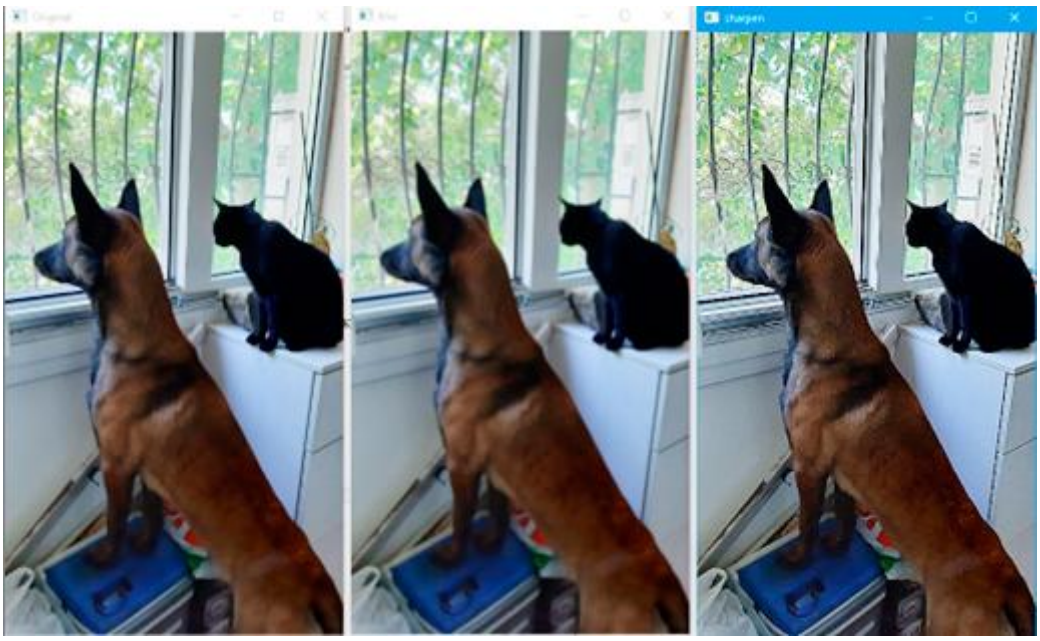
```
# Створення ядра (фільтр розмиття)
kernel1 = np.array( object: [
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
], dtype=np.float32) / 9

# Створення ядра (фільтр різкості)
kernel2 = np.array( object: [
    [0, -1, 0],
    [-1, 5,-1],
    [0, -1, 0]
], dtype=np.float32)

# Застосування фільтру
ddepth = -1
resulting_image_1 = cv2.filter2D(small_image, ddepth, kernel1)
resulting_image_2 = cv2.filter2D(small_image, ddepth, kernel2)
```

Рисунок 6.23. Фрагмент коду програми застосування фільтра високих частот

[276] Python OpenCV filter2D() function – A Complete Guide. URL: https://www.askpython.com/python-modules/opencv-filter2d



а) *б)* *в)*

Рисунок 6.24. Результат виконання програми
а – оригінал; б – фільтр розмиття; в – фільтр різкості

4) Виявлення особливих точок і побудова дескрипторів

У сфері комп'ютерного зору широкого поширення набули методи розпізнавання візуальних об'єктів, які ґрунтуються на локальних ознаках зображення. Ці підходи стосуються виявлення множини особливих точок на зображенні та побудову їх опису у формі числових або бінарних векторів – дескрипторів, які відображають вміст локальних околиць навколо виявлених точок[277]. Основна причина їх використання в тому, що дані методи потребують менше зберігання потрібної інформації про еталон об'єкта для його пошуку.

Особлива точка – це точка зображення, яка має деякі особливості на відміну від інших точок даного зображення. Ознакою такої точки може бути різкий перепад кольорів, яскравості, кути, краї тощо. Ці ознаки мають бути особливими, щоб їх можна було знайти на іншому зображенні. Процес виявлення та фіксування особливих точок досягають шляхом використання детектора і дескриптора.

Детектор – це метод отримання особливих точок із зображення[278].

Дескриптор – це числове подання області або ознаки на зображенні, призначене для фіксації її унікальних характеристик у такий спосіб, щоб алгоритми могли порівнювати або зіставляти її з іншими ознаками. Дескриптори є критично важливими, оскільки необроблені дані пікселів занадто шумні та мінливі для прямого порівняння – зміни освітлення, повороти або різниця в масштабі можуть зробити так, щоб та сама ознака виглядала зовсім по-різному. Перетворюючи ознаки на числові вектори, дескриптори дозволяють виконувати такі завдання:

---

[277] Гороховатський В.О., Пупченко Д.В., Солодченко К.Г. Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення. *Системи управління навігації та зв'язку*. 2018. № 1(47). С. 93-98. DOI: 10.26906/SUNZ.2018.1.093

[278] Стоян А.О. Дослідження методів та засобів виявлення об`єктів на відеозображеннях. URL: https://eom.lpnu.ua/sntk/doc/spr2020/stoyan.pdf

розпізнавання об'єктів, зшивання зображень або відстеження шляхом вимірювання подібності між ознаками[279].

Дескриптори зазвичай генеруються шляхом аналізу пікселів навколо ключової точки, яку виявляють за допомогою алгоритмів, як-то SIFT, SURF або ORB.

Дескриптор SIFT (Scale-invariant feature transform, перетворення ознак, інваріантне до масштабу) – це широко використовуваний дескриптор ознак, інваріантний до масштабування, обертання та трансформацій. Він описує локальні області зображення за їх градієнтними величинами та орієнтаціями. Відрізняється стійкістю до різних трансформацій, що робить його ефективним у використанні у комп'ютерному зорі[280].

На рис. 6.25 наведений фрагмент коду програми із застосування дескриптора SIFT, а результат виконання коду наведено на рис. 6.26.

```
#Ініціалізація детектора SIFT
sift = cv2.SIFT_create()

# Виявлення особливих точок та обчислення дескрипторів
keypoints, descriptors = sift.detectAndCompute(image, None)

# Додавання особливих точок на зображення
image_with_keypoints = cv2.drawKeypoints(image, keypoints, outImage: None)
```

Рисунок 6.25. Фрагмент коду програми із застосування дескриптора SIFT

Рисунок 6.26. Застосування дескриптора SIFT

[279] What is descriptor in computer vision? URL: https://surl.li/sokqhh

[280] Mistry D., Banerjee A. Comparison of Feature Detection and Matching Approaches: SIFT and SURE. *GRD Journals-Global Research and Development Journal for Engineering*. 2017. Vol. 2 No. 3. DOI: 10.70179/3qhg4p03

Дескриптор SIFT (Speeded-Up Robust Features, прискорені робастні ознаки) – це алгоритм, який базується на концепціях SIFT, але має на меті покращити швидкість та ефективність, зберігаючи при цьому робастність. Цей дескриптор описує локальні особливості зображення, які характеризуються низькою обчислювальною складністю та високим інформативним вмістом. Він має більші переваги в описі деталей зображення та обробленні змін масштабу.

Принцип SURF подібний до принципу SIFT, оскільки ці дескриптори орієнтовані на просторовий розподіл градієнтної інформації. Алгоритм SURF стійкий до масштабування, обертання, розмиття та шуму, які він успадковує від алгоритму SIFT. Однак традиційний алгоритм SURF вимагає великого обсягу пам'яті та займає багато часу в процесі інтеграції зображень. Тому його було вдосконалено з точки зору інтегрованого оброблення зображень, пропонуючи швидкий інтегрований алгоритм оброблення зображень, який займає лише один простір зображення[281].

На рис. 6.27 та рис. 6.28 наведено фрагмент коду програми із застосування дескриптора SURF та результат його виконання.

```
# Ініціалізація SURF (hessianThreshold керує кількістю ключових точок)
surf = cv2.xfeatures2d.SURF_create(hessianThreshold=400)

# Виявлення ключових точок та обчислення дескрипторів
keypoints, descriptors = surf.detectAndCompute(image, None)

# Візуалізація результату
image_with_keypoints = cv2.drawKeypoints(image, keypoints, outImage: None, color: (0, 255, 0), flags: 4)
```

Рисунок 6.27. Фрагмент коду програми із застосування дескриптора SURF

Рисунок 6.28. Застосування дескриптора SURF

[281] Lei Z., Jiexin P., Gui Ch., Xiaoli S. An Improved SURF and Modified Zernike Moments Descriptor for Object Recognition. *Electronics.* 2025. Vol. 14(5), No 1025. DOI: https://doi.org/10.3390/electronics14051025

Дескриптор ORB (Oriented FAST and Rotated BRIEF) є детектором ознак та екстрактором дескрипторів, який поєднує в собі методи FAST і BRIEF з метою забезпечення ефективного й обчислювально недорогого виявлення ключових точок і побудови відповідних дескрипторів. ORB виконує виявлення ознак на кожному рівні побудованої піраміди зображення, обчислює для кожної з них координати, рівень (октаву), на якому вона була виявлена, а також бінарний дескриптор ознаки.

В якості детектора ключових точок використовується алгоритм FAST (Features from Accelerated Segment Test), який базується на аналізі пікселів у певному радіусі навколо точки-кандидата на ключову точку. Він визначає, чи існує послідовність з n суміжних пікселів, які значно яскравіші або темніші за центральний піксель. Для прискорення обчислень застосовується попередній відбір підмножини пікселів перед повним тестуванням області.

Оскільки FAST не забезпечує інформацію про орієнтацію ключових точок, в ORB для цього використовується метод обчислення центроїда з урахуванням інтенсивності пікселів у локальній області. Вектор, спрямований від ключової точки до центроїда, визначає її орієнтацію. Це дозволяє компенсувати чутливість методу BRIEF до обертання. Перед обчисленням дескриптора область навколо ключової точки орієнтується відповідно до обчисленого напрямку, після чого виконується серія бінарних тестів над парами пікселів, результати яких формують бітовий вектор – дескриптор ознаки[282].

Основною перевагою ORB є його висока швидкість роботи та низькі обчислювальні витрати, що робить його придатним для застосування у задачах реального часу, таких як відеооброблення або вбудовані системи з обмеженими ресурсами. Водночас недоліком алгоритму є обмежена стійкість до змін масштабу та кута зору порівняно з більш складними методами, як-от SIFT або SURF, хоча ORB частково компенсує ці недоліки завдяки врахуванню орієнтації.

На рис. 6.29 та 6.30 наведено фрагмент коду програми із застосування дескриптора ORB та результат його виконання.

```
# Ініціалізація ORB
orb = cv2.ORB_create()

# знаходження ключових точок ORB
keypoints = orb.detect(image,None)

# обчислення дескриптора
keypoints, descriptors = orb.compute(image, keypoints)

# розташування ключових точок
img = cv2.drawKeypoints(image, keypoints, outImage: None, color=(0,255,0), flags=0)
```

Рисунок 6.29. Фрагмент коду програми із застосування дескриптора ORB

[282] Kennerley M. A Comparison of SIFT, SURF and ORB on OpenCV. URL: https://mikhail-kennerley.medium.com/a-comparison-of-sift-surf-and-orb-on-opencv-59119b9ec3d0

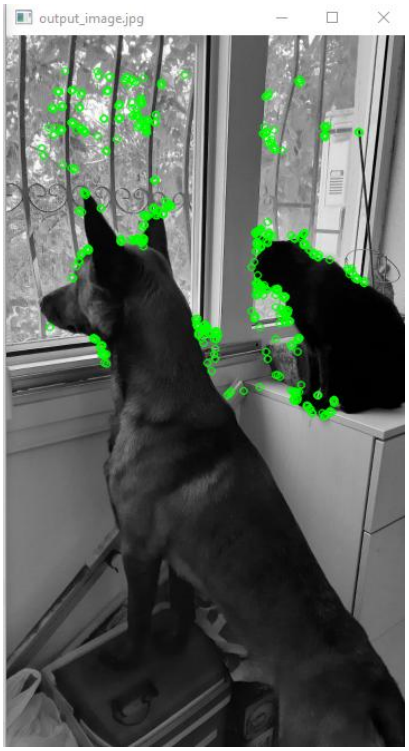


Рисунок 6.30. Застосування дескриптора ORB

Розуміння різниці між розміром дескриптора, часом обчислення та точністю зіставлення є важливим для створення ефективних систем комп’ютерного зору.

5) Хаф-перетворення

Хаф-перетворення (Hough Transform) є ефективним інструментом комп’ютерного зору, який широко застосовується для виявлення геометричних фігур на зображеннях, як-от: прямі, кола та інші об’єкти, які мають математично визначену форму[283]. Цей метод є особливо цінним у задачах, в яких об’єкти можуть бути частково зруйнованими, зашумленими або спотвореними, оскільки дозволяє виявити їх на основі акумуляції голосів у параметричному просторі.

У загальному вигляді Хаф-перетворення перетворює задачу виявлення геометричних структур у зображенні на задачу пошуку кластерів точок у параметричному просторі. Наприклад, для виявлення прямих використовується параметризація у полярних координатах, де кожна точка на зображенні відповідає синусоїді в параметричному просторі (ρ, θ). Точки, які лежать на одній прямій на зображенні, матимуть спільне перетинання синусоїд, яке вказує на наявність відповідної лінії.

Для практичного застосування методу в OpenCV використовується функція *cv2.HoughLines()*, яка має такий синтаксис:

HoughLines(image, rho, theta, threshold),

де *image* – вхідне зображення, зазвичай після операції виявлення країв, наприклад, методом Кенні;

rho – роздільність по радіальній координаті ρ (у пікселях);

theta – кутова роздільність (у радіанах);

[283] Hough Line Transform. URL: https://docs.opencv.org/4.x/d6/d10/tutorial_py_houghlines.html

threshold – мінімальна кількість перетинів, щоб лінія вважалась знайденою.

Перед застосуванням Хаф-перетворення рекомендовано попередньо виконати процедуру виявлення країв зображення, наприклад, за допомогою оператора Кенні (*cv2.Canny()*), щоб зменшити кількість зайвих пікселів та покращити точність виявлення.

На рис. 6.31 подано фрагмент коду із застосуванням функції *HoughLines()* з бібліотеки OpenCV. На рис. 6.32 зображено результат виконання цього коду, який демонструє виявлені прямі на зображенні.

```
# Виявлення країв за допомогою оператора Кенні
edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Застосування перетворення Хафа для знаходження прямих
lines = cv2.HoughLines(edges, rho=1, theta=np.pi / 180, threshold=140)

# Якщо є знайдені лінії — малюємо їх
if lines is not None:
    for line in lines:
        rho, theta = line[0]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))

        cv2.line(image, pt1: (x1, y1), pt2: (x2, y2), color: (0, 0, 255), thickness: 2)
```

Рисунок 6.31. Фрагмент коду програми із застосування Хаф-перетворення

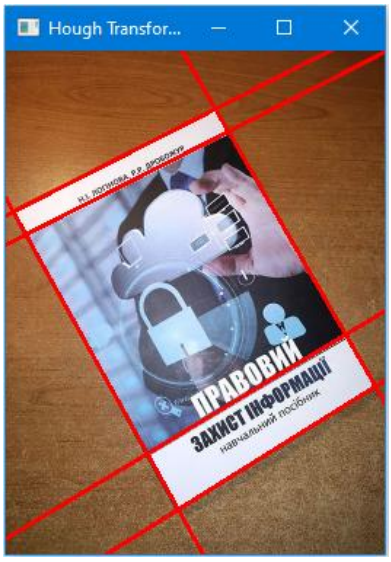


Рисунок 6.32. Застосування Хаф-перетворення

6) Ректифікація зображень

Ректифікація зображень – це процес геометричного перетворення, яке усуває спотворенні і викривлення, спричинені оптикою камери або перспективою, з метою приведення зображення до уніфікованої проєкції. Основна мета – забезпечити коректне співвідношення між пікселями та реальними координатами об'єктів, що дозволяє здійснювати точні вимірювання, порівняння та реконструкцію сцени[284].

Приклад поду для покращене виявлення кутів із субпіксельною точністю[285]:

```
import cv2
import numpy as np

# Load image
img = cv2.imread("chessboard.png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find chessboard corners
ret, corners = cv2.findChessboardCorners(gray, (7, 10), None)

# Refine corner positions using subpixel accuracy
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
corners = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

# ... rest of the calibration code ...
```

У цьому прикладі реалізовано калібрування та випрямлення зображення камери за допомогою OpenCV й удосконалено виявлення кутів за допомогою cv2.cornerSubPix, покращуючи точність розташування кутів, що призводить до кращих результатів калібрування. Значне покращення порівняно з базовим виявленням кутів забезпечує субпіксельну точність, що призводить до більш точного калібрування.

У комп'ютерному зорі ректифікація є критичною для *стереозору*, де необхідно знайти відповідні точки на двох зображеннях. Ректифіковані зображення мають вирівняні епіполярні лінії, що значно спрощує пошук відповідностей.

У системах *доповненої реальності (AR)* та *3D-реконструкції* ректифікація дозволяє точно накладати віртуальні об'єкти на фізичне середовище.

У *геоінформаційних системах (GIS)* ректифікація використовується для приведення аерофотознімків до єдиної координатної системи.

У тестуванні ПЗ для *розпізнавання облич* ректифікація дозволяє нормалізувати положення голови, що підвищує точність алгоритмів і зменшує кількість хибних спрацьовувань.

У розробленні *систем контролю якості* на виробництві, наприклад, при аналізі зображень деталей, ректифікація забезпечує точне вимірювання розмірів незалежно від кута зйомки.

7) Стиснення зображень і відеоданих

Стиснення зображень і відео — це фундаментальний напрям комп'ютерного зору, який забезпечує ефективне зберігання, передавання та оброблення візуальної інформації. Є два основні підходи: стиснення без втрат (lossless), що зберігає повну інформацію, та стиснення з втратами (lossy), яке дозволяє суттєво

[284] Camera Calibration and Image Rectification Project. URL: https://github.com/Thiago-Ramalho/vision-project-ive

[285] OpenCV Image Rectification: Fixing Distortion Issues in Python. URL: https://tech-champion.com/programming/python-programming/opencv-image-rectification-fixing-distortion-issues-in-python/

зменшити обсяг даних за рахунок часткової втрати якості, прийнятної для більшості практичних задач.

Стиснення дозволяє оптимізувати зберігання великих обсягів візуальних даних, що є критичним для хмарних сервісів, мобільних застосунків та систем реального часу. У системах машинного зору для виробництва, стиснення дозволяє зберігати великі обсяги відео з конвеєрів або камер контролю якості, не перевантажуючи локальні або хмарні сховища. У нейромережевих системах, оптимізоване стиснення дозволяє зменшити обсяг вхідних даних без втрати точності інференції, що критично для edge AI та real-time inference.

У CI/CD-процесах тестування ПЗ, що працює з візуальними даними, стиснення дозволяє зберігати великі набори тестових зображень у репозиторіях без перевищення лімітів, а також пришвидшує передачу між модулями.

У розробленні систем відеоспостереження, стиснення відео (наприклад, за стандартами H.264, HEVC) дозволяє зменшити навантаження на мережу та сервери без втрати критичної інформації.

У мобільних застосунках, які використовують камеру (наприклад, для розпізнавання об'єктів або доповненої реальності), стиснення JPEG, WebP або HEVC дозволяє зменшити затримку при завантаженні та обробленні зображень.

Приклади сучасних рішень для стиснення зображень і відео:

– *CompressAI-Vision* – відкритий фреймворк для оцінки методів стиснення з урахуванням точності виконання комп'ютерних зорових задач. Він дозволяє тестувати нові кодеки в контексті downstream inference, зберігаючи баланс між розміром даних і точністю моделі[286];

– методи *Deep learning-guided video compression* використовують адаптивні нейромережі для виділення важливих регіонів кадру, які мають бути закодовані з високою точністю, тоді як менш важливі області стискаються агресивніше. Це дозволяє зменшити обсяг відео без втрати критичної інформації для машинного аналізу[287];

– підхід *Multi-modality compression* враховує кореляцію між різними сенсорними каналами (наприклад, інфрачервоним і видимим зображенням), дозволяючи ефективніше стискати дані без дублювання інформації[288].

Попри те що сучасні системи ШІ в комп'ютерному зорі дедалі частіше застосовують глибоке навчання, класичні методи все ще залишаються актуальними. Їх перевага у тому, що вони ґрунтуються на чітких математичних принципах та легко інтерпретуються. Дані методи не вимагають багато даних для навчання, обробляють зображення на високої швидкості й можуть працювати у реальному часі. Недоліками є те, що налаштування параметрів може бути трудомістким та вимагає додаткових знань для досягнення результатів. Класичні методи

[286] Choi H., Han H., Rosewarne C., Racapé F. CompressAI-Vision: Open-source software to evaluate compression methods for computer vision tasks. *arXiv*. 2025. URL: https://arxiv.org/pdf/2509.2077

[287] Deep learning-guided video compression for machine vision tasks. *EURASIP Journal on Image and Video Processing*, 2024. URL: https://jivp-eurasipjournals.springeropen.com/articles/10.1186/s13640-024-00649-w

[288] Lu G., Zhong T., Geng J., Hu Q., Xu D. Learning Based Multi-Modality Image and Video Compression. *CVPR*. 2022. URL: https://openaccess.thecvf.com/content/CVPR2022/papers/Lu_Learning_Based_Multi-Modality_Image_and_Video_Compression_CVPR_2022_paper.pdf

чутливі до змін за умов зйомки. Їхня продуктивність може знижуватися в складних умовах.

Тому класичні методи можна використовувати як попередній етап оброблення зображень або тоді, коли ресурси системи обмежені, а також для завдань, де потрібна пояснювати результати аналізу та контроль над процесом аналізу.

6.3.2 Методи глибокого навчання

На початку 2010-х років галузь комп'ютерного зору зазнала суттєвих змін завдяки прориву в технологіях глибокого навчання. Визначальним етапом стала перемога моделі AlexNet, заснованої на згорткових нейронних мережах (Convolutional Neural Networks, CNN), у змаганні ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 року[289]. Ця модель продемонструвала суттєве покращення точності, порівняно з попередніми методами, що стало поштовхом до активного розвитку підходів до оброблення зображень[290].

Глибоке навчання дозволяє автоматично вивчати складні подання візуальних даних, без потреби в ручному створенні ознак, що було необхідно в класичних методах. CNN автоматично навчаються виявляти локальні патерни (краї, текстури, форми) і поступово будують більш абстрактні подання, що робить їх дуже ефективними для широкого спектра задач комп'ютерного зору.

Серед основних і найпоширеніших методів та моделей глибокого навчання варто виділити:

– CNN – базова архітектура, яка використовується для класифікації зображень, виявлення ознак, виведення карти ознак тощо. Вона відома за моделями типу LeNet, AlexNet, VGGNet, GoogLeNet, ResNet[291];

– YOLO (You Only Look Once)[292] – швидкий алгоритм для розпізнавання об'єктів у реальному часі. Він виконує виявлення об'єктів за один прохід мережі, що дозволяє досягати високої продуктивності при збереженні задовільної точності[293];

– SSD (Single Shot MultiBox Detector) – ще один підхід для одноетапного виявлення об'єктів, який забезпечує баланс між точністю та швидкістю. Він менш вимогливий до ресурсів, порівняно з більш складними архітектурами;

– Faster R-CNN – двохетапна архітектура для точного виявлення об'єктів, яка спочатку пропонує області інтересу (Region Proposal Network), а потім класифікує їх. Цей метод є більш обчислювально складним, але й дуже точним;

– Mask R-CNN – розширення Faster R-CNN, яке дозволяє не лише виявляти об'єкти, а й будувати точні маски сегментації для кожного з них, поєднуючи детекцію та піксельну сегментацію.

---

[289] AlexNet and ImageNet: The Birth of Deep Learning. URL: https://www.pinecone.io/learn/series/image-search/imagenet

[290] Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems.* 2012. Vol. 25(2). DOI: https://doi.org/10.1145/3065386

[291] What are convolutional neural networks? URL: https://www.ibm.com/think/topics/convolutional-neural-networks

[292] YOLO: Algorithm for Object Detection Explained. URL: https://www.v7labs.com/blog/yolo-object-detection

[293] Deepika H.C., Srinivasan G.N. An Overview of You Only Look Once: Unified, Real-Time Object Detection. *International Journal for Research in Applied Science & Engineering Technology* (*IJRASET*). 2020. P. 607-609. DOI: https://doi.org/10.22214/IJRASET.2020.6098

CNN (Convolutional Neural Networks) – це особливий тип штучних нейронних мереж, спеціально розроблений для автоматичного вилучення ознак із даних, які мають сітчасту структуру, зокрема двовимірну або тривимірну. Такий підхід демонструє високу ефективність у задачах оброблення зображень і відео, де важливо зберігати та аналізувати просторові залежності. Саме завдяки цим властивостям CNN стали фундаментальним інструментом у сфері комп’ютерного зору.

Однією з перших успішних реалізацій CNN є модель *LeNet*, створена для задачі розпізнавання рукописних цифр. Вона стала основоположною для подальших досліджень у галузі глибокого навчання, запровадивши такі ключові компоненти, як згорткові шари, шари субдискретизації (pooling) та принцип ієрархічного подання ознак.

Архітектура LeNet побудована з кількох послідовно з’єднаних шарів, серед яких виділяють згорткові, об’єднуючі та повністю зв’язані шари. Передавання даних здійснюється у прямому напрямку, коли вихід кожного шару слугує вхідними даними для наступного (рис. 6.33)[294].

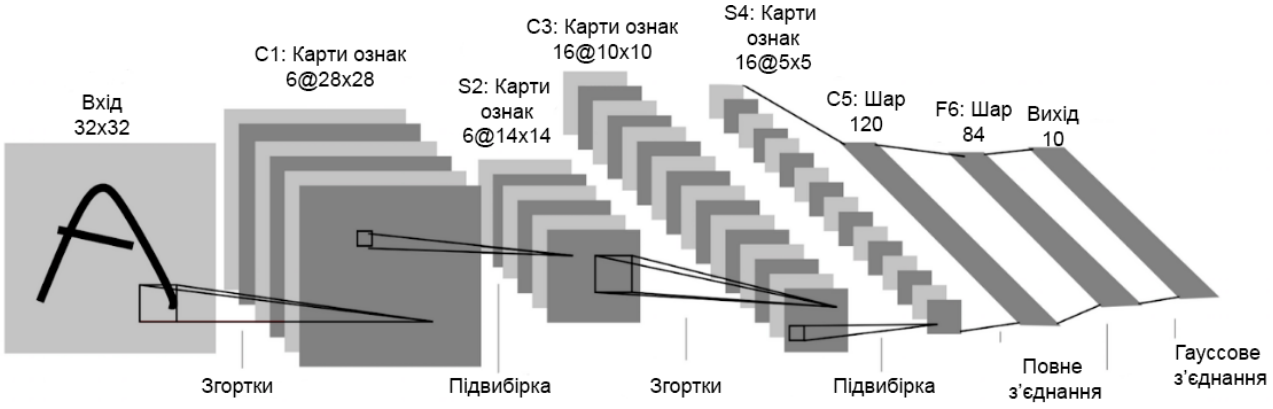


Рисунок 6.33. Архітектура LeNet

Згорткові шари (C1, C3) складаються з набору параметризованих фільтрів, які проходять по вхідному зображенню, виявляючи локальні просторові закономірності та формуючи карти ознак.

Шари субдискретизації (S2, S4) виконують зменшення просторової роздільності карт ознак шляхом усереднення або вибору максимального значення, що знижує обчислювальну складність і підвищує інваріантність до зміщення.

Повністю зв’язані шари (F5, F6) перетворюють згорткові ознаки на векторну форму та здійснюють кінцеву класифікацію за допомогою densely-connected нейронів.

Останній шар використовує функцію активації softmax, що дозволяє отримати ймовірнісний розподіл по класах.

Водночас, через обмежену складність і невелику кількість параметрів модель LeNet найкраще підходить для розв’язання задач з розпізнавання простих образів, таких як окремі символи чи цифри.

2012 року згорткова нейронна мережа **AlexNet** стала першим масштабним прикладом використання методів глибокого навчання для розв’язання задач

[294] Bas A., Topal M., Duman C., Heerden I. A Brief History of Deep Learning-Based Text Generation. *International Conference on Computer and Applications (ICCA).* 2022. DOI: 10.1109/ICCA56443.2022.10039545

класифікації зображень, що стало визначальним проривом у розвитку комп'ютерного зору. Архітектурні рішення, реалізовані в AlexNet, дозволили суттєво знизити складність моделі шляхом оптимізації кількості параметрів, а також подолати низку ключових проблем, пов'язаних з навчанням глибоких нейронних мереж.

Структура AlexNet поєднує згорткові та повнозв'язні шари. Загальна кількість параметрів моделі становить 62,3 млн, а обчислювальна складність одного прямого проходу – близько 1,1 млрд операцій. При цьому згорткові шари, які містять лише близько 6% загальної кількості параметрів, забезпечують понад 95% загального обсягу обчислень (рис. 6.34)[295].

Рисунок 6.34. Архітектура AlexNet

AlexNet складається з восьми навчуваних шарів, серед яких перші п'ять є згортковими, а три останні – повністю з'єднані. На фінальному етапі роботи мережі вихідний вектор подається до функції softmax, яка формує ймовірнісний розподіл між 1000 можливими класами. Для навчання використовується багатокласова логістична регресія, яка полягає у максимізації середнього логарифму ймовірності правильного класифікаційного рішення.

Однією з архітектурних особливостей AlexNet є часткове з'єднання між шарами: ядра згорткових шарів другого, четвертого й п'ятого рівнів підключені лише до частини карт ознак попереднього шару, що є на тому самому графічному процесорі. Водночас ядра третього згорткового шару мають повні з'єднання з усіма вихідними картами другого шару. Повністю з'єднані шари, у свою чергу, реалізують повне з'єднання між усіма нейронами суміжних рівнів.

AlexNet запровадила низку інноваційних рішень, які згодом стали стандартними у проєктуванні глибоких згорткових мереж:

1. *Активаційна функція ReLU* (Rectified Linear Unit)

Замість традиційних функцій активації (sigmoid, tanh) використано ReLU: $f(x) = max(0, x)$. Вона сприяє пришвидшенню навчання, знижує проблему

[295] Krizhevsky A., Sutskever I., Hinton G. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. 2012. Vol. 60. P. 84-90. URL: https://api.semanticscholar.org/CorpusID:195908774

зникнення градієнта та забезпечує кращу переданість похибки при зворотному поширенні.

2. *Dropout*

Для запобігання перенавчанню застосовано техніку Dropout: у процесі тренування випадкові нейрони тимчасово деактивуються з певною ймовірністю. Це сприяє зменшенню залежності від окремих нейронів і покращує здатність моделі до узагальнення.

3. *Локальна нормалізація відгуку* (Local Response Normalization, LRN)

Механізм LRN виконує нормалізацію значень активації в межах локального простору, імітуючи бічне гальмування, що спостерігається у біологічних нейронних мережах. Цей підхід тимчасово сприяв стабілізації навчання, хоча згодом його було замінено пакетною нормалізацією (Batch Normalization).

4. *Використання кількох графічних процесорів* (GPU)

Задля ефективного оброблення великих обсягів даних мережа була розділена на дві частини, кожна з яких оброблялася окремим GPU. Такий підхід дозволив суттєво скоротити час навчання.

5. *Аугментація даних* (Data Augmentation)

Для розширення варіативності навчального набору застосовувалися такі методи: випадкове обрізання зображень, горизонтальне віддзеркалення та зміна інтенсивності кольорів у просторі RGB. Це сприяло підвищенню узагальнюючої здатності моделі.

Створення AlexNet стало визначальним кроком у становленні глибокого навчання, продемонструвавши його практичну ефективність для задач комп'ютерного зору. Вона показала здатність згорткових мереж формувати ієрархічне подання вхідних даних та проклала шлях до створення більш складних архітектур, як-от: VGGNet, GoogLeNet (Inception) та ResNet. AlexNet значно вплинула на подальший розвиток ШІ й прискорила поширення глибоких нейронних мереж у різних галузях.

Мережа **VGGNet** (Visual Geometry Group Network) є однією з найвідоміших архітектур глибоких згорткових нейронних мереж, яка широко застосовується в задачах класифікації зображень. Вона показала високу ефективність, сягнувши точності 92,7% на тестовому наборі даних ImageNet, що складається з понад 14 млн зображень, розподілених між 1000 категоріями. Розмітка була здійснена вручну з використанням краудсорсингової платформи Amazon Mechanical Turk, а самі зображення отримані з відкритих джерел в інтернеті.

Архітектура VGGNet є у двох основних конфігураціях – із 16 або 19 шарами. Її головна особливість полягає в послідовному розміщенні згорткових шарів із малими фільтрами 3×3 (із кроком 1), після яких ідуть шари максимального об'єднання (max-pooling) розміром 2×2 (із кроком 2). Такий підхід дозволяє ефективно вилучати складні ознаки зі збереженням просторової структури зображення, водночас зберігаючи архітектурну простоту. У фінальній частині мережі розташовані повнозв'язні шари (fully connected), які забезпечують класифікацію.

Простота та ефективність цієї архітектури роблять її ідеальним вибором для завдань класифікації зображень (рис. 6.35)[296].

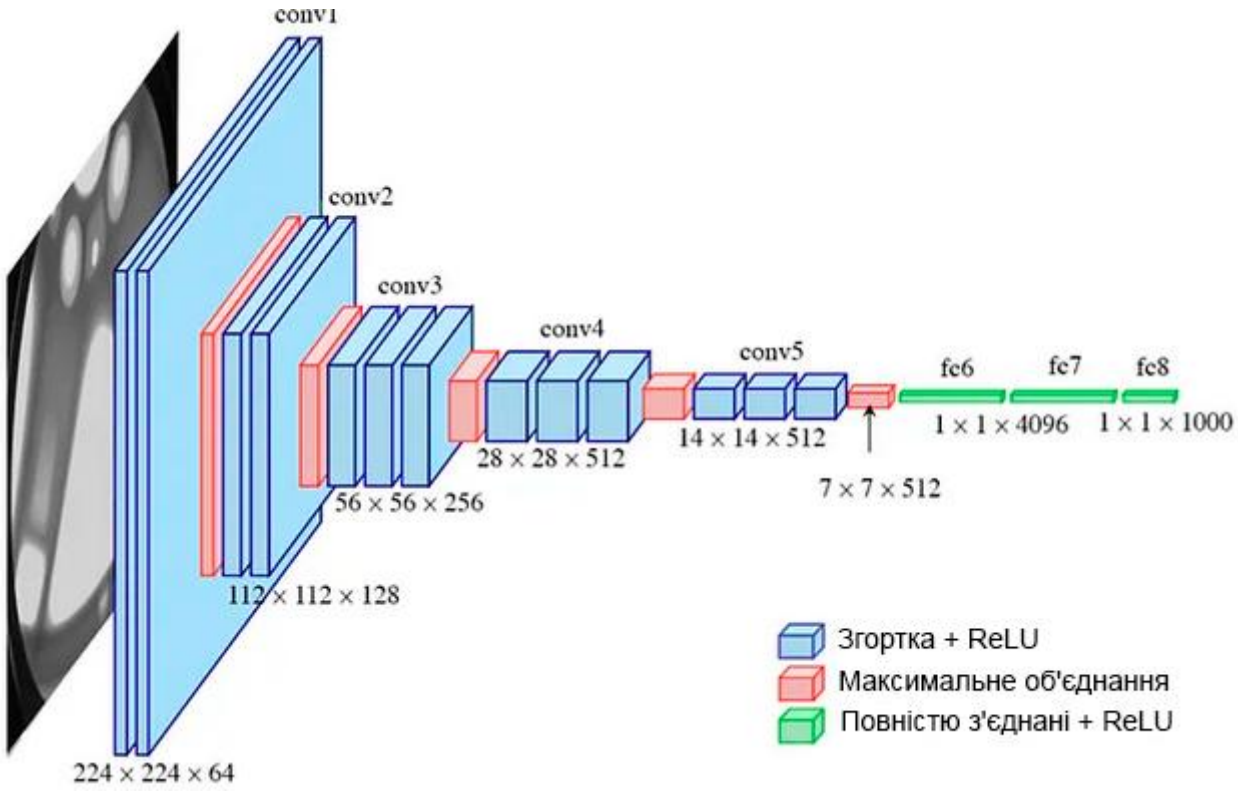


Рисунок 6.35. Архітектура VGGNet

На вхід подаються RGB-зображення розміром 224×224. Вони проходять через стек згорткових шарів із дуже малим рецептивним полем 3×3, яке дає змогу моделі розпізнавати локальні патерни (наприклад, напрямок чи положення об’єкта). У деяких конфігураціях також використовуються згортки 1×1, які виконують лінійне перетворення каналів. Для збереження просторової роздільної здатності застосовується доповнення (padding) з нульовими значеннями шириною 1 піксель.

Просторовий пулінг виконується за допомогою п’яти max-pooling шарів, розташованих після окремих згорткових шарів. Наприкінці розміщується три повнозв’язані шари: перші два мають по 4096 нейронів, а останній – 1000, що відповідає кількості класів у змаганні ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Після останнього шару використовується функція softmax для формування ймовірнісного розподілу по класах.

Усі приховані шари в VGGNet активуються за допомогою функції ReLU, яка забезпечує нелінійність моделі. Шар локальної нормалізації (LRN) у більшості варіантів VGGNet не використовується, оскільки його застосування не призводило до покращення точності, натомість збільшувало споживання пам’яті та час обчислень.

[296] Bezdan T., Džakula N. Convolutional Neural Network Layers and Architectures. *International Scientific Conference on Information Technology and Data Related Research*. Belgrade, Serbia, 2019. P. 445-451. DOI: 10.15308/Sinteza-2019-445-451

Втім простота побудови та структурна послідовність VGGNet роблять її зручною для реалізації й досліджень. Вона залишається важливою базовою архітектурою, яка суттєво вплинула на розвиток нейронних мереж для аналізу зображень.

Попри ефективність, VGGNet має низку обмежень. Зокрема, вона потребує значного часу на навчання і характеризується великою кількістю параметрів. Наприклад, модель VGG16 має обсяг понад 533 МБ, що ускладнює її розгортання в реальних умовах. Саме тому з часом перевагу часто надають компактнішим архітектурам, як-от: SqueezeNet, GoogLeNet чи MobileNet. Попри недоліки ця архітектура є простим блоком для навчання, оскільки її легко реалізувати.

Модель глибокого навчання *GoogLeNet* перевершила попередні архітектури за глибиною, маючи 22 шари, що забезпечує вищу здатність до навчання та покращену продуктивність. При її проєктуванні враховано обчислювальну ефективність і практичність застосування, зокрема можливість використання на пристроях з обмеженими ресурсами, включно з тими, які мають низький обсяг оперативної пам'яті.

Ключовим конструктивним рішенням архітектури є врахування того факту, що об'єкти на зображенні можуть мати різні масштаби. Оскільки згорткові шари мають обмежену область рецептивного поля, ефективне розпізнавання дрібних об'єктів потребує згорток з меншими ядрами, тоді як для більших об'єктів доцільно використовувати згортки з більшими розмірами ядер.

Щоб забезпечити інваріантність до масштабу об'єктів, в архітектурі GoogLeNet запропоновано застосування спеціального компонента – Inception-блока (рис. 6.36)[297]. У базовій версії Inception-блока (рис. 6.36, а) згортки з ядрами різного розміру (1×1, 3×3, 5×5) та операція максимального об'єднання виконуються паралельно, після чого результати всіх гілок об'єднуються. Проте такий підхід призводить до значного зростання кількості вихідних каналів, що ускладнює оброблення, особливо при великій кількості вхідних каналів. Для зменшення обчислювального навантаження було розроблено оптимізовану версію Inception-блока (рис. 6.36, б), в якій перед застосуванням згорток 3×3 та 5×5 виконується попереднє зменшення кількості каналів за допомогою ефективних згорток 1×1.

Структура Inception-модуля має паралельні шляхи оброблення:

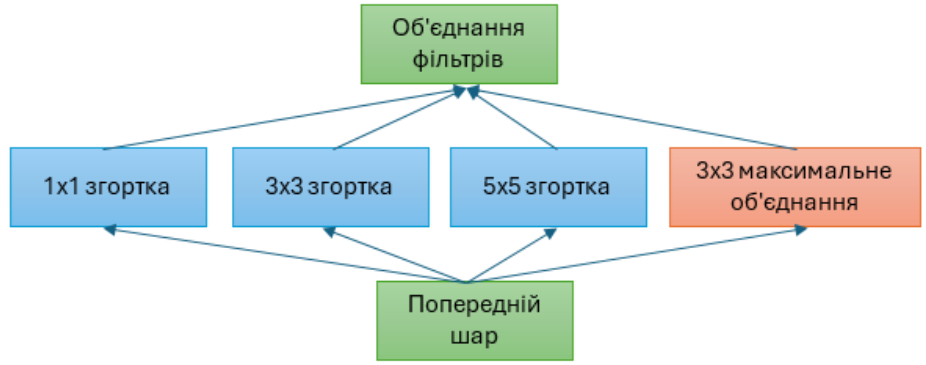
- згортка 1×1 – зменшує кількість вхідних каналів і, відповідно, знижує обчислювальну складність;
- згортка 3×3 – виконує стандартне вилучення ознак середнього масштабу;
- згортка 5×5 – дозволяє виділяти ознаки великого масштабу.

Операція максимального об'єднання (pooling) – зменшує просторові розміри вхідного зображення, зберігаючи найважливішу інформацію.

Після паралельного оброблення результати з усіх чотирьох гілок об'єднуються (concatenation), після чого може бути застосована додаткова згортка 1×1 для зменшення розміру вихідного тензора та покращення характеристик узагальнення моделі.

---

[297] Szegedy, C., et al. Going Deeper with Convolutions. 2015 *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), Boston, MA, 7-12 June 2015. P. 1-9. DOI: https://doi.org/10.1109/CVPR.2015.7298594



а)

б)

Рисунок 6.36. Блок Inception
а – базовий блок; б – оптимізований блок

Така архітектурна організація забезпечує збір інформації на різних масштабах та ефективне оброблення зображень без суттєвого збільшення обчислювальних витрат.

Модель **ResNet** (Residual Network) – це тип штучної нейронної мережі, яка реалізує концепцію скорочених з'єднань ідентичності (identity shortcut connections), які дозволяють пропускати один або декілька шарів під час передачі інформації. Такий підхід значно покращує здатність моделі до навчання глибоких мереж, зменшує ймовірність зникнення градієнтів та сприяє підвищенню ефективності в задачах комп'ютерного зору.

На відміну від класичних CNN, де кожен шар передає свій вихід безпосередньо наступному, ResNet використовує додаткові шляхи – skip-з'єднання, які дозволяють об'єднувати вхідні дані з вихідними результатами залишкових блоків. Це створює альтернативний маршрут для поширення градієнтів, що забезпечує стабільне навчання навіть у дуже глибоких мережах. Завдяки такій структурі шари мережі фактично навчаються залишковим функціям, тобто не повному перетворенню вхідних даних, а лише їхнім відхиленням (залишкам). Якщо певний шар не покращує результат, модель може автоматично зменшити його вплив до нуля, зберігаючи ідентичну трансформацію завдяки пропущеному з'єднанню[298].

[298] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. DOI: https://doi.org/10.1109/CVPR.2016.90

На рис. 6.37 зображено архітектурні елементи ResNet-блоків[299].

Res-блоки типу x-a – це блоки ідентичності, які не змінюють розмірність даних. Res-блоки типу x-b – це згорткові блоки, які можуть змінювати розмірність, наприклад, при переході між шарами з різною кількістю фільтрів або просторовим розміром.

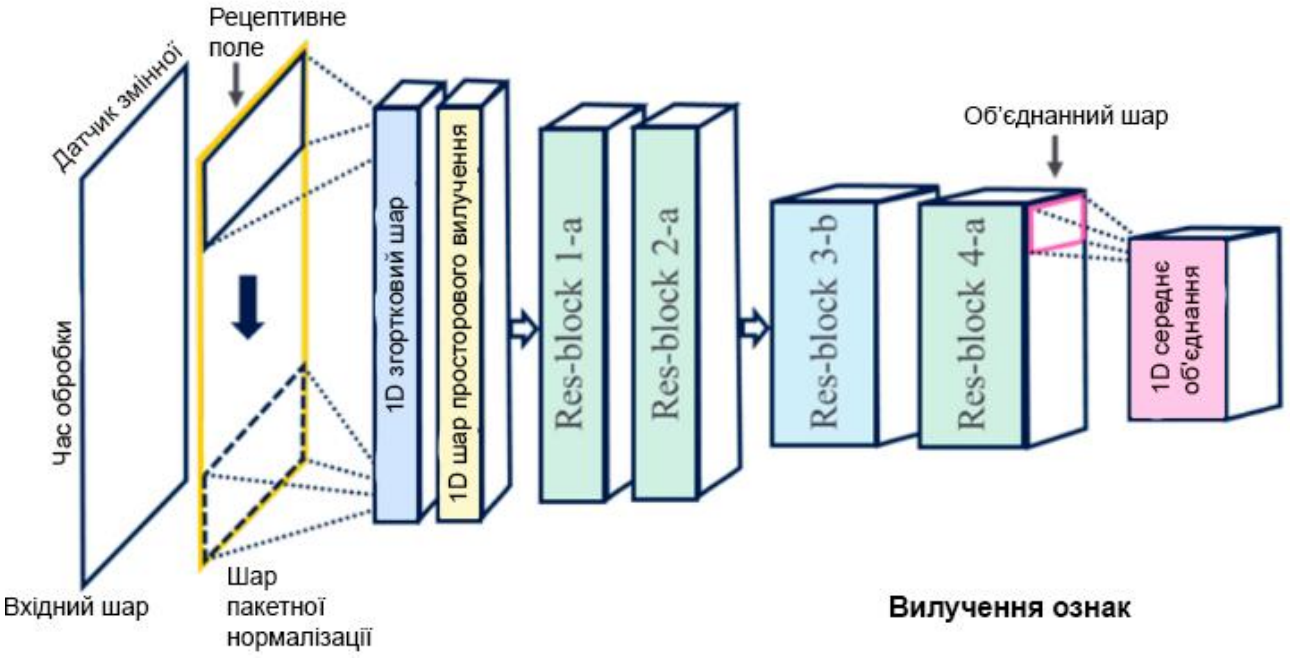


Рисунок 6.37. Архітектура ResNet

Основними елементами ResNet є:

– залишкові блоки (Residual Blocks): забезпечують ефективну передачу інформації через глибину мережі, обходячи деякі шари. Завдяки цьому модель зберігає цілісність сигналу навіть у мережах із сотнями шарів;

– згорткові шари та функції активації: кожен залишковий блок містить згорткові шари з ядрами 3×3, які забезпечують витягування ознак, та функцію активації ReLU, яка додає нелінійність у модель, дозволяючи вловлювати складні залежності у даних;

– нормалізація та стабілізація навчання: для підвищення стабільності й швидкості навчання ResNet використовує метод пакетної нормалізації (Batch Normalization), який нормалізує виходи шарів і зменшує внутрішньошарове зміщення;

– масштабованість: ResNet легко адаптується до завдань різної складності. Наприклад, ResNet-18 містить 18 навчальних шарів, ResNet-50 – 50, а ResNet-152 – 152 шари. Така гнучкість дає змогу обирати модель залежно від доступних ресурсів та вимог до точності.

Завдяки зазначеним архітектурним особливостям ResNet стала основою для багатьох сучасних моделей у сфері класифікації зображень, сегментації об’єктів та інших задач комп’ютерного зору.

[299] Tchatchoua P, Graton G, Ouladsine M, Christaud J-F. Application of 1D ResNet for Multivariate Fault Detection on Semiconductor Manufacturing Equipment. *Sensors*. 2023. No 23(22):9099. DOI: https://doi.org/10.3390/s23229099

Загалом, розвиток згорткових нейронних мереж від LeNet до ResNet засвідчив значний прогрес у глибокому навчанні, зокрема в галузі комп'ютерного зору. Від простих архітектур з кількома шарами до надглибоких мереж із десятками або сотнями шарів, CNN еволюціонували завдяки впровадженню ключових концепцій – ієрархічного вилучення ознак, нормалізації, регуляризації та структур, які дозволяють ефективно передавати інформацію. Кожна розглянута модель зробила свій внесок у підвищення точності, стійкості та обчислювальної ефективності нейронних мереж, створивши фундамент для подальших інновацій у ШІ.

Алгоритм **YOLO** (You Only Look Once) є одним із провідних підходів до задачі виявлення об'єктів у реальному часі і вирізняється високою швидкістю оброблення й прийнятною точністю. Його назва відображає головну концепцію – одноразове оброблення зображення для одночасного визначення об'єктів та їх просторових координат. На відміну від традиційних двоетапних підходів, де спочатку генеруються регіони-пропозиції, а згодом відбувається їх класифікація, YOLO трактує задачу виявлення як єдину регресійну проблему[300].

Перша версія моделі, YOLOv1, базувалася на архітектурі, подібній до GoogLeNet, і містила 24 згорткові та 2 повнозв'язані шари. Згорткові шари відповідали за вилучення просторових ознак, а повнозв'язані реалізовували регресійну частину – прогноз координат та класів об'єктів. З метою покращення ефективності навчання було використано активаційну функцію Leaky ReLU (для запобігання «вмиранню» нейронів) і Dropout-регуляризацію після першого повнозв'язаного шару для зменшення ризику перенавчання (рис. 6.38)[301].

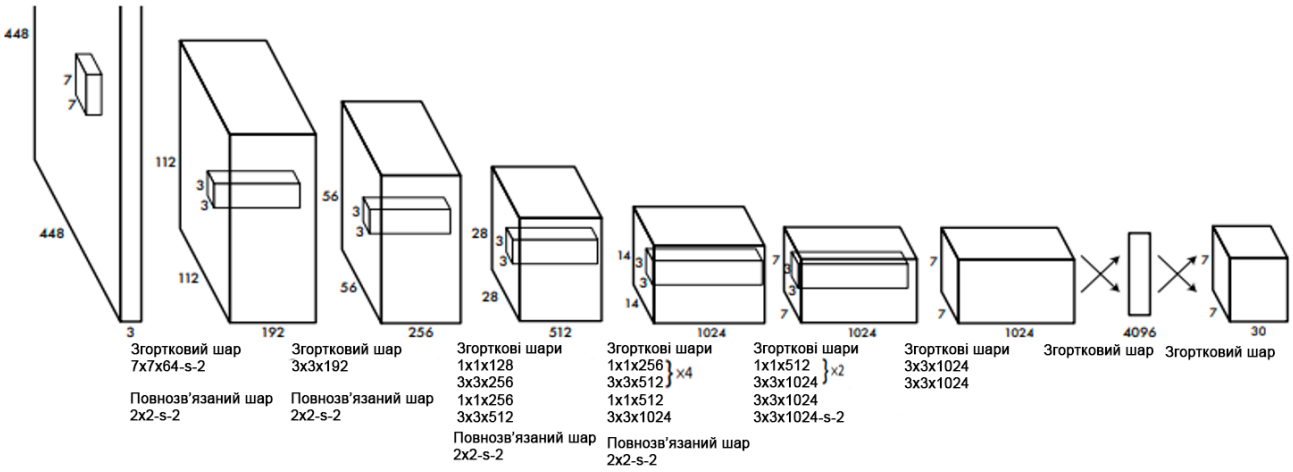


Рисунок 6.38. Архітектура YOLO1

На вхід моделі подається зображення, яке обробляється згортковою нейронною мережею для вилучення ознак. Далі ці ознаки надходять до повнозв'язаних шарів, які здійснюють одночасне прогнозування координат обмежувальних рамок (за параметрами x, y, w, h – координати центру, ширина та висота) і ймовірностей належності до певного класу. Зображення розбивається на сітку

[300] Khanam R., Hussain M. What is YOLOv5: A deep look into the internal features of the popular object detector. arXiv preprint arXiv:2407.20892, 2024•arxiv.org https://doi.org/10.48550/arXiv.2407.20892

[301] Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. 2016 *IEEE Conference on Computer Vision and Pattern Recognition (CVPR*). DOI: 10.1109/CVPR.2016.91

розміром $S \times S$, де кожна клітинка відповідає за прогноз кількох рамок та векторів класових ймовірностей. Для усунення надмірного перекриття рамок використовується алгоритм немаксимального придушення (NMS), який дозволяє залишати лише найбільш релевантні передбачення. Попри чутливість до масштабу об’єкта та обмежену ефективність у розпізнаванні дрібних об’єктів, YOLO має високу продуктивність, що робить його придатним для застосувань у режимі реального часу (рис. 6.39).

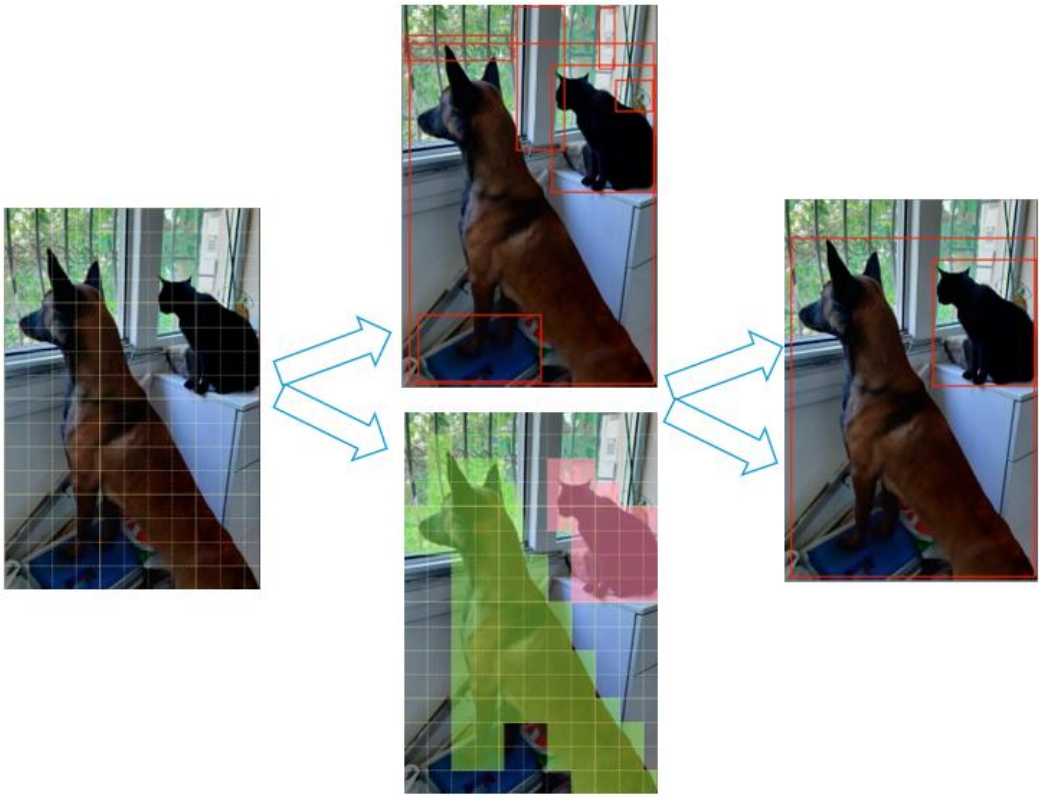


Рисунок 6.39. Принцип роботи YOLO

Попри свою високу швидкість та здатність працювати в режимі реального часу, YOLOv1 має низку недоліків:

– обмеження узагальнення – модель демонструє знижену точність при розпізнаванні нових або раніше невідомих об’єктів;

– просторові обмеження – кожна клітинка сітки може передбачити лише два об’єкти і лише один клас для кожної рамки, що ускладнює розпізнавання дрібних об’єктів або скупчень;

– недосконаліть функції втрат – помилки у визначенні координат об’єктів розглядаються однаково незалежно від розміру рамки, що призводить до невиправдано високих штрафів за дрібні помилки у великих об’єктах;

– похибки локалізації – модель іноді неточно визначає положення об’єкта в межах рамки.

Протягом років YOLO зазнала значної трансформації, поступово вдосконалюючись за всіма ключовими параметрами – точністю, швидкістю та ефективністю використання ресурсів (табл. 6.2).

Таблиця 6.2 – Характеристика версій YOLO

Версія	Рік	Ключові особливості
YOLOv1	2016	Розділення зображення на сітку S×S, один етап передбачення. Висока швидкість, але низька точність на дрібних об'єктах
YOLOv2 (YOLO9000)	2017	Покращена точність, використання попереднього навчання (ImageNet), підтримка >9000 класів, нова архітектура Darknet-19
YOLOv3	2018	Використання багаторівневої детекції (multi-scale), залишкових зв'язків (residuals), Darknet-53. Покращена точність при збереженні швидкості
YOLOv4	2020	Впровадження сучасних технік (Mish-активація, багатошарова агрегація ознак, DropBlock). Баланс між швидкістю та точністю. Працює на звичайному обладнанні
YOLOv5	2020	Реалізована в PyTorch. Висока модульність, підтримка мобільних пристроїв, автовибір гіперпараметрів, проста інтеграція
YOLOv6	2022	Оптимізована для індустріального застосування, підтримка TensorRT, ефективна архітектура backbone + neck, точність на рівні SOTA
YOLOv7	2022	Уніфіковане навчання для класифікації, детекції та сегментації. Оптимізований inference, підтримка легких моделей
YOLOv8	2023	Нова архітектура, підтримка ONNX, TensorRT, інтеграція трансформерів, вища точність та швидкість. Гнучкість у навчанні та розгортанні
YOLOv9	2023	Покращене кодування ознак, баланс між точністю та обчислювальною ефективністю, орієнтація на edge-пристрої
YOLOv10	2024	Впровадження EfficientRep, покращений інкапсульований детектор. Підтримка швидкого навчання та легкого розгортання
YOLOv11	2024	Орієнтація на мобільній embedded-системи. Удосконалення attention-механізмів, зменшення ваги моделі
YOLOv12	2025	Інтеграція компонентів RT-DETR, подальша оптимізація latency, підтримка адаптивної роздільної здатності та самообслуговуваного навчання

Від YOLOv1, яка запропонувала інноваційний одностадійний підхід, архітектура поступово перетворилася на гнучку, модульну та оптимізовану систему. YOLOv2 і YOLOv3 підвищили точність і розширили функціональність через глибші мережі та багаторівневу детекцію. YOLOv4 інтегрувала сучасні практики у побудові CNN, що забезпечило баланс між продуктивністю та ресурсною ефективністю. YOLOv5 стала основою для практичного застосування завдяки зручності й підтримці широкого кола пристроїв.

Подальші версії (YOLOv6–YOLOv12) продовжують орієнтуватися на високу швидкодію, точність й ефективність, з особливим акцентом на мобільні й edge-пристрої. У YOLOv10 – YOLOv12 особливо активно застосовуються архітектурні новації, як-от: EfficientRep, адаптивне масштабування, RT-DETR-компоненти, що сприяє зниженню латентності та покращенню якості детекції в реальному часі[302].

YOLOv12 – це новіша модифікація в сімействі моделей YOLO, орієнтована на інтеграцію ефективних механізмів уваги з метою підвищення точності виявлення об'єктів без втрати продуктивності в режимі реального часу. На відміну

[302] Jegham N., Koh Ch., Abdelatti M., Hendawi A.YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions. *arXiv*. 2025. Vol. 2411.00201v3. URL: https://arxiv.org/pdf/2411.00201v3

від попередніх версій, які переважно базувалися на класичних згорткових архітектурах, YOLOv12 впроваджує модуль регіональної (областної) уваги, що дозволяє моделі фокусуватися на релевантних частинах зображення, зменшує при цьому обчислювальні витрати за рахунок сегментованого оброблення карти ознак. Такий підхід забезпечує збереження широкого рецептивного поля при значному зниженні складності, характерної для традиційної повної самоуваги.

Крім того, YOLOv12 використовує архітектуру R-ELAN (Residual Efficient Layer Aggregation Network), яка сприяє ефективнішій агрегації ознак та покращенню стабільності навчання, зокрема для моделей з великою кількістю параметрів. Ця архітектура забезпечує:

– поглиблене поєднання ознак з різних рівнів абстракції, що дозволяє моделі краще моделювати контекстні взаємозв'язки;

– усунення проблем з затуханням градієнта, що забезпечує ефективніше навчання глибоких мереж;

– оптимізоване об'єднання ознак, яке сприяє підвищенню точності локалізації та класифікації об'єктів (рис. 6.40).

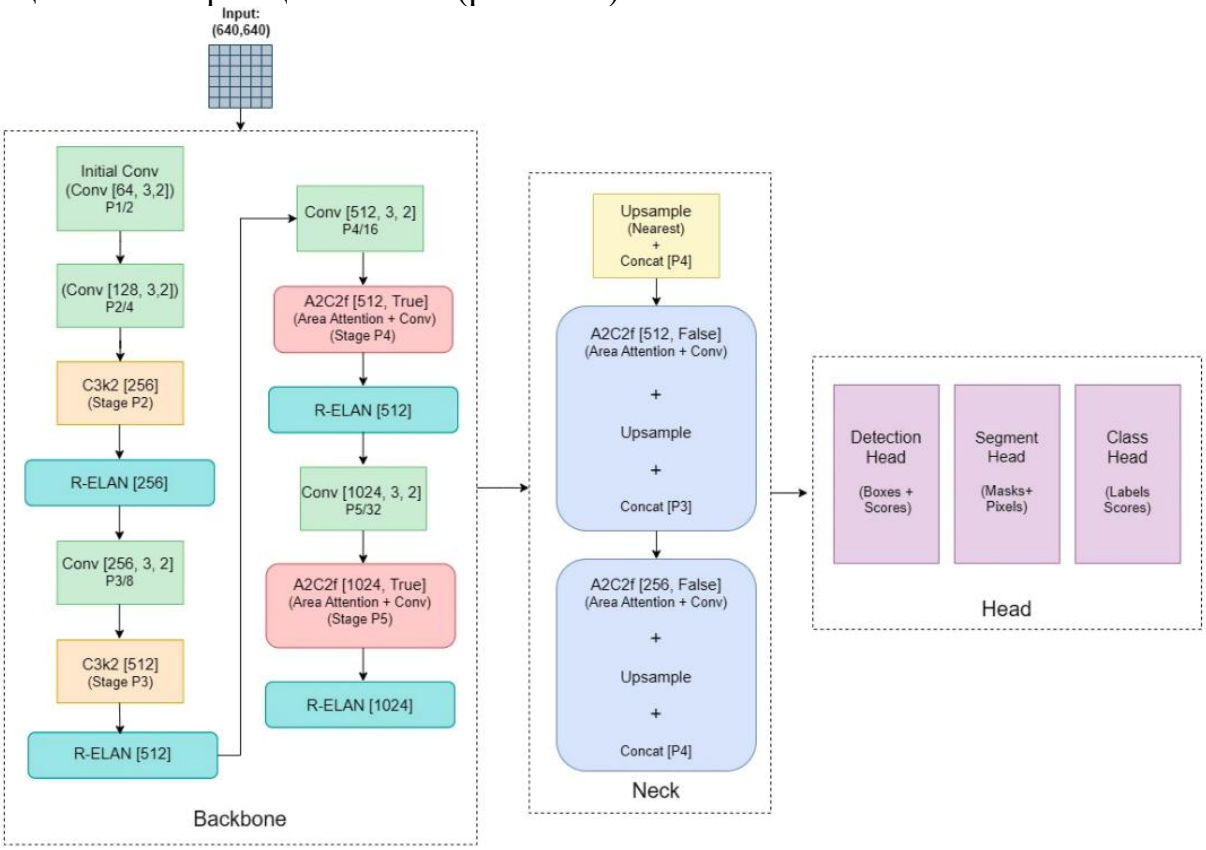


Рисунок 6.40. Архітектура YOLO12

Основними компонентами архітектури YOLO12 є: Backbone (Swin Transformer v2); Neck (PANet + BiFPN) та Head (Dynamic Head)[303].

Backbone – виконує первинне оброблення вхідного зображення та витягує просторові та семантичні ознаки (features), які є основою для подальшого аналізу.

Swin Transformer v2 – це покращена версія віконного трансформера, яка працює з локальними областями (вікнами), зберігає при цьому глобальний

[303] A Simple YOLOv12 Tutorial: From Beginners to Experts. URL: https://so-development.org/a-simple-yolov12-tutorial-from-beginners-to-experts

контекст через зсуви вікон та забезпечує ієрархічне подання ознак, що дозволяє обробляти як дрібні, так і великі об'єкти на зображенні. Завдяки механізму локалізованої уваги Swin Transformer v2 знижує обчислювальні витрати, порівняно з класичними Vision Transformers, а також підтримує велике рецептивне поле та високу точність, що важливо для складних об'єктів і сцен.

Neck (PANet + BiFPN) – агрегує ознаки з різних рівнів глибини (масштабів) та покращує їх перед передачею на фінальний шар (head), щоб модель могла ефективно виявляти об'єкти різного розміру.

PANet (Path Aggregation Network) – покращує потік інформації з нижчих шарів (низькорівневі ознаки) до вищих. Забезпечує збереження дрібних деталей, необхідних для точного виявлення дрібних об'єктів.

BiFPN (Bidirectional Feature Pyramid Network) – реалізує двосторонній потік ознак: як зверху вниз (top-down), так і знизу догори (bottom-up). Має вбудований механізм зважування важливості ознак, що дозволяє мережі самостійно визначати, які ознаки є релевантними на кожному рівні. BiFPN оптимізовано для ефективної агрегації з низькою затратою обчислювальних ресурсів.

Head – компонент, який виконує завдання виявлення об'єктів: класифікацію, регресію обмежувальних рамок та (за потреби) додаткові завдання, як-от сегментація.

Dynamic Head поєднує декілька гілок (attention, convolution, task-specific heads), що дозволяє йому гнучко адаптуватися до різних завдань. Він використовує динамічну агрегацію ознак, що дозволяє покращити точність виявлення за рахунок кращого врахування контексту. Dynamic Head підтримує однопрохідне (one-stage) виявлення, що робить його швидким і придатним для застосувань у реальному часі.

Кожен з компонентів YOLOv12 є модульним та налаштовуваним. Можна окремо вибирати різні версії трансформерів чи CNN як backbone. Neck може бути спрощений або ускладнений залежно від ресурсів. Head можна адаптувати до конкретного завдання: лише виявлення, виявлення + сегментація, багатокласове виявлення тощо.

YOLOv12 показує суттєвий прогрес у розв'язанні проблем, притаманних попереднім версіям, як-от низька точність при виявленні дрібних об'єктів та висока обчислювальна вартість. Завдяки поєднанню ефективної уваги і покращеної агрегації ознак, модель здатна забезпечувати високу якість розпізнавання об'єктів різного розміру за умов обмежених обчислювальних ресурсів, що робить її придатною для широкого спектра застосувань у сфері комп'ютерного зору.

Отже, модель YOLO пройшла шлях від базового рішення до високопродуктивної, універсальної платформи для детекції об'єктів, яка поєднує найсучасніші досягнення глибинного навчання. Постійне вдосконалення дозволяє YOLO залишатися одним із найпопулярніших інструментів комп'ютерного зору.

Модель **SSD** (Single Shot MultiBox Detector) є високоефективним алгоритмом виявлення об'єктів у реальному часі, здатним ідентифікувати кілька об'єктів на зображенні з високою точністю. Вона ґрунтується на глибокій згортковій нейронній мережі, яка приймає зображення як вхідні дані й одночасно виконує задачі локалізації та класифікації об'єктів, що й зумовлює назву «однокадровий детектор».

На відміну від інших моделей, SSD забезпечує одночасне виконання обох завдань у межах одного прямого проходу по мережі, що значно підвищує її швидкодію. Це досягається шляхом використання послідовності згорткових шарів зі зменшеною просторовою роздільною здатністю, що дозволяє виявляти об'єкти різного розміру та з різним співвідношенням сторін[304].

Архітектура SSD побудована на основі згорткової нейронної мережі VGG-16 (рис. 6.35), однак без використання повнозв'язаних шарів. Вибір саме VGG-16 зумовлений її високою ефективністю у задачах класифікації зображень та широким застосуванням у задачах перенавчання (transfer learning). Замість повнозв'язаних шарів до моделі були додані допоміжні згорткові шари, починаючи з шару conv6, які дозволяють витягувати ознаки на різних масштабах та поступово зменшувати розмірність вхідних зображень (рис. 6.41).

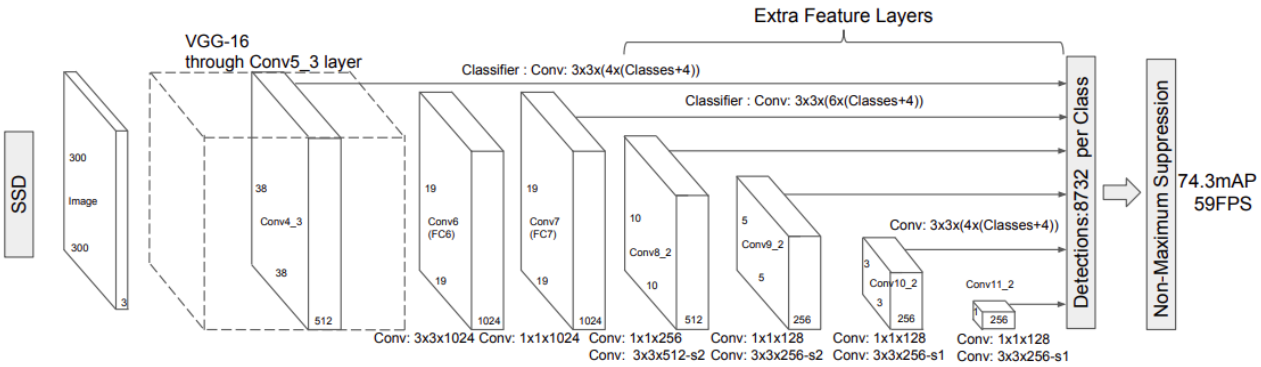


Рисунок 6.41. Архітектура SSD

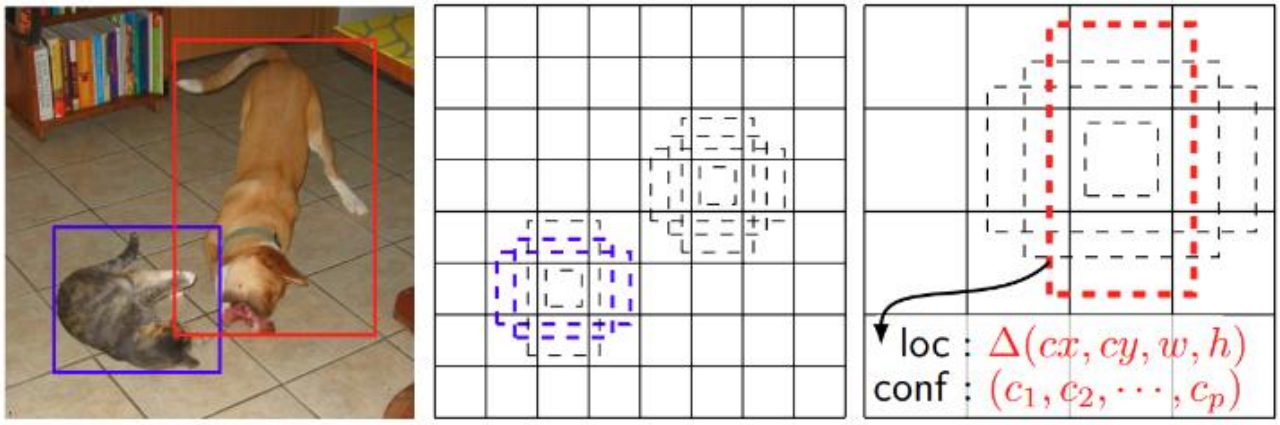
SSD використовує багатомасштабні карти ознак, що забезпечує ефективне виявлення об'єктів різного розміру. Для цього модель здійснює прогнозування з шести згорткових шарів (Conv4_3, Conv7, Conv8_2, Conv9_2, Conv10_2, Conv11_2), кожен з яких має різну роздільну здатність – від 38×38 до 1×1. Результати з усіх цих шарів агрегуються в єдиний блок виявлення. Загалом з цих карт ознак формується 8732 обмежувальні рамки.

Замість безпосереднього прогнозування координат обмежувальних рамок модель визначає усталене значення зміщення (offsets) відносно рамок (default boxes або anchor boxes). Кожна клітинка на карті ознак усталено пов'язана з кількома рамками з різними співвідношеннями сторін, а для кожної з них модель прогнозує чотири параметри зміщення та ймовірності належності до кожного з класів.

Наприклад, карта ознак має розміри $m \times n$ та p каналів. На неї накладається ядро згортки розміром 3×3 з відповідною кількістю каналів. У результаті для кожної позиції $m \times n$ генеруються локальні прогнози у вигляді зміщень та класових ймовірностей. Для c класів та k рамок усталено у кожній клітинці сітки вихід для цієї карти ознак має форму $(c+4) \times k \times m \times n$ (рис. 6.42)[305].

[304] Understanding Single Shot Detector (SSD). URL: https://surl.lu/cvtagv

[305] Liu W. et al. Ssd: Single shot multibox detector. *European conference on computer vision*. Cham : Springer International Publishing, 2016. P. 21-37. DOI: https://doi.org/10.48550/arXiv.1512.02325



а) *б)* *в)*

Рисунок 6.42. SSD у різних масштабах карт ознак
а – зображення з блоками; б – карта ознак 8x8; в – карта ознак 4x4

Отже, модель SSD забезпечує ефективне, масштабоване виявлення об'єктів за рахунок поєднання інформації з кількох рівнів глибини нейронної мережі, що дозволяє досягати конкурентної точності при збереженні високої швидкодії, необхідної для застосувань у реальному часі.

Модель **Faster R-CNN** (Faster Region-Convolutional Neural Network), створена 2015 року, попри тривалий час після її появи, залишається однією з еталонних архітектур для розв'язання завдань комп'ютерного зору. Її основне призначення полягає в ідентифікації та точному локалізуванні об'єктів на зображеннях або у відеопотоці, що є ключовою проблемою в області розпізнавання візуальної інформації.

Faster R-CNN є еволюційним продовженням архітектури R-CNN та її вдосконалених варіантів (Fast R-CNN), яке поєднує в собі ефективність згорткових нейронних мереж (CNN) із високошвидкісною генерацією регіонів-пропозицій завдяки впровадженню мережі пропозицій регіонів (Region Proposal Network, RPN). Саме тому ця модель забезпечує значне зменшення часу оброблення зображень і підвищену точність розпізнавання об'єктів. Архітектура дозволяє одночасно виконувати завдання виявлення об'єктів та уточнення їх локалізації у межах єдиної нейронної мережі[306].

Архітектура Faster R-CNN складається з двох компонентів: мережі пропозицій регіонів (Region Proposal Network, RPN) та швидкого детектора R-CNN (рис. 6.43)[307].

[306] Ren S., He K., Girshick R., Sun J. Faster R-CNN: towards real-time object detectionwith region proposal networks, *IEEE Trans. Pattern. Anal. Mach. Intell*. 2016. Vol. 39 (6). P. 1137–1149. DOI: https://doi.org/10.1109/tpami.2016.2577031

[307] Zhipeng D., Hao S., Shilin Z., Juanping Z., Lin L., Huanxin Z. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2018. Vol. 145. Part A. P. 3-22. DOI: https://doi.org/10.1016/j.isprsjprs.2018.04.003.

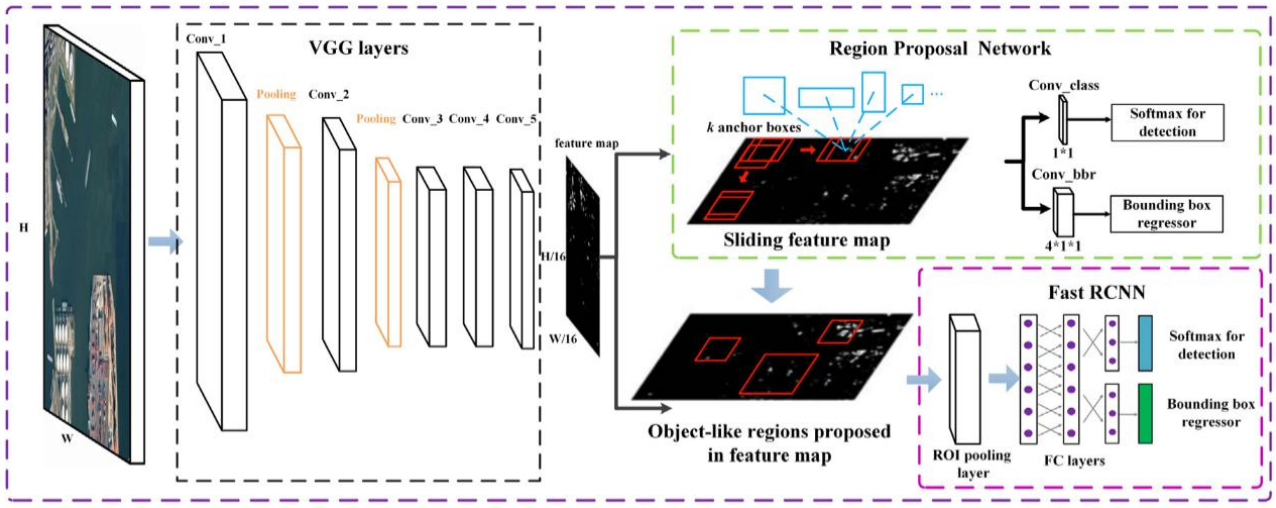


Рисунок 6.43. Архітектура Faster R-CNN

RPN – генерує регіони-кандидати, які ймовірно містять об'єкти.

Fast R-CNN – класифікує ці області та виконує регресію координат обмежувальних рамок (bounding boxes).

Вхідне зображення спочатку обробляється глибокою згортковою мережею (наприклад, ResNet або VGG), яка генерує карту ознак. Ця карта передається до RPN, яка, скануючи зображення з використанням множини якорів (anchor boxes) різних масштабів та пропорцій, формує регіони-пропозиції. Далі застосовується шар об'єднання регіонів інтересу (Region of Interest Pooling, RoI Pooling), який нормалізує розмір регіонів для подальшого оброблення. Нарешті, кожна пропозиція аналізується класифікатором, який визначає клас об'єкта, а також виконується регресія координат для уточнення локалізації.

RPN приймає на вхід карту ознак та створює якорі – прямокутні області різних розмірів і форм, які покривають зображення. Для кожного якоря RPN прогнозує:

– ймовірність належності до об'єкта (класифікація) – визначення, чи містить якір об'єкт, чи він є фоновим;

– зміщення координат (регресія) – коригування якоря для кращої відповідності дійсному об'єкту.

Тут використовуються дві функції втрат: втрата класифікації (classification loss), яка використовується для визначення наявності об'єкта та втрата регресії (regression loss), використовується для точнішого позиціонування рамки.

Отримані від RPN регіони-кандидати мають довільні розміри, тому для їх оброблення застосовується RoI Pooling, який перетворює кожну область до фіксованого розміру. Потім кожна така область обробляється невеликою нейронною мережею, яка виконує класифікацію (визначення класу об'єкта) та здійснює регресію координат для уточнення обмежувальної рамки.

Для класифікації використовується функція втрат на основі перехресної ентропії, а для уточнення координат – регресійна втрата (smooth L1 loss).

Під час інференсу модель генерує велику кількість пропозицій, з яких вибираються найкращі за допомогою методу Non-Maximum Suppression (NMS), що дозволяє усунути дублікати та зберегти лише найінформативніші рамки.

Є два основних підходи до навчання Faster R-CNN:

– поетапне навчання (alternating training) – спочатку навчається RPN, потім класифікатор;

– спільне навчання (joint training) – усі компоненти навчаються одночасно.

Попри високу ефективність, Faster R-CNN має низку обмежень. Наприклад, під час навчання усі якорі в межах одного міні-батчу зазвичай походять з одного зображення, що знижує різноманіття навчальних прикладів і може уповільнювати збіжність. Проте загалом архітектура залишається потужною основою для задач виявлення об'єктів і стала фундаментом для подальших удосконалень, зокрема для Mask R-CNN, яка розширює можливості Faster R-CNN, додаючи модуль для сегментації об'єктів на рівні пікселів.

Mask R-CNN (Mask Region-based Convolutional Neural Network) – це розширення архітектури Faster R-CNN, яке додає гілку для прогнозування масок сегментації на кожній області інтересу (RoI), паралельно з наявною гілкою для класифікації та регресії обмежувальної рамки.

Гілка маски – це невелика FCN, яка застосовується до кожної RoI, прогнозуючи маску сегментації піксельним способом. Mask R-CNN проста у реалізації та навчанні завдяки фреймворку Faster R-CNN, який сприяє широкому спектру гнучких архітектурних проєктів (рис. 6.44)[308].

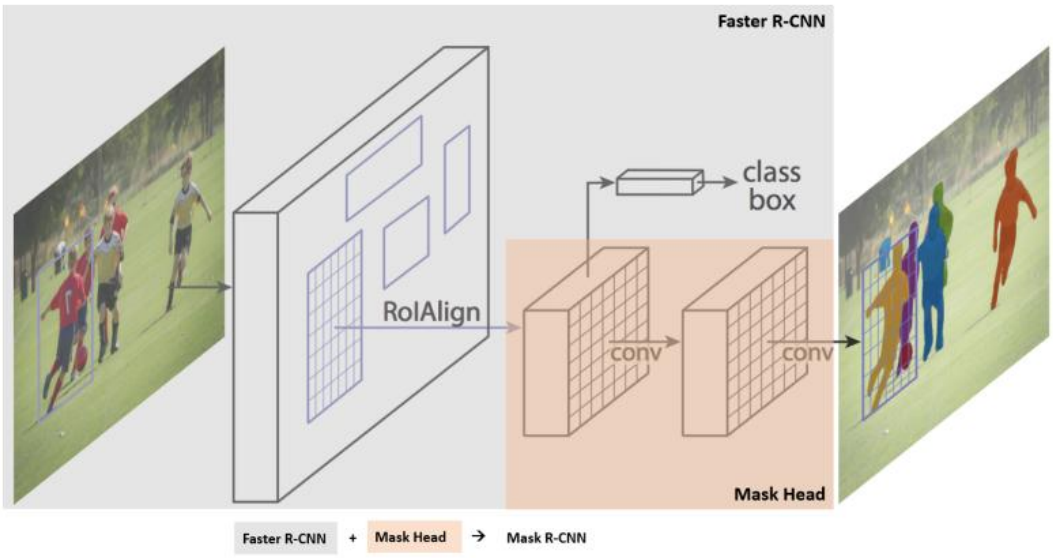


Рисунок 6.44. Архітектура Mask R-CNN

Ця модель відноситься до класу двостадійних методів, де спочатку на зображенні генеруються RoI, в яких можуть утримуватися об'єкти, що цікавлять. Потім, кожен регіон-кандидат відноситься до того чи іншого класу, уточнюється розташування рамки, яке містить його та передбачається маска, що виділяє об'єкт. Її складовими є:

– магістральна мережа (Backbone Network);

– мережа пропозицій регіонів (RPN);

– класифікація областей;

[308] He K., Gkioxari G., Dollar P., Girshick R. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017. P. 2961-2969 URL: https://surli.cc/izfchk

– маски сегментації.

Магістральна мережа – це нейронна мережа, яка є екстрактором функцій. Наприклад, вона перетворить зображення 1024x1024x3 на карту функцій 32x32x2048, яка вводиться для наступних шарів.

RPN, використовуючи регіони, визначені з 200-кілобайтними якорями, RPN сканує кожен регіон і передбачає, чи є об'єкт чи ні. Однією з переваг RPN є те, що не сканується конкретне зображення, а мережа сканує карту функцій, роблячи це набагато швидше.

Класифікація областей інтересу та граничне вікно – на цьому кроці алгоритм приймає області інтересу, запропоновані RPN як вхід і вихід, класифікацію (софтмакс) та граничне вікно (регресор).

Маски сегментації – це останній крок, де алгоритм приймає як входи позитивні області RoI, а як виходи для об'єктів генеруються маски 28x28 пікселів зі значеннями рухомої крапки. Під час виведення ці маски масштабуються.

Перевагами Mask R-CNN є:

– точна сегментація екземплярів: Mask R-CNN чудово забезпечує точні маски сегментації на рівні пікселів для кожного виявленого об'єкта;

– універсальність: можна обробляти кілька завдань в одному фреймворку;

– завдяки включенню RPN, Mask R-CNN ефективно генерує пропозиції регіонів, що дозволяє йому зосередитися на відповідних регіонах та значно зменшує обчислювальне навантаження порівняно з підходами сканування;

– гнучка архітектура;

– продуктивність: Mask R-CNN постійно досягає найкращих результатів у тестах сегментації екземплярів.

Попри переваги, Mask R-CNN має й низку недоліків:

– гілка прогнозування масок збільшує обчислювальне навантаження;

– Mask R-CNN зазвичай вимагає значних обчислювальних ресурсів, таких як графічні процесори;

– проблеми з анотацією даних – модель трудомістка та складна у створенні для певних доменів;

– є обмеження застосування в реальному часі – Mask R-CNN може не підходити для застосувань у реальному часі із суворими вимогами до затримки[309].

Отже, Mask R-CNN є потужною та ефективною архітектурою для задач сегментації екземплярів, яка поєднує точність виявлення об'єктів із сегментацією на рівні пікселів. Вона є логічним розвитком Faster R-CNN, зберігає його модульність і гнучкість, але додає нову гілку для прогнозування масок, що дозволяє отримувати детальну інформацію про форму об'єктів.

У цілому Mask R-CNN є ефективним інструментом для досліджень і практичних застосувань у комп'ютерному зорі, однак її використання потребує відповідного апаратного забезпечення та належної підготовки даних.

[309] Mask R-CNN | ML. URL: https://www.geeksforgeeks.org/machine-learning/mask-r-cnn-ml

6.4 Висновки до розділу

Комп'ютерний зір є однією з ключових галузей ШІ, яка забезпечує комп'ютерам здатність автоматично аналізувати, інтерпретувати та осмислювати візуальну інформацію. Завдяки поєднанню класичних методів цифрового оброблення зображень і сучасних технологій глибокого навчання, системи комп'ютерного зору успішно розв'язують завдання класифікації, виявлення, сегментації та розпізнавання об'єктів.

Значний прорив у цій сфері досягнутий завдяки використанню згорткових нейронних мереж, зокрема моделі AlexNet, яка продемонструвала високу ефективність у задачах класифікації зображень. Подальший розвиток технологій, як-от: YOLO, Faster R-CNN, Mask R-CNN та ін., забезпечив можливість виконання детекції та сегментації об'єктів у режимі реального часу з високою точністю.

Класичні методи (детектори країв, гістограми, колірні простори, дескриптори ознак) залишаються актуальними в задачах з обмеженими обчислювальними ресурсами або коли потрібна інтерпретованість алгоритмів. Водночас методи глибокого навчання забезпечують вищу точність, гнучкість та адаптивність до складних і змінних візуальних середовищ.

Перспективи досліджень у галузі комп'ютерного зору охоплюють низку актуальних напрямів. Одним із ключових є удосконалення алгоритмів детектування, класифікації, сегментації та розпізнавання об'єктів. Подальші наукові дослідження можуть бути зосереджені на підвищенні точності та швидкодії таких алгоритмів, особливо в умовах обмежених навчальних даних або під час роботи в реальному часі. Важливим напрямом залишається розроблення нових архітектур нейронних мереж. Оскільки згорткові нейронні мережі є основою сучасних систем комп'ютерного зору, актуальними є дослідження, спрямовані на створення більш ефективних, компактних та енергоощадних моделей, придатних для застосування на мобільних пристроях і вбудованих системах.

Значний науковий інтерес становить інтеграція комп'ютерного зору з іншими технологіями, зокрема доповненою і віртуальною реальністю, робототехнікою та IoT, що відкриває можливості для створення інтелектуальних систем.

Використання технологій комп'ютерного зору в автоматизації тестування програмного забезпечення впливає на широкий спектр повсякденних процесів, оскільки воно орієнтоване на вдосконалення рутинних завдань, зниження кількості помилок, підвищення рівня надійності й якості, а також на суттєве зростання загальної продуктивності. Застосування таких методів, як класифікація зображень, виявлення та відстеження об'єктів, а також пошук інформації за візуальним вмістом, докорінно змінило концептуальні підходи до автоматизації тестування програмних систем. Ці технології дозволяють не лише прискорювати процес перевірки функціональності та зменшувати витрати часу, а й забезпечують вищий рівень точності аналізу, що особливо важливо за умов зростання складності сучасних програмних продуктів.

Тому інтеграція інструментів комп'ютерного зору в системи автоматизованого тестування стає ключовим етапом на шляху до глибокої оптимізації процесів розроблення програмного забезпечення. Це створює передумови для

гіперавтоматизації, коли поєднання ШІ, машинного навчання та комп’ютерного зору забезпечує максимізацію результативності, ефективне використання ресурсів та досягнення високої якості кінцевого продукту без компромісів у надійності. Цей підхід можна розглядати як стратегічний чинник розвитку сучасних технологічних систем, який забезпечує конкурентні переваги на ринку та формує нові стандарти у сфері тестування програмного забезпечення.

Розширення сфер застосування комп’ютерного зору є ще одним перспективним напрямом, що передбачає його впровадження в нові галузі. Окрему увагу заслуговує оптимізація методів попереднього оброблення вхідних даних. Зокрема, актуальними є дослідження, присвячені розробленню нових способів перетворення кольорових просторів та інших методик, здатних підвищити ефективність функціонування моделей ШІ.

Список використаних та рекомендованих джерел

1. 10 Best SEO Copywriting Tools for Founders [2025 Updated]. URL: https://founderpal.ai/best-marketing-tools/seo-copywriting-tools
2. 20+ найкращих AI-інструментів 2024 для дизайнера. URL: https://www.komarov.design/20-naikrashchikh-ai-nstrumientiv-2024/
3. 27 Eye-Opening Statistics About User Experience, Website First Impressions, and Website Design That You Should Be Taking Very Seriously. URL: https://www.sweor.com/firstimpressions
4. 3D Computer Vision: A Comprehensive Guide. URL: https://viso.ai/computer-vision/3d-computer-vision/
5. A METR Study Reveals that AI Slows Down Experienced Developers. 2025. URL: https://www.actuia.com/en/news/a-metr-study-reveals-that-ai-slows-down-experienced-developers/
6. A Perspective on Explainable Artificial Intelligence Methods: SHAP and LIME. *arXiv*. 2024. URL: https://arxiv.org/html/2305.02012v3
7. A Simple YOLOv12 Tutorial: From Beginners to Experts. URL: https://so-development.org/a-simple-yolov12-tutorial-from-beginners-to-experts
8. Abadi M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv*. 2016. Vol. 1603.04467. DOI: https://doi.org/10.48550/arXiv.1603.04467
9. Abbes M., Khomh F., Guéhéneuc Y.-G., Antoniol G. An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, On Program Comprehension. *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. 2011. P. 181-190. DOI: https://doi.org/10.1109/CSMR.2011.24
10. Accelerate AWS Well-Architected reviews with Generative AI. *Amazon.* URL: https://aws.amazon.com/blogs/machine-learning/accelerate-aws-well-architected-reviews-with-generative-ai/
11. Adithya K. Anomaly Detection with Isolation Forest & Visualization. Towards Data Science. URL: https://towardsdatascience.com/anomaly-detection-with-isolation-forest-visualization-23cd75c281e2
12. AI Development Cost: Analyzing Expenses and Returns. *TechMagic*. URL: https://www.techmagic.co/blog/ai-development-cost
13. AI in the workplace: A report for 2025. *McKinsey*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ais-full-potential-at-work
14. AI use in video game development - statistics & facts. *Statista*. URL: https://statista.com/topics/13437/artificial-intelligence-ai-use-in-video-game-development/

15. AI Workload Documentation. *Microsoft Azure Well-Architected Framework. Microsoft Learn*. URL: https://learn.microsoft.com/en-us/azure/well-architected/ai/
16. AI/ML orchestration on GKE documentation. *Google Cloud.* URL: https://cloud.google.com/kubernetes-engine/docs/integrations/ai-infra
17. Air India's digital transformation takes flight with Microsoft Azure. *Microsoft Customer Stories*. URL: https://www.microsoft.com/en/customers/story/23774-tata-sia-airlines-limited-azure
18. AIVA Technology – Composing Music using AI. URL: https://d3.harvard.edu/platform-digit/submission/aiva-technology-composing-music-using-ai/
19. Alenezi M, Akour M. AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Diections. *Applied Sciences.* 2025. Vol. 15(3). DOI: https://doi.org/10.3390/app15031344
20. Al-Hawawreh M., Aljuhani A., Jararweh Y. ChatGPT for cyberse-curity: practical applications, challenges, and future directions. *ClusterComputing*. 2023. Vol. 26, № 6. P. 3421-3436. DOI: https://doi.org/10.1007/s10586-023-04124-5
21. Amazon Augmented AI Resources. URL: https://aws.amazon.com/augmented-ai/resources/
22. Amazon Q Developer. URL: https://aws.amazon.com/ru/q/developer/
23. Aniche M., Bavota G., Treude C. et al. Code smells for Model-View-Controller architectures. *Empir Software Eng*. 2018. Vol. 23. P. 2121-2157. DOI: https://doi.org/10.1007/s10664-017-9540-2
24. Announcing the AWS Well-Architected Generative AI Lens. *AWS Architecture Blog*. URL: https://aws.amazon.com/blogs/architecture/announcing-the-aws-well-architected-generative-ai-lens/
25. Antal G., Vozár R., Ferenc R. Toward a New Era of Rapid Development: Assessing GPT-4-Vision's Capabilities in UML-Based Code Generation. *arXiv*. 2024. Vol. 2404.14370. DOI: https://doi.org/10.48550/arXiv.2404.14370
26. Application Design for AI Workloads on Azure. *Microsoft Azure Well-Architected Framework. Microsoft Learn*. URL: https://learn.microsoft.com/en-us/azure/well-architected/ai/application-design
27. Architectural Patterns Consolidation. *Thoughtworks*. *Technology Radar*. 2025. Vol. 32. URL: https://www.thoughtworks.com/radar/techniques/architectural-patterns-consolidation
28. Artificial Intelligence in Financial Services. Report on the uses, opportunities, and risks of artificial intelligence in the financial services sector. URL: https://home.treasury.gov/system/files/136/Artificial-Intelligence-in-Financial-Services.pdf
29. AutoML.org. Neural Architecture Search Overview. URL: https://www.automl.org/nas-overview
30. Average duration of downtime after a ransomware attack at organizations in the United States from 1st quarter 2020 to 2nd quarter 2022. *Statista.* URL:

https://www.statista.com/statistics/1275029/length-of-downtime-after-ransomware-attack-us/
31. AWS Graviton Processors. *AWS*. URL: https://aws.amazon.com/ec2/graviton/
32. AWS Well-Architected Framework: Designing Efficient Cloud Applications. *DEV Community*. URL: https://dev.to/imsushant12/aws-well-architected-framework-designing-efficient-cloud-applications-38dj
33. Bakal G., Dasdan A., Katz Y., Kaufman M., Levin G. Experience with GitHub Copilot for Developer Productivity at Zoominfo. *arXiv*. 2025. Vol. 2501.13282v1. DOI: https://doi.org/10.48550/arXiv.2501.13282
34. Bansal M., Goyal A., Choudhary A. A comparative analysis of K-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning. *Decision analytics journal*. 2022. Vol. 3. DOI: https://doi.org/10.1016/j.dajour.2022.100071
35. Bâra R., Costin A., Cătălin T. Analysing the performance impacts of lazy loading in web applications. *Journal of Information Systems & Operations Management*. 2024. Vol. 18.1. P. 1-15. URL: http://web.rau.ro/websites/jisom/Vol.18%20No.1%20-%202024/JISOM%2018.1.pdf
36. Barbez A., Khomh F., Guéhéneuc Y.-G. A Machine-learning Based Ensemble Method For Anti-patterns Detection. *Journal of Systems and Software*. 2020. Vol. 161. P. 110486. DOI: https://doi.org/10.1016/j.jss.2019.110486
37. Bas A., Topal M., Duman C., Heerden I. A Brief History of Deep Learning-Based Text Generation. *International Conference on Computer and Applications (ICCA).* 2022. DOI: 10.1109/ICCA56443.2022.10039545
38. Basiri A., Hochstein L., Jones N., Tucker H. Automating chaos experiments in production. *Arxiv.* 2019. URL: https://arxiv.org/pdf/1905.04648
39. Bates A., Vavricka R., Carleton S., Shao R., Pan C. Unified modeling language code generation from diagram images using multimodal large language models. *Machine Learning with Applications*. 2025. Vol. 20. P. 100660. ISSN 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2025.100660
40. Baudry B., Etemadi K., Fang S., Gamage Y., Liu Y., Liu Y. Generative AI to Generate Test Data Generators. *arXiv e-prints*, arXiv-2401. 2024. DOI: https://doi.org/10.48550/arXiv.2401.17626.
41. Best AI Tools for Architects in 2025: A Comprehensive Guide. *ArchEyes*. URL: https://archeyes.com/best-ai-tools-for-architects-in-2025-a-comprehensive-guide/
42. Best AI-Driven SEO Tools for Content Optimization in 2025. URL: https://www.outranking.io/blog/ai-driven-seo-tools-content-optimization/
43. Best practices for implementing machine learning on Google Cloud. *Cloud Architecture Center*. URL: https://cloud.google.com/architecture/ml-on-gcp-best-practices
44. Bezdan T., Džakula N. Convolutional Neural Network Layers and Architectures. *International Scientific Conference on Information Technology*

and Data Related Research. Belgrade, Serbia, 2019. P. 445-451. DOI: https://doi.org/10.15308/Sinteza-2019-445-451
45. Bousquet L., Nakamura M. Improving Testability of Software Systems that Include a Leaning Feature. *The 10th International Conference on Advances in System Testing and Validation Lifecycle.* October, 14-18, 2018, Nice, France. URL: http://www27.cs.kobe-u.ac.jp/achieve/data/pdf/1333.pdf
46. Breck E., Cai Sh., Nielsen E., Salib M., Sculley D. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. *IEEE International Conference on Big Data*. 2017. P. 1123-1132. DOI: https://doi.org/10.1109/BigData.2017.8258038
47. Breckner K., Neumayr T., Mara M., Streit M, Augstein M. The Changing Nature of Human-AI Relations: A Scoping Review on Terminology and Evolvement in the Scientific Literature. *International Journal of Human–Computer Ineraction*. 2025. P.1-58. DOI: https://doi.org/10.1080/10447318.2025.2482742
48. Brito P. Computer Vision – The Importance of Filtering. URL: https://surl.lt/nlnkyo
49. Brown T. et al. Language models are few-shot learners. *Advances in neural information processing systems*. 2020. Vol. 33. P. 1877–1901. DOI: https://doi.org/10.48550/arXiv.2005.14165
50. Building Event-Driven Architecture on AWS. *AWS eBook*. 2024. URL: https://d1.awsstatic.com/psc-digital/2023/gc-300/build-eda-on-aws/Build-EDA-AWS-eBook.pdf
51. Building HIPAA-Compliant AI Applications for Healthcare in 2025. URL: https://mobidev.biz/blog/how-to-build-hipaa-compliant-ai-applications
52. Building Meta's GenAI Infrastructure. *Engineering at Meta.* URL: https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/
53. Chaos Engineering Saved Your Netflix. *IEEE Spectrum*. URL: https://spectrum.ieee.org/chaos-engineering-saved-your-netflix
54. ChatGPT vs. Microsoft Copilot vs. Claude: Full Report and Comparison on Features, Capabilities, Pricing and more (Mid-2025). URL: https://www.datastudios.org/post/chatgpt-vs-microsoft-copilot-vs-claude-full-report-and-comparison-on-features-capabilities-pric
55. Chen Y., Xie H., Ma M., et al. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys'24)*. 2024. P. 674-688. DOI: https://doi.org/10.1145/3627703.3629553.
56. Cheng H., Husen J.H., Lu Y., Racharak T., Yoshioka N., Ubayashi N., Washizaki H. Generative AI for Requirements Engineering: A Systematic Literature Review. *arXiv*. 2024. Vol. 2409.06741. DOI: https://doi.org/10.48550/arXiv.2409.06741
57. Chiappetta M. IBM and AMD Collaborate on Hybrid Quantum-Supercomputing Architectures. *Forbes*. URL:

https://www.forbes.com/sites/marcochiappetta/2025/08/31/ibm-and-amd-collaborate-on-new-hybrid-quantum-supercomputing-architectures
58. Clark A., Walkinshaw N., Hierons R. Test case generation for agent-based models: a systematic literature review. *Information and Software Technology*. 2021. Vol. 135. P. 106567. DOI: https://doi.org/10.1016/j.infsof.2021.106567
59. Claude vs. GPT-4.5 vs. Gemini: A Comprehensive Comparison. URL: https://www.evolution.ai/post/claude-vs-gpt-4o-vs-gemini
60. Combine elements from different images, and other new ways to bring your creativity to life. URL: https://ai.google/
61. Computer Vision – Worldwide. URL: https://surl.lu/qmmvkd
62. Computer Vision in AR and VR – The Complete Guide. URL: https://surl.li/dcsgse
63. Computer Vision Market Summary. URL: https://surl.lu/geeazf
64. Computer Vision Tasks (Comprehensive 2025 Guide). URL: https://viso.ai/deep-learning/computer-vision-tasks/
65. Context7 MCP - Up-to-date Code Docs For Any Prompt. URL: https://github.com/upstash/context7
66. Copilot4DevOps – Your AI Assistant for DevOps. URL: https://marketplace.visualstudio.com/items?itemName=edevtech-mr.CoPilot4DevOps-Standalone-US-01
67. Cost of a data breach 2025. *IBM*. URL: https://www.ibm.com/reports/data-breach
68. CSR vs SSR vs ISR Rendering Patterns in Next.js. URL: https://medium.com/@sureshdotariya/csr-vs-ssr-vs-isr-patterns-in-next-js-c526757e837a
69. CTO's Guide to AI Development Tool ROI. Augment Code. URL: https://www.augmentcode.com/guides/cto-s-guide-to-ai-development-tool-roi
70. Dakowicz Ja. 5 Best Headless CMS Platforms in 2025: Reviews & Comparisons. URL: https://pagepro.co/blog/top-5-best-headless-cms-platforms/
71. Deepika H.C., Srinivasan G.N. An Overview of You Only Look Once: Unified, Real-Time Object Detection. *Internaional Journal for Research in Applied Science & Engineering Technology* (*IJRASET*). 2020. P. 607-609. DOI: https://doi.org/10.22214/IJRASET.2020.6098
72. Demonstrating a true realization of quantum-centric supercomputing. URL: https://www.ibm.com/quantum/blog/supercomputing-24
73. Divyansh B. The AI Model Race: Claude 4 vs GPT-4.1 vs Gemini 2.5 Pro. *Medium*. 2025. URL: https://medium.com/@divyanshbhatiajm19/the-ai-model-race-claude-4-vs-gpt-4-1-vs-gemini-2-5-pro-dab5db064f3e
74. Dobrescu P., Flavia D. AI First. *The New Age of Development. Contributions to Political Science.* 2022. P. 1-43. DOI: https://doi.org/10.1007/978-3-031-05664-2_1
75. Earney Sh. What is Edge Computer Vision, and How Does it Work? URL: https://surl.li/ldnyrd

76. Elham M., Rawaa T., Howraa M. Design and Fundamentals of Sobel Edge Detection of an Image. 2022. Vol. 9. P. 2458-9403. URL: https://www.researchgate.net/publication/359894954_Design_and_Fundamentals_of_Sobel_Edge_Detection_of_an_Image
77. ELITEA – AI Workflow Automation Tool. *EPAM SolutionsHub.* URL: https://solutionshub.epam.com/solution/elitea
78. Emerenini C. Top 10 AI Video Generation Tools for Creative Content. URL: https://techtraversex.hashnode.dev/top-ai-video-generation-tools-for-creative-content
79. Engineering More Reliable Transportation with ML and AI at Uber. *Uber Blog*. URL: https://www.uber.com/en-UA/blog/machine-learning/
80. Esnaashari M., Damia A. H. Automation of software test data generation using genetic algorithm and reinforcement learning. *Expert Systems with Applications*. 2021. Vol. 183. Issue C. P. 115446. DOI: https://doi.org/10.1016/j.eswa.2021.115446
81. Explainable AI: Understanding and Trusting Machine Learning Models. *DataCamp.* URL: https://www.datacamp.com/tutorial/explainable-ai-understanding-and-trusting-machine-learning-models
82. Explainable Artificial Intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA).* URL: https://www.darpa.mil/research/programs/explainable-artificial-intelligence
83. Exploring Uber's Tech Stack & Software Architecture. URL: https://intuji.com/ubers-tech-stack-software-architecture/
84. FDA Issues Comprehensive Draft Guidance for Developers of Artificial Intelligence-Enabled Medical Devices. URL: https://www.fda.gov/news-events/press-announcements/fda-issues-comprehensive-draft-guidance-developers-artificial-intelligence-enabled-medical-devices
85. Fiala S. Comparison of Top AI Image Models: DALL·E 3, Midjourney, Stable Diffusion, Adobe Firefly, Grok, and deogram. URL: https://www.linkedin.com/pulse/comparison-top-ai-image-models-dalle-3-midjourney-stable-sabra-iala-xdxvf/
86. Floridi L. Artificial Intelligence, Deepfakes and a Future of Ectypes. *Ethics, Governance, and Policies in Artificial ntelligence*. *Philosophical Studies Series.* 2021. Vol 144. P. 307–312. DOI: https://doi.org/10.1007/978-3-030-81907-1_17
87. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional. 2003. 560 p. ISBN 978-0321127426
88. From idea to impact: Real-world success stories of building intelligent apps with Azure. URL: https://azure.microsoft.com/en-us/blog/from-idea-to-impact-real-world-success-stories-of-building-intelligent-apps-with-azure
89. Furkan Ö. How AI Voice Technology is Transforming the Gaming Industry. URL: https://speaktor.com/ai-voice-in-gaming/

90. Game On: How AI is Revolutionizing Game Localization! URL: https://linkedin.com/pulse/game-how-ai-revolutionizing-localization-midlocalize-wq6of
91. Gao Y., Gu X., Zhang H., Lin H., Yang M. Runtime Performance Prediction for Deep Learning Models with Graph Neural Network. *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Melbourne, Australia, May 2023. P. 368-380. DOI: https://dx.doi.org/10.1109/ICSE-SEIP58684.2023.00039
92. Gao, Y. et al. Retrieval-augmented generation for large language models: A survey. *arXiv*. 2023. Vol. 2. DOI: https://doi.org/10.48550/arXiv.2312.10997
93. Gartner Survey Finds 45% of Organizations With High AI Maturity Keep AI Projects Operational for at Least Three Years. URL: https://www.gartner.com/en/newsroom/press-releases/2025-06-30-gartner-survey-finds-forty-five-percent-of-organizations-with-high-artificial-intelligence-maturity-keep-artificial-intelligence-projects-operational-for-at-least-three-years
94. Gaudenz Boesch. Image Recognition: The Basics and Use Cases. URL: https://viso.ai/computer-vision/image-recognition/
95. Generative AI fundamentals: Exploring the 6-Layer architecture. URL: https://www.epicalgroup.com/blogs/generative-ai-fundamentals-exploring-6-layer-architecture
96. GitHub Copilot gets smarter at finding your code: Inside our new embedding model. The GitHub Blog. URL: https://github.blog/news-insights/product-news/copilot-new-embedding-model-vs-code/
97. GKE cluster architecture. *Google Kubernetes Engine (GKE). Google Cloud*. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture
98. Global video game market value from 2022 to 2032 (in billion U.S. dollars). *Statista*. URL: https://www.statista.com/statistics/292056/video-game-market-value-worldwide/
99. Google Kubernetes Engine (GKE). *Google Cloud*. URL: https://cloud.google.com/kubernetes-engine
100. Gosala B, Chowdhuri SR, Singh J, Gupta M, Mishra A. Automatic Classification of UML Class Diagrams Using Deep Learning Technique: Convolutional Neural Network. *Applied Sciences*. 2021. Vol. 11(9). P. 4267. DOI: https://doi.org/10.3390/app11094267
101. GPT-4o Benchmark - Detailed Comparison with Claude & Gemini. *Wielded*. URL: https://www.wielded.com/blog/gpt-4o-benchmark-detailed-comparison-with-claude-and-gemini
102. Guide to Image Segmentation in Computer Vision: Best Practices. URL: https://encord.com/blog/image-segmentation-for-computer-vision-best-practice-guide/
103. Gupta N. Mercedes-Benz: Revolutionizing Automotive with Digital Twins. *LinkedIn Pulse*. URL: https://www.linkedin.com/pulse/mercedes-benz-revolutionizing-automotive-digital-twins-neha-gupta-laoec

104. Hands On Behavioral Code Analysis: CodeScene Use Cases. *CodeScene 3.0.2 Documentation.* URL: https://docs.enterprise.codescene.io/versions/3.0.2/usage/
105. He K., Gkioxari G., Dollar P., Girshick R. Mask R-CNN. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017. P. 2961-2969 URL: https://surli.cc/izfchk
106. Agrawal H., Desai K. Canny edge detection: a comprehensive review. *International Journal of Technical Research & Science.* 2024. Vol. 9. P. 27-35. DOI: https://doi.org/10.30780/specialissue-ISET-2024/023
107. Hough Line Transform. URL: https://docs.opencv.org/4.x/d6/d10/tutorial_py_houghlines.html
108. How FDA Regulates Artificial Intelligence in Medical Products. *The Pew Charitable Trusts*. URL: https://www.pew.org/en/research-and-analysis/issue-briefs/2021/08/how-fda-regulates-artificial-intelligence-in-medical-products
109. How to Measure Technical Debt: Step by Step Guide. URL: https://vfunction.com/blog/how-to-measure-technical-debt/
110. Huang L., Yu W., Ma W., Zhong W., Feng Z., Wang H., Chen Q., Peng W., Feng X., Qin B., Liu T. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Trans*. *Inf. Syst.* 2025. Vol. 43, Issue 2, Article 42. 55 p. DOI: https://doi.org/10.1145/3703155
111. IBM DOORS Next Requirements Intelligence Assistant. URL: https://github.com/IBM/doors-next-ri
112. Image Classification vs. Object Detection: Everything You Need to Know. URL: https://mindy-support.com/news-post/image-classification-vs-object-detection-everything-you-need-to-know/
113. Implementing advanced prompt engineering with Amazon Bedrock. Artificial Intelligence. *Amazon.* URL: https://aws.amazon.com/blogs/machine-learning/implementing-advanced-prompt-engineering-with-amazon-bedrock/
114. INVEST (mnemonic). *Wikipedia*. URL: https://en.wikipedia.org/wiki/INVEST_(mnemonic)
115. Inzoi's 'Smart Zoi' AI system sounds great on paper but seeing it in a live demo didn't exactly wow me. URL: https://www.pcgamer.com/games/life-sim/inzois-smart-zoi-ai-system-sounds-great-on-paper-but-seeing-it-in-a-live-demo-didnt-exactly-wow-me/
116. Islam M., Khan F., Alam S., Hasan M. Artificial Intelligence in Software Testing: A Systematic Review. *IEEE Region 10 Conference (TENCON'2023*). DOI: https://doi.org/10.1109/TENCON58879.2023.10322349
117. Vaibhav J., Simic M. Microservices Circuit Breaker Pattern. *Baeldung*. 2023. URL: https://www.baeldung.com/cs/microservices-circuit-breaker-pattern
118. Jegham N., Koh Ch., Abdelatti M., Hendawi A.YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions. *arXiv*. 2025. Vol. 2411.00201v3. URL: https://arxiv.org/pdf/2411.00201v3

119. Junaid Kh. Create Slide Decks and Product Market Analysis 5X FASTER. 2024. URL: https://newsletter.ertiqah.com/p/create-slide-decks-product-market-analysis-5x-faster
120. Kabir H., Rahman T., Mridul A. Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms. *Smart Wearable Technology*. 2025. P. 1-15. DOI: https://doi.org/10.47852/bonviewSWT52025645
121. Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR*). 2016. DOI: https://doi.org/10.1109/CVPR.2016.90
122. Karhu K., Kasurinen J., Smolander K. Expectations vs Reality – A Secondary Study on AI Adoption in Software Testing. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2504.04921
123. Karpathy A. Software 2.0. *Medium*. URL: https://karpathy.medium.com/software-2-0-a64152b37c35
124. Khanam R., Hussain M. What is YOLOv5: A deep look into the internal features of the popular object detector. *arXiv*. 2024. Vol. 2407.20892. DOI: https://doi.org/10.48550/arXiv.2407.20892
125. Kim S., Sim A., Wu K., Byna S., Son Y., Eom P. Towards HPC I/O Performance Prediction through Large-scale Log Analysis. *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 2020. P. 77-88. DOI: https://doi.org/10.1145/3369583.3392678
126. Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems.* 2012. Vol. 25(2). DOI: https://doi.org/10.1145/3065386
127. Kumar A., Anand A.K., Sahoo B. From Stateless to Stateful: A Comparative Analysis of Stateful Serverless Computing Frameworks. *Proceedings of International Conference on Computational Intelligence. ICCI 2023*. Springer, Singapore. 2024. P. 223-237. DOI: https://doi.org/10.1007/978-981-97-3526-6_19
128. Kuriakose A. Automating Performance Degradation Detection in CI/CD Pipelines. *International Journal of Science and Research*. 2025. Vol. 14(4). P.15-18. DOI: https://doi.org/10.21275/SR25330053612
129. Lakshmanarao K. Different Types of Architectures in Frontend Design and Development. *International Journal for Multidisciplinary Research*. 2024. Vol. 6, Issue 5. DOI: https://doi.org/10.36948/ijfmr.2024.v06i05.28707
130. Layers in software architecture. URL: https://medium.com/@sagar.hudge/layers-in-software-architecture-c8cc16329ff6
131. Lei Z., Jiexin P., Gui Ch., Xiaoli S. An Improved SURF and Modified Zernike Moments Descriptor for Object Recognition. *Electronics.* 2025. Vol. 14(5), No 1025. DOI: https://doi.org/10.3390/electronics14051025

132. Li J., Cao H., Lin L., Hou Y., Zhu R., Ali A. User Experience Design Professionals' Perceptions of Generative Artificial Intelligence. *Arxiv*. 2023. Vol. 1. P. 1-25. DOI: https://doi.org/10.48550/arXiv.2309.15237
133. Li R., Zhang Y., Zhou X., Liang P., et al. MAAD: Automate Software Architecture Design through Knowledge-Driven Multi-Agent Collaboration. 2025. *arXiv.* Vol. 2507.21382. URL: https://arxiv.org/html/2507.21382v1
134. Liu W. et al. Ssd: Single shot multibox detector. *European conference on computer vision*. Cham : Springer International Publishing, 2016. P. 21-37. DOI: https://doi.org/10.48550/arXiv.1512.02325
135. Load Balancing: Handling Heterogeneous Hardware. *Uber Blog.* URL: https://www.uber.com/en-UA/blog/load-balancing-handling-heterogeneous-hardware/
136. Lomio F., Moreschini S., Lenarduzzi V. A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction. *Empir Software Eng.* 2022. Vol. 27. P. 189. DOI: https://doi.org/10.1007/s10664-022-10164-z
137. Lyashenko V. Understanding Few-Shot Learning in Computer Vision: What You Need to Know. URL: https://surl.li/luvrhj
138. Maalej W., Biryuk V., Wei J., Panse F. On the Automated Processing of User Feedback. *arXiv*. 2024. DOI: https://doi.org/10.48550/arXiv.2407.15519
139. Machine Learning Lens. URL: https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/machine-learning-lens.html
140. Mandal K., Nadim M., Chanchal R., Banani R., Schneider K. Evaluating the Performance of a D-Wave Quantum Annealing System for Feature Subset Selection in Software Defect Prediction. *arXiv*. 2024. DOI: https://doi.org/10.48550/arXiv.2410.16469.
141. Mask R-CNN | ML. URL: https://www.geeksforgeeks.org/machine-learning/mask-r-cnn-ml
142. Massri B., Przychodzeń J. GKE at 65,000 nodes: Evaluating performance for simulated mixed AI workloads. 2025. URL: https://cloud.google.com/blog/products/containers-kubernetes/benchmarking-a-65000-node-gke-cluster-with-ai-workloads
143. Mbemba C. Overcoming memory limitations in generative AI: Managing context windows effectively. *CapitalTG Blog.* 2025. URL: https://blog.capitaltg.com/overcoming-memory-limitations-in-generative-ai-managing-context-windows-effectively
144. McKinsey Technology Trends Outlook 2025. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-top-trends-in-tech
145. McKinsey: AI Could Generate Up to $23 Trillion Annually by 2040. URL: https://www.marketingaiinstitute.com/blog/mckinsey-ai-economic-impact
146. Meshy AI Review: How I Generated 3D Models in One Minute. URL: https://www.unite.ai/meshy-ai-review/

147. Meshy AI: Transform Text into 3D Textures. URL: https://cut-out.pro/learn/meshy-ai/
148. Meta AI Infrastructure: Mark Zuckerberg's $100B Vision for Data Centers. *AI CERTs News.* https://www.aicerts.ai/news/meta-ai-infrastructure-zuckerberg-billion-dollar-data-centers/
149. Meta's Bold Investment in AI Infrastructure. URL: https://techtony.co/2025/07/15/meta-ai-data-centres/
150. Microservice Architecture pattern. URL: https://microservices.io/patterns/microservices.html
151. Microservices vs monolithic architecture. *Atlassian*. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith
152. Miño J., Andrade R., Torres J., Chicaiza K. Leveraging Generative Artificial Intelligence for Software Antipattern Detection. *ICIM'24. Communications in Computer and Information Science*. 2024. Vol. 2102. P. 138-149. DOI: https://doi.org/10.1007/978-3-031-64359-0_11
153. Mistry D., Banerjee A. Comparison of Feature Detection and Matching Approaches: SIFT and SURE. *GRD Journals-Global Research and Development Journal for Engineering*. 2017. Vol. 2 No. 3. DOI: https://doi.org/10.70179/3qhg4p03
154. Monolith vs Microservices in 2025. URL: https://foojay.io/today/monolith-vs-microservices-2025/
155. Moreschini S., Pour S., Lanese I., Balouek D., Bogner J., Li X., Pecorelli F., Soldani J., Truyen E., Taibi D. AI Techniques in the Microservices Life-Cycle: a Systematic Mapping Study. *Computing*. 2025. Vol. 107(4). P.50. DOI: http://doi.org/10.1007/S00607-025-01432-Z
156. Natha S. A Systematic Review of Anomaly detection using Machine and Deep Learning Techniques. *Quaid-e-Awam University Research Journal of Engineering, Science & Technology*. 2022. Vol. 20. P. 83-94. DOI: https://doi.org/10.52584/QRJ.2001.11
157. Navneet S.K., Chandra J. Rethinking Autonomy: Preventing Failures in AI-Driven Software Engineering. *arXiv*. 2025. Vol. 2508.11824. DOI: https://doi.org/10.48550/arXiv.2508.11824
158. Neon Giant teams up with Altered for voice in action RPG game The Ascent. URL: https://www.altered.ai/blog/neon-giant-teams-up-with-altered-for-voice-in-action-rpg-game-the-ascent/
159. Netflix-Style Recommendation System Demo. URL: https://github.com/Theglassofdata/Netflix-Redis-Cache-Recommendation-Engine
160. Neural Architecture Search Algorithm. *GeeksforGeeks.* URL: https://www.geeksforgeeks.org/deep-learning/neural-architecture-and-search-methods

161. Nigenda D. et al. Amazon SageMaker Model Monitor: A System for Real-Time Insights into Deployed Machine Learning Models. *arXiv*. 2022. DOI: https://doi.org/10.48550/arXiv.2111.13657
162. Niño-Mora J. Markovian Restless Bandits and Index Policies: A Review. *Mathematics.* 2023. Vol. 11. DOI: https://doi.org/10.3390/math11071639
163. Okeseeyin T. Job Trends 2025: Top 20 Fastest Growing Jobs. *LinkedIn Pulse*, 2025. URL: https://www.linkedin.com/pulse/job-trends-2025-top-20-fastest-growing-jobs-temitope-okeseeyin-2cs5e
164. Open Source Computer Vision. URL: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html
165. OpenAI. GPT-4 Technical Report. *arXiv preprint*. 2024. Vol. 2303.08774. DOI: https://doi.org/10.48550/arXiv.2303.08774
166. OpenCV is the world's biggest computer vision library. URL: https://opencv.org
167. Optimize GKE resource utilization for mixed AI/ML training and inference workloads. *GKE AI/ML. Google Cloud*. URL: https://cloud.google.com/kubernetes-engine/docs/tutorials/mixed-workloads
168. Optimizing Microservices Architecture Using Amazon EKS with Harri. *AWS*. URL: https://aws.amazon.com/solutions/case-studies/harri-eks-case-study/
169. OWASP ZAP. URL: https://www.zaproxy.org/
170. Parekh K. Next-Gen Quality Assurance: Leveraging AI, Automation, and DevOps for Scalable Software Excellence. *International Research Journal on Advanced Engineering and Management (IRJAEM)*. 2025. Vol. 3. P. 2345-2351. DOI: https://doi.org/10.47392/IRJAEM.2025.0370.
171. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*. *arXiv*. 2019. Vol. 1912.01703. DOI: https://doi.org/10.48550/arXiv.1912.01703
172. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2023. Vol. 24. P. 2825-2830. DOI: https://doi.org/10.48550/arXiv.1201.0490
173. Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv*. 2023. Vol. 2302.06590. DOI: https://doi.org/10.48550/arXiv.2302.06590
174. Płoszaj-Mazurek M., Ryńska E. Artificial Intelligence and Digital Tools for Assisting Low-Carbon Architectural Design: Merging the Use of Machine Learning, Large Language Models, and Building Information Modeling for Life Cycle Assessment Tool Development. *Energies*. 2024. Vol. 17(12). DOI: https://doi.org/10.3390/en17122997.
175. Prime Video Sparks Serverless Debate by Switching to Monolith. URL: https://virtualizationreview.com/articles/2023/05/08/serverless-vs-monolith.aspx
176. Python OpenCV filter2D() function – A Complete Guide. URL: https://www.askpython.com/python-modules/opencv-ilter2d
177. Quantum Black AI by McKinsey. The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024

178. Quin F., Weyns D., Galster M., Silva C. A/B testing: A systematic literature review. *Journal of Systems and Software*. 2024. Vol. 211. DOI: https://doi.org/10.1016/j.jss.2024.112011
179. Razer launches new Wyvrn game dev platform with automated AI bug tester. URL: https://www.theverge.com/news/632121/razer-wyvrn-developer-ai-qa-gamer-copilot-sensa-haptics-thx
180. Razer Reveals WYVRN: AI-Powered Gaming Ecosystem Set to Transform Play and Development. URL: https://press.razer.com/company-news/razer-reveals-wyvrn-ai-powered-gaming-ecosystem-for-game-developers/
181. RealTime Data Infrastructure at Uber. URL: https://www.cs.purdue.edu/homes/csjgwang/CS440-F22/slides/Lec31-Uber-lecture.pdf
182. Real-Time Vision: Accelerating Innovations in Computer Processing. URL: https://surl.li/sybliy
183. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. 2016 *EEE Conference on Computer Vision and Pattern Recognition (CVPR*). DOI: 10.1109/CVPR.2016.91
184. Ren S., He K., Girshick R., Sun J. Faster R-CNN: towards real-time object detectionwith region proposal networks, *IEEE Trans. Pattern. Anal. Mach. Intell*. 2016. Vol. 39 (6). P. 1137–1149. DOI: https://doi.org/10.1109/tpami.2016.2577031
185. Replicastudios: AI Voice Actors for Games, Films, and Animation I. URL: https://hunts-ai.onrender.com/ai/replicastudios/
186. Research Areas in Computer Vision: Trends and Challenges. URL: https://surl.lu/dkcbei
187. Ribeiro M. T., Singh S., Guestrin C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of KDD*. 2016. P. 1135-1144. DOI: https://doi.org/10.1145/2939672.2939778
188. RNN vs LSTM vs GRU vs Transformers. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/deep-learning/rnn-vs-lstm-vs-gru-vs-transformers/
189. Rudd I. Microservices Architecture using Netflix tech stack – Conceptual view. *ResearchGate*. 2009. DOI: https://doi.org/10.13140/RG.2.2.32182.78401
190. Russo D. Navigating the Complexity of Generative AI Adoption in Software Engineering – RCR Report. *ACM Transactions on Software Engineering and Methodology*. 2024. URL: https://dl.acm.org/doi/full/10.1145/3680471?af=R
191. Shakked N., Whitney Z. Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. 2023. DOI: http://dx.doi.org/10.2139/ssrn.4375283
192. Sharma M., Agrawal A. Test Case Design and Test Case Prioritization using Machine Learning. *International Journal of Engineering and Advanced Technology*. 2019. Vol. 9, Issue 1. P. 2742-2748. DOI: https://doi.org/10.35940/ijeat.A9762.109119

193. Shcherban S., Liang P., Li Z., Yang C. Multiclass Classification of UML Diagrams from Images Using Deep Learning. *International Journal of Software Engineering and Knowledge Engineering.* 2021. Vol. 31. P. 1683-1698. DOI: https://doi.org/10.1142/S0218194021400179
194. Shopify API, libraries, and tools. URL: https://shopify.dev/docs/api
195. Shrestha J. Evaluating the Application of SOLID Principles in Modern AI Framework Architectures. *arXiv preprint*. Vol. 2503.13786. DOI: https://doi.org/10.48550/arXiv.2503.13786
196. Singh S., Sharma K., Karna B.K., Raj P. Pruning and Quantization for Deeper Artificial Intelligence (AI) Model Optimization. *Intelligent Control, Robotics, and Industrial Automation.* 2023. Vol. 1066. P. 933–945. DOI: https://doi.org/10.1007/978-981-99-4634-1_73
197. SoftServe launches strategic Gen AI program to drive employee productivity. URL: https://www.softserveinc.com/uk-ua/news/softserve-launches-strategic-gen-ai-program
198. Software 3.0 vs AI Agentic Mesh: Why McKinsey Got It Wrong. URL: https://natesnewsletter.substack.com/p/software-30-vs-ai-agentic-mesh-why
199. Software Developers Statistics 2024 - State of Developer Ecosystem Report. *JetBrains: Developer Tools for Professionals and Teams.* URL: https://www.jetbrains.com/lp/devecosystem-2024/
200. Stack Overflow Developer Survey 2025. URL https://survey.stackoverflow.co/2025/
201. Sugali K., Sprunger C., Inukollu V. Software testing: issues and challenges of artificial intelligence & machine learning. *International Journal of Artificial Intelligence and Applications (IJAIA)*. 2021. Vol.12, No.1. P. 101-112. DOI: https://doi.org/10.5121/ijaia.2021.12107
202. Superagency in the workplace: Empowering people to unlock AI's full potential at work. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ais-full-potential-at-work
203. Syed N. et al. Artificial Intelligence as a Service (AIaaS) for Cloud, Fog and the Edge: State-of-the-Art Practices. *ACM Computing Surveys*. 2025. Vol. 57. № 8. P. 1-36. DOI: https://doi.org/10.1145/3712016
204. Szegedy, C., et al. Going Deeper with Convolutions. 2015 *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), Boston, MA, 7-12 June 2015. P. 1-9. DOI: https://doi.org/10.1109/CVPR.2015.7298594
205. Taking Neuromorphic Computing to the Next Level with Loihi 2. *Intel Corporation.* URL: https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf
206. Tchatchoua P., Graton G., Ouladsine M., Christaud J-F. Application of 1D ResNet for Multivariate Fault Detection on Semiconductor Manufacturing Equipment. *Sensors*. 2023. No 23(22):9099. DOI: https://doi.org/10.3390/s23229099
207. Testing the Intelligence of Your AI. EXACTPRO, 2019. URL: https://exactpro.com/ideas/white-papers/testing-ntelligence-your-ai

208. The economic impact of the AI-powered developer lifecycle and lessons from GitHub Copilot. URL: https://github.blog/news-insights/research/the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-github-copilot/
209. The Future of Jobs Report 2025. *World Economic Forum.* URL: https://www.weforum.org/publications/the-future-of-jobs-report-2025/digest
210. The new economics of enterprise technology in an AI world. *McKinsey*. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-new-economics-of-enterprise-technology-in-an-ai-world
211. The prompt: What are long context windows and why do they matter? *Google Cloud.* URL: https://cloud.google.com/transform/the-prompt-what-are-long-context-windows-and-why-do-they-matter
212. The Role of AI in Sustainable Architecture: How Generative Design is Helping to Reduce Carbon Footprints. *Maket*. URL: https://www.maket.ai/post/the-role-of-ai-in-sustainable-architecture-how-generative-design-is-helping-to-reduce-carbon-footprints
213. The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024
214. The State of AI: Global survey. *McKinsey*. URL: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai
215. Thomas A. Review of AI-Driven Approaches for Automated Defect Detection and Classification in Software. Testing. *International Journal of Research and Review*. 2025. Vol. 12; Issue 6. P. 154-160. DOI: https://doi.org/10.52403/ijrr.20250619
216. Tokarev K. DeepMotion: AI Driven Motion for Games. URL: https://80.lv/articles/deepmotion-ai-driven-motion-for-games
217. Transforming Kubernetes and GKE into the leading platform for AI/ML. *Google Open Source Blog*. URL: https://opensource.googleblog.com/2025/05/transforming-kubernetes-and-gke-into-the-leading-platform-for-aiml.html
218. Trofymenko O., Dyka A., Loboda Y., Tolocknov A., Bondarenko M. Artificial Intelligence in the GameDev. *Electronic Professional Scientific Journal «Cybersecurity: Education, Science, Technique»*. 2025. Vol. 27. P. 109-119. DOI: https://doi.org/10.28925/2663-4023.2025.27.701
219. Try it yourself: Requirements AI assistant for DOORS Next. URL: https://community.ibm.com/community/user/blogs/daniel-moul/2025/02/13/requirements-ai-assistant-for-doors-next
220. Understanding Message Queues: A Guide for Developers day 40 of system design basics. URL: https://dev.to/vincenttommi/understanding-message-queues-a-guide-for-developers-day-40-of-system-design-basics-3b09
221. Understanding Single Shot Detector (SSD). URL: https://surl.lu/cvtagv
222. Unwinding Technical Debt. URL: https://learn.castsoftware.com/unwinding-technical-debt
223. van Remmen J.S., Horber D., Lungu A., et al. Natural language processing in

requirements engineering and its challenges for requirements modelling in the engineering design domain. *Proceedings of the Design Society*. 2023. Vol. 3. P. 2765-2774. DOI: https://doi.org/10.1017/pds.2023.277
224. Vendor Lock-in in Cloud Computing. URL: https://www.geeksforgeeks.org/mobile-computing/vendor-lock-in-in-cloud-computing/
225. Wang J., Kim D. Automatic Classification of UML Class Diagrams Using Deep Learning Technique. *Applied Sciences*. 2021. Vol. 11(9). P. 4267. DOI: https://doi.org/10.3390/app11094267
226. Wang J., Suleiman B., Alibasa M. Automated Unit Test Case Generation: A Systematic Literature Review. *arXiv*. Vol. 2504.20357. DOI: https://doi.org/10.48550/arXiv.2504.20357
227. Wang L., Zhang X., Su H., Zhu J. A comprehensive survey of continual learning: Theory, method and application. *EEE transactions on pattern analysis and machine intelligence*. 2024. Vol. 46(8). P. 5362-5383. DOI: https://doi.org/10.1109/TPAMI.2024.3367329
228. Weiche H. et al. A Comprehensive Guide to Explainable AI: From Classical Models to LLMs. *arXiv.* 2024. DOI: https://doi.org/10.48550/arXiv.2412.00800
229. What are convolutional neural networks? URL: https://www.ibm.com/think/topics/convolutional-neural-networks
230. What is AWS Well-Architected Tool? *AWS Well-Architected Tool.* 2025. URL: https://docs.aws.amazon.com/wellarchitected/latest/userguide/intro.html
231. What is descriptor in computer vision? URL: https://surl.li/sokqhh
232. What is GitHub Copilot? URL: https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot
233. What is Human-in-the-Loop (HITL) in AI & ML? URL: https://cloud.google.com/discover/human-in-the-loop
234. What is Kubeflow and how can it be used? *Google Cloud*. URL: https://cloud.google.com/discover/what-is-kubeflow
235. What Is Named Entity Recognition? *IBM*. URL: https://www.ibm.com/think/topics/named-entity-recognition
236. What is Retrieval-Augmented Generation (RAG)? URL: https://cloud.google.com/use-cases/retrieval-augmented-generation
237. What's the scope of computer vision in AI? URL: https://surl.lu/cqwyou
238. World Intellectual Property Organization. Artificial Intelligence and Intellectual Property. URL: https://www.wipo.int/en/web/frontier-technologies/artificial-intelligence/index
239. Wotawa F., Klampfl L., Jahaj L. A framework for the automation of testing computer vision systems. *arXiv*. 2021. DOI: https://doi.org/10.48550/arXiv.2105.04383
240. Yang D., Kleinman E., Harteveld C. GPT for Games: An Updated Scoping Review (2020-2024). *arXiv*. 2024. Vol. 2411.00308. https://doi.org/10.48550/arXiv.2411.00308
241. Yokelson D., Charest M., Wai Li Y. HPC Application Performance Prediction with Machine Learning on New Architectures. *Proceedings of the 2023 on*

Performance EngineeRing, Modelling, Analysis, and VisualizatiOn Strategy (PERMAVOST '23). Association for Computing Machinery, New York, NY, USA, 2023. P. 1-8. DOI: https://doi.org/10.1145/3588993.3597262
242. Yokelson D., Robert M., Charest J., Wai Y. HPC Application Performance Prediction with Machine Learning on New Architectures. *Proceedings of PERMAVOST'23*. 2023. P. 1-8. DOI: https://doi.org/10.1145/3588993.3597262
243. YOLO: Algorithm for Object Detection Explained. URL: https://www.v7labs.com/blog/yolo-object-detection
244. Your Cloud Strategy Is Now Your AI Strategy, Too. URL: https://www.forrester.com/blogs/your-cloud-strategy-is-now-your-ai-strategy-too/
245. Zhang M., Li. Y., Yue T., Cai K-Y. Quantum Optimization for Software Engineering: A Survey. *arXiv*. 2025. Vol. 2506.16878. DOI: https://doi.org/10.48550/arXiv.2506.16878
246. Zhang Yu., Chen J., Liu H., Chen Y., Xiao B., Li H. Recent advancements of human-centered design in building engineering: A comprehensive review. *Journal of Building Engineering*. 2024. Vol. 84. DOI: https://doi.org/10.1016/j.jobe.2024.108529
247. Zhao L., Alhoshan W., Ferrari A., Letsholo K.J. Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study. *arXiv*. 2020. Vol. 2004.01099. DOI: https://doi.org/10.48550/arXiv.2004.01099
248. Zhipeng D., Hao S., Shilin Z., Juanping Z., Lin L., Huanxin Z. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2018. Vol. 145. Part A. P. 3-22. DOI: https://doi.org/10.1016/j.isprsjprs.2018.04.003.
249. Zhuy H., Liu D., Bayley I., Harrison R., Cuzzolin F. Datamorphic Testing: A Methodology for Testing AI Applications. *Technical Report OBU-ECM-AFM-2018-02.* URL: https://arxiv.org/ftp/arxiv/papers/1912/1912.04900.pdf
250. Волокита А., Зоткін К. Проблематика проєктування програмних систем на основі подійно-орієнтованих архітектур (EDA). *Measuring and computing devices in technological processes*. 2024. Vol. 4. P. 130–136. DOI: https://doi.org/10.31891/2219-9365-2024-80-16.
251. Гороховатський В.О., Пупченко Д.В., Солодченко К.Г. Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення. *Системи управління навігації та зв'язку*. 2018. № 1(47). С. 93-98. DOI: https://doi.org/10.26906/SUNZ.2018.1.093
252. Дика А. І. Тестування безпеки вебзастосунків за допомогою ChatGPT. *Актуальні тенденції розвитку освіти, науки та технологій*: матер. VII міжнар. наук.-практ. конф. Харків-Бахмут, 15 травня 2024 р. Т. 2. С. 28-30. URL: https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi
253. Дика А. І., Трофименко О. Г. Аналіз потенційних загроз засобами ChatGPT для тестування безпеки вебзастосунків. *Європейські орієнтири розвитку України: науково-практичний вимір в умовах воєнних викликів*: матер.

міжнар. наук.-практ. конф. Одеса, 26 квітня 2024 р. Одеса: НУ «ОЮА», 2024. С.876-878. URL: https://hdl.handle.net/11300/28085
254. Дика А.І. Роль штучного інтелекту у тестуванні безпеки вебзастосунків. *Інформаційне суспільство: проблеми та перспективи* : матер. IX всеукр. наук.-практ. конф. (24 травня 2024). Одеса, 2024. С. 153-156. DOI: https://doi.org/10.32837/11300.27842
255. Дика А.І. Тестування штучного інтелекту: ключові виклики, стратегії вдосконалення. *Електронні інформаційні ресурси: створення, використання, доступ:* матер. міжнар. наук.-практ. інтернет-конф. Вінниця, 20-21 листопада 2023, КЗВО "Вінницька академія безперервної освіти". С.93-95
256. Котенко М.М., Вакалюк Т.А. Впровадження мікросервісної архітектури: технічні виклики та рішення. *Вісник Херсонського національного технічного університету*. 2024. №. 4 (91). С. 299-305. DOI: https://doi.org/10.35546/kntu2078-4481.2024.4.39
257. Кудрик О.В., Бісікало О.В., Здітовецький Ю.С. Методи тонкого налаштування штучного інтелекту. *Вісник Вінницького політехнічного інституту*. 2024. № 4. С. 139–146. DOI: https://doi.org/10.31649/1997-9266-2024-175-4-139-146
258. Лобода Ю.Г., Трофименко О.Г., Манаков С.Ю., Гура В.І. Штучний інтелект в сучасній веброзробці та вебдизайні: багаторівнева класифікація та систематизація. *Informatics and Mathematical Methods in Simulation*. 2025. № 15(1). С. 126-136. DOI: https://doi.org/10.15276/imms.v15.no1.126
259. Логінова Н.І. Використання мови програмування Python для вирішення завдань розпізнавання образів. *Інформаційне суспільство: проблеми та перспективи:* матер. VIII Всеукр. наук.-практ. конф. (м. Одеса, 2 червня 2023 р.). С. 56-60. https://doi.org/10.32837/ 11300.25263
260. Логінова Н.І., Толокнов А.А. Візуалізація даних у Python. *Європейські орієнтири розвитку України: науково-практичний вимір в умовах воєнних викликів* : матер. міжнар.наук.-практ. конф. (Одеса, 26 квітня 2024 р.) Одеса: Фенікс, 2024. С. 854-856.
261. Розпізнавання образів та обробка зображень: метод. вказівки до виконання лабораторних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення». Чернігів: НУ «Чернігівська політехніка», 2021. 146 с. URL: https://surl.li/qfczfa
262. Стоян А.О. Дослідження методів та засобів виявлення об`єктів на відеозображеннях. URL: https://eom.lpnu.ua/sntk/doc/spr2020/stoyan.pdf
263. Тестування та забезпечення якості програмних систем : навч. посібник [Електронне видання] / О.Г. Трофименко, А.І. Дика; Нац. ун-т «Одес. юрид. академія». Одеса: Фенікс, 2024. 195 с. URL: https://hdl.handle.net/11300/27717 . ISBN 978-617-8395-88-9
264. Тестування та забезпечення якості програмних систем : навч.-метод. посіб. [Електронне видання] / О.Г. Трофименко, А.І. Дика; Нац. ун-т «Одес. юрид. академія». Одеса: Фенікс, 2024. 140 с. URL: https://hdl.handle.net/11300/27246.

265. Трофименко О. Г. Інструменти статичного тестування безпеки. *Інформаційні технології: моделі, алгоритми, системи (ITMAS-2024):* матер. V Всеукр. наук.-практ. інтернет-конф. (30-31 жовтня 2024 р.) Миколаїв. С. 188-190. URL: https://itconf.nuos.edu.ua/2024/publications/static-security-testing-tools/
266. Трофименко О. Г. Складнощі та недоліки використання генеративного штучного інтелекту в розробленні та тестуванні ігор. *Інформаційне суспільство: проблеми та перспективи* : матер. X всеукр. наук.-практ. конф. (23 травня 2025). Одеса, 2025. С. 28-31. DOI: https://doi.org/10.32837/11300.29842
267. Трофименко О. Г., Задерейко О. В., Логінова Н. І., Шевченко О. В. Інструменти штучного інтелекту в розробці графічних елементів, 3D-моделей та інших компонентів відеоігор. *Інформатика та математичні методи в моделюванні*. 2025. Т.15, № 2. С. 276-287. DOI: https://doi.org/10.15276/imms.v15.no2.276
268. Трофименко О. Г., Соколов А. В., Чикунов П. О., Ахмаметьєва Г. В., Атанасевич А. О. Аналіз ролі фахівців із тестування та забезпечення якості. *Збірник наукових праць Національного університету кораблебудування мені адмірала Макарова*. 2024. № 3 (496). С. 106-113. DOI: https://doi.org/10.15589/znp2024.3(496).16.
269. Трофименко О.Г., Дика А.І., Задерейко О.В., Шевченко О.В. Сфери застосування штучного інтелекту в розробці та тестуванні ігор. *Кібербезпека: освіта, наука, техніка*. 2025. № 1(29). С. 41-59. DOI: https://doi.org/10.28925/2663-4023.2025.29.832
270. Трофименко О.Г., Дика А.І., Лобода Ю.Г. Аналіз інструментів тестування вебзастосунків. *Кібербезпека: освіта, наука, техніка.* 2023. № 4(20). С. 62-71. DOI: https://doi.org/10.28925/2663-4023.2023.20.6271
271. Трофименко О.Г., Пастернак Ю.Ю., Манаков С.Ю., Лобода Ю.Г. Автоматизація тестування вебсайтів електронної комерції. *Сучасна спеціальна техніка*. 2021. № 2(65). С. 46-59. DOI: https://doi.org/10.36486/mst2411–3816.2021.2(65).5
272. Фургала Ю., Вельгош А., Вельгош С., Русин Б. Використання гістограм кольору для ідентифікації об'єктів при масштабуванні та обертанні зображень *Електроніка та інформаційні технології*. 2020. Вип. 13. С. 28–37. DOI: https://doi.org/10.30970/eli.13.3
273. Час розробки софту скоротився на 20%. SoftServe розпочинає масштабну інтеграцію генеративного ШІ. URL: https://forbes.ua/news/chas-rozrobki-softu-skorotivsya-na-20-softserve-rozpochinae-masshtabnu-integratsiyu-generativnogo-shi-18042024-20630
274. Як використовувати ChatGPT для ефективного тестування безпеки: вичерпний посібник. URL: https://brainberry.ua/uk/newsroom/blog/how-to-use-chatgpt-for-effective-security-testing-a-comprehensive-guide

Наукове видання

Олена Григорівна Трофименко,
Наталія Іванівна Логінова,
Сергій Юрійович Манаков,
Юлія Геннадіївна Лобода,
Анастасія Іванівна Дика

Штучний інтелект у розробленні та тестуванні програмного забезпечення

Монографія

Електронне видання

В авторській редакції

Ум-друк. арк. 12,9.
Заказ № 2606-03.

Видано в ПП «Фенікс»
(Свідоцтво суб'єкта видавничої справи ДК № 1044 від 17.09.02).
Україна, м. Одеса, 65009, вул. Зоопаркова, 25.
e-mail: fenix-izd@ukr.net