

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА КІБЕРНЕТИКИ
КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА

**МАТЕМАТИЧНА ЛОГІКА
ТА ПРОГРАМУВАННЯ.
ДОСВІД ВИКЛАДАННЯ.
2025-2026 навчальний рік**

ЗБІРНИК ПРАЦЬ



Видавничий дім
«Гельветика»
2026

УДК 378.1:004.4:510.6

М34

*Рекомендовано до друку вченою радою
факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
(протокол № __ від ____ 2026 року)*

*Ухвалено науково-методичною комісією
факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
(протокол № 3 від 2 жовтня 2026 року)*

Рецензенти:

Дорошенко Анатолій Юхимович	доктор фізико-математичних наук, професор Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського"
Роскладка Андрій Анатолійович	доктор економічних наук, кандидат фізико- математичних наук, професор Державного торговельно-економічного університету

М34 Математична логіка та програмування. Досвід викладання. 2025-2026 навчальний рік: Збірник праць / За заг. редакцією Г. В. Лівінської, Н. Г. Русіної. – Одеса : Видавничий дім «Гельветика», 2026. – 178 с.
ISBN

У збірнику наукових праць представлено результати теоретичних і прикладних досліджень, присвячених актуальним проблемам розвитку інформаційних технологій, математичної логіки, програмування, комп'ютерних наук, системного аналізу та науки про дані, а також суміжних міждисциплінарних напрямів.

Матеріали збірника висвітлюють сучасні наукові підходи, методи, моделі та технології, результати їх дослідження, апробації й практичного застосування. Окрему увагу приділено проблемам формалізації знань і вимог, розроблення та аналізу програмних систем, застосування математичних методів, технологій штучного інтелекту, системного аналізу та методів опрацювання й аналізу даних.

Представлені у збірнику наукові результати можуть бути використані у подальших дослідженнях за спеціальностями «Комп'ютерні науки» та «Системний аналіз та наука про дані» галузі знань F «Інформаційні технології», а також враховані під час розвитку змісту освітніх та освітньо-наукових програм відповідного спрямування, зокрема ОПП «Інформатика» першого (бакалаврського) рівня вищої освіти, ОНП «Інформатика» другого (магістерського) рівня вищої освіти та ОНП «Аналіз соціальних даних» другого (магістерського) рівня вищої освіти.

ISBN

ЗМІСТ

ВСТУП	5
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ВИКЛАДАННЯ МАТЕМАТИЧНОЇ ЛОГІКИ	
<i>Нікітченко М.С.</i>	8
Про методологічні засади курсу «Математична логіка»	
<i>Шкільняк С.С.</i>	30
Нетрадиційні та програмно-орієнтовані логіки в курсі «Математична логіка»	
РОЗДІЛ 2. ПРАКТИЧНІ АСПЕКТИ ВИКЛАДАННЯ ТА ІНТЕГРАЦІЇ КУРСІВ ПРОГРАМУВАННЯ	
<i>Криволап А.В.</i>	37
Особливості використання мови програмування Python у курсі «Об'єктно-орієнтоване програмування»	
<i>Панченко Т.В.</i>	43
Побудова курсів системного програмування та операційних систем у підготовці ІТ-фахівців: від низькорівневих механізмів до дослідницьких практик	
<i>Русіна Н.Г.</i>	56
Практико-орієнтоване навчання основ програмування здобувачів вищої освіти	
<i>Свистунов А.О., Омельчук Л.Л.</i>	67
Імплементація положення про використання штучного інтелекту при реалізації освітніх програм у змісті навчальної дисципліни «Web-технології»	
<i>Ткаченко О.М.</i>	77
Досвід використання ШІ для формальної специфікації в курсі "Формальні методи розробки програмних систем"	
<i>Шишацька О.В.</i>	83
Інтеграція теоретичного та парадигмального підходів у підготовці бакалаврів інформатики	
<i>Галавай О.М.</i>	98
Інтеграція індустріальних практик у викладання Data Mining: навчальний хакатон як міст до продуктового ІТ	

<i>Шишацький А.В.</i>	105
Інтегрування штучних агентів в розробку в межах курсу «Інструментальні середовища та технології програмування»	
РОЗДІЛ 3 ПРАКТИКО-ОРІЄНТОВАНІ ТЕХНОЛОГІЇ ТА ПРИКЛАДНІ АСПЕКТИ ПІДГОТОВКИ ФАХІВЦІВ ІЗ СИСТЕМНОГО АНАЛІЗУ ТА НАУКИ ПРО ДАНІ	
<i>Лівінська Г.В., Розора І. В., Пономарьов В. Д., Моренець В. І., Сидоров М. В.</i>	110
Міждисциплінарна інтеграція методів візуалізації даних та алгоритмів машинного навчання для верифікації задач класифікації	
<i>Лівінська Г.В., Розора І. В., Шарапов М. М., Македон Т. В.</i>	143
Системна методологія статистичного аналізу складних стохастичних систем та верифікації алгоритмів машинного навчання на основі імітаційного моделювання та методів ресемплінгу	
<i>Шарапов М. М., Розора І. В., Лівінська Г. В., Пономарьов В. Д., Моренець В. І., Макушенко І. А.</i>	158
Формування імовірісно-статистичного мислення як базису підготовки фахівців з ІТ в межах курсу «Теорія ймовірностей та математична статистика»	
<i>Моренець В. І., Пономарьов В. Д., Шарапов М. М.</i>	165
Методи інтелектуального аналізу даних при автоматизованому зборі інформації з відкритих джерел	

ВСТУП

Сучасний розвиток інформаційних технологій, штучного інтелекту, інтелектуальних програмних систем та засобів автоматизації розробки суттєво змінює як професійну діяльність ІТ-фахівців, так і вимоги до їх підготовки у закладах вищої освіти. З одного боку, стрімке оновлення технологій потребує постійної актуалізації змісту навчальних дисциплін, упровадження сучасних інструментів і практик професійної діяльності. З іншого – дедалі більшого значення набуває фундаментальна підготовка, що забезпечує здатність майбутнього фахівця усвідомлювати принципи побудови складних інформаційних та програмних систем, формалізувати задачі, критично оцінювати отримані результати та адаптуватися до технологічних змін.

Особливої актуальності набуває пошук оптимального співвідношення між фундаментальною та практичною складовими ІТ-освіти. Підготовка сучасного фахівця вже не може обмежуватися опануванням окремих мов програмування, технологій або програмних засобів. Вона передбачає формування системного й алгоритмічного мислення, розуміння теоретичних засад програмування та математичних методів, уміння працювати з даними, здійснювати верифікацію отриманих результатів, використовувати сучасні професійні інструменти та розв'язувати комплексні практичні задачі.

Окремим чинником трансформації освітнього процесу стало широке поширення генеративного штучного інтелекту та ШІ-агентів. Їх використання поступово виходить за межі допоміжного інструментарію і впливає на сам характер розробки програмного забезпечення, аналізу даних, формальної специфікації та інших видів професійної діяльності. Відповідно змінюються й освітні завдання: поряд із умінням застосовувати такі засоби необхідно формувати здатність коректно формулювати вимоги, перевіряти й інтерпретувати результати роботи інтелектуальних систем, усвідомлювати їх обмеження та відповідально використовувати їх із дотриманням принципів академічної доброчесності.

Збірник праць «Математична логіка та програмування. Досвід викладання. 2025–2026 навчальний рік» об'єднує результати методичної, науково-методичної та практичної роботи викладачів, пов'язаної з удосконаленням підготовки фахівців у галузі інформаційних технологій. Представлені матеріали охоплюють питання викладання математичної логіки, теорії та технологій програмування, системного програмування, формальних методів розробки програмних систем, об'єктно-орієнтованого програмування, інтелектуального аналізу даних, теорії ймовірностей і

математичної статистики, машинного навчання та інших взаємопов'язаних складових сучасної ІТ-освіти.

Перший розділ збірника присвячено теоретичним аспектам викладання математичної логіки. У представлених роботах аналізуються методологічні засади побудови відповідного курсу, можливості переходу від традиційних підходів до діяльнісно-базованого розуміння логіки, а також місце нетрадиційних і програмно-орієнтованих логік у підготовці фахівців з комп'ютерних наук. Такий підхід підкреслює значення математичної логіки не лише як фундаментальної дисципліни, а й як теоретичної основи для подальшого опанування формальних методів, програмування та сучасних інтелектуальних технологій.

Матеріали другого розділу відображають різні аспекти модернізації викладання програмування та споріднених дисциплін. Розглядаються питання побудови взаємопов'язаного комплексу курсів із системного програмування й операційних систем, інтеграції теоретичної та парадигмальної підготовки, використання Python при вивченні об'єктно-орієнтованого програмування, а також практико-орієнтованого навчання програмуванню здобувачів інших предметних галузей.

Важливе місце в цьому розділі посідають матеріали, присвячені трансформації освітнього процесу під впливом штучного інтелекту. Представлено досвід використання ШІ для формальної специфікації програмних систем, інтеграції генеративного штучного інтелекту та ШІ-агентів у процес розробки програмного забезпечення, а також практичної реалізації вимог щодо відповідального й прозорого використання таких технологій у навчальній дисципліні. Показано, що їх упровадження доцільно спрямовувати не лише на автоматизацію окремих операцій, а насамперед на розвиток умінь формувати вимоги, аналізувати й верифікувати запропоновані рішення та приймати обґрунтовані інженерні рішення.

Ще одним важливим напрямом є наближення навчання до реальних умов професійної діяльності. У збірнику представлено досвід інтеграції індустріальних практик у викладання Data Mining, використання реалістичних продуктових кейсів, командної роботи та навчального хакатону. Такі форми роботи створюють можливості для поєднання теоретичних знань із практичним розв'язанням комплексних задач та сприяють формуванню професійних і комунікативних компетентностей здобувачів освіти.

Третій розділ присвячено практико-орієнтованим технологіям та прикладним аспектам підготовки фахівців із системного аналізу та науки про дані. У представлених матеріалах поєднуються фундаментальна математична

підготовка, статистичне моделювання, машинне навчання, візуалізація та робота з реальними даними. Особлива увага приділяється питанням верифікації моделей і результатів аналізу, формуванню імовірнісно-статистичного мислення, використанню імітаційного моделювання та методів ресемплінгу, а також організації повного циклу роботи з відкритими даними – від їх автоматизованого збору й попередньої обробки до аналізу та візуалізації.

Об'єднувальною ідеєю представлених у збірнику робіт є розуміння сучасної ІТ-освіти як цілісної системи, у якій фундаментальна теоретична підготовка, програмування, математичні й статистичні методи, робота з реальними даними, сучасний професійний інструментарій та дослідницька діяльність не протиставляються, а взаємно доповнюють одне одного. Водночас технологічні зміни, зокрема розвиток штучного інтелекту, потребують не відмови від фундаментальних знань, а переосмислення способів їх застосування та посилення уваги до критичного мислення, верифікації, відповідальності й здатності самостійно приймати рішення.

Представлений у збірнику досвід може бути використаний при оновленні змісту освітніх програм і навчальних дисциплін, удосконаленні методик викладання та оцінювання, розробленні нових практико-орієнтованих і дослідницьких завдань, а також при визначенні підходів до інтеграції штучного інтелекту в освітній процес.

Колектив авторів сподівається, що матеріали збірника будуть корисними для науковців, науково-педагогічних працівників, аспірантів і студентів закладів вищої освіти та сприятимуть подальшому обговоренню шляхів розвитку сучасної освіти в галузі інформаційних технологій.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ВИКЛАДАННЯ МАТЕМАТИЧНОЇ ЛОГІКИ

Нікітченко Микола Степанович, д.ф.-м.н, професор

ПРО МЕТОДОЛОГІЧНІ ЗАСАДИ КУРСУ «МАТЕМАТИЧНА ЛОГІКА»

У статті обговорюються методологічні засади курсу «Математична логіка». Традиційні засади полягають у наступному: математична логіка визначається як наука про правильні міркування, основним поняттям є поняття висловлювання, при дослідженні висловлювань наголос робиться на їх формі в абстракції від їх змісту. Ці поняття фактично ведуть до лінгвістично-базованого підходу, в той час як логіка мала б розвиватись на власних засадах. В статті пропонується викладати курс «Математична логіка» на підставі діяльнісно-базованого підходу.

Ключові слова: математична логіка, засади логіки, діяльнісно-базований підхід, викладання логіки

The article discusses the methodological principles of the course "Mathematical Logic". The traditional principles are as follows: mathematical logic is defined as the science of correct reasoning, the basic concept (notion) is the concept of proposition; when studying propositions, the emphasis is on their form in abstraction from their content. These principles actually lead to a linguistic-based approach, while logic should be developed on its own principles. The article proposes teaching the course "Mathematical Logic" according to an activity-based approach.

Keywords: mathematical logic, basics of logic, activity-based approach, teaching logic

Вступ

На наших очах відбувається нова науково-технічна революція, пов'язана, в першу чергу, з бурхливим розвитком штучного інтелекту. Це актуалізує питання методологічних та теоретичних засад штучного інтелекту, до яких, насамперед, належить логіка, формалізована частина якої вивчається в курсі «Математична логіка».

Але чи дійсно стан викладання цього курсу закладає такі засади математичної логіки, які відповідають сучасним проблемам? Щоб дати відповідь на це запитання, слід проаналізувати методологічні засади як математичної логіки, так і логіки в цілому.

1. Традиційні методологічні засади математичної логіки

Спочатку розглянемо основні визначення математичної логіки. Розпочнемо з найпростішого – з тлумачення словосполучення ‘Математична логіка’.

1.1 Основні тлумачення словосполучення ‘Математична логіка’

Словосполучення ‘Математична логіка’ виокремлює слово (термін) ‘логіка’ як головний, а термін ‘математичний’ – як залежний. Таке тлумачення дозволяє визначити математичну логіку як спеціальну логіку, яка характеризується застосуванням математичних методів. Такої думки дотримується більшість спеціалістів з математичної логіки. Разом з тим, є і інші тлумачення, за якими математична логіка трактується як логіка математики, тобто термін ‘математика’ виступає як головний, а ‘логіка’ – як залежний. Такий підхід притаманний “чистим” математикам, які головну увагу приділяють методам доведення, а не засадам логіки чи математики. Є і змішані тлумачення, які проголошують подвійну роль логіки: як спеціальної логіки, поданої математично, так і як логіку математики.

Таким чином, основними є такі тлумачення:

- 1) математична логіка як спеціальна логіка, подана математично;
- 2) математична логіка як логіка математики;
- 3) математична логіка як спеціальна логіка та як логіка математики.

Розглянемо вказані тлумачення у традиційних визначеннях логіки та математичної логіки.

1.2 Традиційні визначення логіки та математичної логіки

Почнемо з визначень математичної логіки, поданих у підручниках чи популярних статтях. В них часто розглядають математичну логіку як розділ

формальної логіки (чи символічної логіки, чи просто логіки). Наведемо низку таких визначень:

- «Зазвичай, (формальна) логіка займається аналізом речень або висловлювань і доведень; причому основна увага припадає на *форму* в абстракції від змісту» [1].
- «Математична логіка (також звана символічною логікою) — це логіка, що подається математичними методами. Але наша назва має подвійне значення, оскільки ми вивчатимемо логіку, яка використовується в математиці. Логіка має важливу функцію — говорити, що з чого випливає» [2].
- «Одне з популярних визначень логіки полягає в тому, що це аналіз методів міркування. Вивчаючи ці методи, логіку цікавить форма, а не зміст аргументу» [3].
- «Логіка вивчає міркування, а математична логіка вивчає ті типи міркувань, якими користуються математики. Тому, щоб знайти правильний підхід до математичної логіки, ми повинні досліджувати методи, якими користуються у математиці» [4].
- «Логіка – це вивчення методів і принципів, що використовуються для розрізнення правильних і неправильних міркувань» [5].
- «Математична логіка досліджує те, що можна довести, визначити та обчислити в різних математичних умовах. Вона робить це за допомогою строгих математичних методів (метаматематики)» [6].
- «Логіка, кажучи загальними словами, — це наука про те, як виражати поняття і ситуації та правильно міркувати про них. Математична логіка головним чином займається виразами у формальних мовах, тим, як приписувати значення формальним виразам, та тим, як міркувати з формальними виразами, використовуючи правила висновку» [7].
- «Логіка — це аналіз та оцінка аргументів. Тут ми розглянемо міркування щодо філософських питань (таких як Бог, свобода волі та мораль) та інших питань» [8].
- «Логіку зазвичай визначають як науку про обґрунтоване мислення. Але оскільки ‘мислення’ може означати як акт мислення, так і об’єкт

мислення, ми маємо два визначення логіки: логіка як наука (1) про акт обґрунтованого мислення або (2) про об'єкти обґрунтованого мислення» [9].

Отже, наведені визначення дійсно вказують на три тлумачення словосполучення 'Математична логіка'. Зауважимо також, що деякі автори взагалі не надають визначення математичної логіки, а зразу переходить до висловлювань і далі до пропозиційної логіки та логіки предикатів, як це зроблено, наприклад, у [10].

Подібні визначення наведені у словниках та енциклопедіях.

1.3 Аналіз традиційних визначень понять математичної логіки

Почнемо з простих результатів аналізу наведених понять логіки, сформулювавши їх наступним чином:

- В першу чергу відзначимо *різноманітність визначень* математичної логіки, на що, зокрема, вказує Елліотт Мендельсон [3].
- Різноманітність визначень говорить про наявність *різних аспектів* (математичної) логіки та її *зв'язків* з математикою, філософією, лінгвістикою, психологією тощо. В рамках цієї статті детальний аналіз цих зв'язків проводити не будемо.
- Тим не менше, переважна більшість авторів розглядає математичну логіку як *науку про методи (правильних) міркувань*, а логіку – як *науку про мислення, його закони та методи*.
- Базовою складовою міркувань є *висловлювання*, яке визначається як *речення, якому можна приписати значення істинності чи хибності*. Зауважимо, що зазвичай виокремлюють два логічних значення (скажімо, T та F), тобто будують *бівалентну (двозначну) логіку*.
- Висловлювання мають *форму* та *зміст*, математична логіка досліджує *форми висловлювань в абстракції від змісту*.
- Форма висловлювання задає його *синтаксичний аспект*, а зміст – *семантичний аспект*. Висловлювання з синтаксичної точки зору розглядається як *формула*, а з семантичної точки зору – як *предикат*, тобто як певне відображення (функція) в множину логічних значень.

- Міркування тлумачаться як *процеси виведення нових висловлювань* з наявних висловлювань; зазвичай, говорячи про правильні міркування, мають на меті виведення *істинних* висловлювань.
 - Як тоді тлумачиться виведення? Є дві можливості: синтаксична та семантична. Оскільки математична логіка абстрагується від змісту (від семантики), то спочатку зосереджуються на синтаксичній вивідності (тобто на виведенні нових формул). Що стосується семантичної вивідності, то вона пов'язана з істиннісними значеннями (зі змістом), тому в цьому випадку краще говорити не про вивідність, а про слідування (про наслідок). Семантичний аспект міркувань був початково формалізований в роботах А. Тарського.
 - Закони логіки (тотожності, несуперечливості, виключеного третього, достатньої підстави) гарантують правильність міркувань.
- Розглянемо основні результати проведеного аналізу більш детально.

1.4 Висловлювання та їх структура

Висловлювання визначається як *речення, якому приписується значення істинності чи хибності*. Таким чином, загальним поняттям для висловлювання є поняття речення. Це означає, що при аналізі висловлювань будемо змушені застосовувати *лінгвістичну термінологію*, зокрема, виокремлювати в реченнях їх *граматичний (синтаксичний)* та *змістовний (семантичний)* аспекти.

Проблема такого визначення висловлювання полягає в тому, що структура речень природної мови може бути надзвичайно складною. Проілюструємо це на прикладі поняття *складного речення*.

Отже, в реченні (з точки зору його синтаксичної структури), виділяють *граматичну основу (предикативний центр)* та *другорядні члени* [11]. Граматична основа виражає *головну думку (смысл, семантику)* речення. Другорядні члени доповнюють, уточнюють або пояснюють головні чи інші члени речення.

Серед граматичних основ виокремлюють *двоскладні* (з підметом та присудком як головними членами), *односкладні* (з підметом або присудком

як головними членами) та *складені* (підмет чи присудок складається з кількох слів).

За кількістю граматичних основ речення поділяються на *прості* та *складні*. Останні бувають *сполучниковими* (поєднуються сполучниками) та *безсполучниковими*.

Для складних речень їх зміст (семантика) будується на підставі змісту (семантики) основ, які входять у речення, з урахуванням структури речень та контексту їх вживання.

Виявляється, у природній мові є багато різних механізмів (правил), які дозволяють зрозуміти зміст складних речень. Один з таких механізмів пов'язаний із класифікацією сполучників, зокрема, з їх морфологічною побудовою (прості, складні, складені); серед складених ще виокремлюють повторювані та парні.

Приклади сполучників за їх побудовою (за їх синтаксисом):

- прості: і, та, а, але, як;
- складні: щоб, якщо, якби, проте;
- складені: так що, тому що, для того щоб, через те що;
- повторювані: і ..., і; ні ..., ні; то ..., то; чи то ..., чи то;
- парні: якщо ..., то; не тільки ..., а.

Інша (більш цікава для логіки) класифікація сполучників пов'язана із їх змістовною (семантичною) роллю. Тут виокремлюють, зокрема, *сурядні* (з'єднують рівноправні основи) та *підрядні* (вказують на залежність одних основ від інших) сполучники.

Приклади сурядних сполучників [11]:

- єднальні: та, і, і...і, ні...ні;
- протиставні: а, але, проте, зате, однак;
- розділові: або, чи, хоч, або...або, то...то, не то...не то;
- градаційні: не тільки ... а й, не так... як, не стільки... скільки;
- приєднувальні; і навіть, а ще й, та ще й.

Груп підрядних сполучників багато, вкажемо лише окремі приклади таких сполучників [11]:

- наслідкові: так що, аж та;
- умовні: якщо, якщо...то, якби, коли б, аби, раз;

- мети: щоб, для того щоб, задля того щоб, аби;
- допустові: хоч, хоча, хай, нехай, дарма що, незважаючи на те що;
- причинові: бо, адже, тому що, через те що, за тим що, тим що, оскільки, позаяк, у зв'язку з, тим що;
- часові: як, перед тим як, після того як, відтоді як, з тих пір як, тим часом як, у(в) міру того як, як тільки, тільки-но, тільки що, скоро, ледве щойно;
- порівняльні: як, що, мовби, немов, немовби, неначе, неначебто, ніби, нібито, буцім.

Наведеним описом не обмежується типологія речень. Для розуміння смислу речень також можуть бути важливими, наприклад, різні частини мови, їх граматичні категорії та граматичні значення. Ще складніша ситуація стосується безсполучникових речень, для розуміння яких потрібно залучати змістовний контекст. В рамках цієї статті такі питання розглядатись не будуть.

Тепер коротко обговоримо ті аспекти висловлювань, які пов'язані з приписуванням їм значень істинності та хибності.

Висловлювання, хоча і є спеціальним типом речень, за складністю суттєво не поступаються загальному випадку. Тому наведені раніше типи речень стосуються і висловлювань, правда, тепер з'являється додаткова можливість характеристики висловлювань, індукована їх логічними значеннями. Однак така характеристика не є простою; існують різні підходи до уточнення поняття висловлювання та його структури (дивись, наприклад, [12]).

Наведена класифікація речень, висловлювань, сполучників демонструє різноманітність структур навіть простих висловлювань, труднощі в уточненні їх семантики, зокрема семантики сполучників.

Наприклад, сполучник *і* у природній мові може мати різні значення (поєднальну, приєднувальну, послідовну, темпоральну, наслідкову, протиставну тощо). Також багато значень має і сполучник *або* (розділову, протиставну тощо). Навіть у простому випадку цей сполучник може бути формалізований як слабка диз'юнкція $\vee$ (сполучник *або* вживається в з'єднувально-розділовому значенні), або як сильна диз'юнкція, позначається

$\vee'$ чи $\oplus$ (тут *або* вживається в строго розділовому значенні). Тому формалізація висловлювання '*A або B*' в цьому випадку задається різними таблицями істинності:

<i>A</i>	<i>B</i>	Слабка диз'юнкція ($\vee$)
<i>T</i>	<i>T</i>	<i>T</i>
<i>T</i>	<i>F</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>F</i>	<i>F</i>

та

<i>A</i>	<i>B</i>	Сильна диз'юнкція ($\vee'$ чи $\oplus$)
<i>T</i>	<i>T</i>	<i>F</i>
<i>T</i>	<i>F</i>	<i>T</i>
<i>F</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>F</i>	<i>F</i>

Найбільші дискусії виникають щодо формалізації умовного сполучника *якщо...то*. Традиційно, така формалізація задається операцією матеріальної імплікації, позначається $\rightarrow$ чи $\Rightarrow$, з такою таблицею істинності:

<i>A</i>	<i>B</i>	Матеріальна імплікація ($\rightarrow$ чи $\Rightarrow$)
<i>T</i>	<i>T</i>	<i>T</i>
<i>T</i>	<i>F</i>	<i>F</i>
<i>F</i>	<i>T</i>	<i>T</i>
<i>F</i>	<i>F</i>	<i>T</i>

Проблема з цією формалізацією полягає в тому, що вираз '*якщо A то B*' тлумачиться як (семантичне) слідування *B* з *A*. Це призводить до так званих *парадоксів істинності*, а саме, з хибності випливає (слідує) все, що завгодно, а істинність випливає з будь-чого. Наведемо приклад Бертрана

Рассела: «якщо $1+1=1$, то я – Папа Римський». Таке тлумачення суперечить інтуїтивному розумінню відношення слідування.

Є різні спроби надати обґрунтування такому визначенню імплікації, в тому числі суто штучні, які не відображають суті слідування. Наприклад, в [3] сказано, що «Обґрунтування таблиці істинності для $\Rightarrow$ полягає в тому, що ми хочемо, щоб твердження ‘якщо A та B , то B ’ було істинним у всіх випадках». Дійсно, коли A та B істинні, то A та B істинне, звідки випливає, що і B істинне (обґрунтовує перший рядок таблиці істинності для $\Rightarrow$); якщо A хибне, а B істинне, то A та B хибне, в той же час B є істинним (третій рядок); якщо A хибне та B хибне, то B також хибне (четвертий рядок). Випадок другого рядку в [3] не наводиться.

Таким чином, серйозної аргументації для тлумачення виразу ‘якщо A то B ’ як матеріальної імплікації не наводиться. Набігаючи наперед, відзначимо, що вказані проблеми з матеріальною імплікацією викликані змішенням різних концептів, а саме вираз ‘якщо A то B ’ має дві різних інтерпретації: як операції побудови нового висловлювання та як відношення слідування.

Підсумовуючи, можна сказати, що

- поняття висловлювання (як спеціального речення природної мови) є достатньо складним, існують різні підходи до його визначення. Ця складність пов’язана з неоднозначністю, відкритістю, складністю, відсутністю точного опису природної мови;
 - зокрема, немає точного опису таких спеціальних аспектів речень як їх синтаксису та семантики, зазвичай обмежуються розглядом окремих випадків. Відповідно, немає точного розділення синтаксису та семантики, тобто висловлювання розглядаються на синкретичному рівні;
 - такий підхід до визначення поняття висловлювання є властивим для початкового етапу розвитку логіки, його можна назвати лінгвістично-базованим. Цей підхід веде до низки проблем в логіці.
- Перейдемо до обговорення формальності логіки.

1.5 Про формальність математичної логіки

Традиційні визначення математичної логіки наголошують, що вона основну увагу надає *формі* висловлювань в абстракції від їх змісту.

Таке тлумачення є вкрай невдалим, бо з філософії [13] добре відомо, що форма змістовна, а зміст – оформлений, тобто не можна повністю абстрагуватись від змісту (інакше не було б можливості надати змістовні формулювання формулам логіки). Тому висловлювання мають певний зміст, який потрібно експлікувати. Як же тлумачити формальність логіки?

Справа в тому, що абстрагування від змісту стосується не логіки в цілому, а лише її окремої складової, а саме, теорії доведень, зокрема, дедуктивної логіки. Але і тут абстрагування є частковим, бо обґрунтування правил доведень спирається на їх зміст, їх семантику.

Таким чином, твердження про формальність логіки (іноді таке твердження навіть називають принципом логіки) не є універсальним, а є частковим, яке може ввести в оману студентів, які вивчають логіку.

Подібна ситуація має місце і для так званих законів логіки.

1.6 Про закони логіки

Традиційний підхід до тлумачення законів логіки полягає в тому, що логіка розглядається як наука, відповідно, закони мають бути науковими законами. Наприклад, в підручнику [14] сказано, що «логічним законом називають внутрішній суттєвий, необхідний зв'язок між логічними формами у процесі побудови міркувань». Але чи дійсно традиційні закони логіки відображують внутрішній суттєвий зв'язок?

Розглянемо, наприклад, закон виключеного третього. Його суть полягає в тому, що висловлювання є або істинним, або хибним, інших варіантів немає; тобто розглядаються лише два логічних значення (бівалентність логіки). А, наприклад, висловлювання 'сьогодні буде дощ' може бути істинним, хибним чи невизначеним (тризначність логіки). Тому так звані закони логіки (тотожності, несуперечливості, виключеного

третього, достатньої підстави) насправді не є універсальними науковими законами. Як же тоді слід тлумачити ці закони?

Можна спробувати описати обмежену область для якої вказані «закони» дійсно будуть науковими законами. Але тут проблема полягає в тому, що чітко описати таку область не вдається. Тому «закони» насправді тлумачать як *постулати*, тобто як твердження, які в рамках певної теорії приймаються за істину. Тим самим ці постулати визначають логіку, в нашому випадку – класичну логіку, а не є похідними у наперед визначеній області.

Тим не менше, термін ‘закони логіки’ є широко вживаним. Це означає, що в цьому випадку термін ‘закон’ слід тлумачити не як науковий закон, а як *нормативний закон*, який для певної області встановлює норми, правила, стандарти, зразки тощо.

Як же співвідносяться поняття наукового та нормативного законів? Розглянемо приклад: батько встановив для дочки закон – повертатись додому до 22 години. Тут можна говорити

- 1) про велику *суб’єктивність* цього закону (воля батька);
- 2) про велику *нормативність* закону у даній сім’ї;
- 3) про певну *обґрунтованість* закону, скажімо через розклад руху транспорту;
- 4) про певну *об’єктивність* закону, скажімо, у силу обмежень комендантської години;
- 5) про слабку *необхідність* закону (не діє в багатьох сім’ях);
- 6) про слабку *науковість* закону тощо.

Як бачимо, ступінь вказаних характеристик закону (суб’єктивність, обґрунтованість, об’єктивність, необхідність, науковість тощо) може варіюватись від сильної до слабкої, також ці характеристики пов’язані між собою і впливають одна на одну. Такі характеристики та їх зв’язки треба досліджувати і для законів логіки.

Таким чином, *традиційні закони логіки не є науковими законами*, вони скоріш є *нормативними законами*, тому у багатьох прикладних областях (юридичній, економічній тощо) логіка визначається як *нормативна наука*.

1.7 Результати аналізу традиційних методологічних засад математичної логіки

Головний результат проведеного аналізу можна коротко сформулювати таким чином:

традиційні методологічні засади математичної логіки є незадовільними.

Розшифруємо цю тезу більш детально, зосередившись на основних моментах традиційних засад.

- *Об'єкт та предмет математичної логіки.* Традиційно, об'єктом математичної логіки є міркування, а предметом – істинні міркування. В більш загальному контексті *об'єктом логіки є мислення, а предметом – правильність мислення.* Зведення логіки до мислення є сильним обмеженням об'єкту та предмету логіки, відповідно і математичної логіки.
- Базовим елементом міркувань є *висловлювання*, яке визначається як речення, якому приписується значення істинності чи хибності. Таке визначення ґрунтується на понятті речення, тому подальше розгортання традиційної логіки базується на лінгвістичному аналізі висловлювань (дивись, наприклад, [2], де перший параграф присвячено саме лінгвістичним аспектам). Для речень природньої мови розкриття їх синтаксичної структури та особливо асоційованого з ними змісту є складним, не формалізованим, ґрунтується на інтуїтивному розумінні речень, не надає ієрархії їх складності тощо. Тому немає критеріїв, які дозволяють виокремити ті типи речень, які ведуть, скажімо, до пропозиційної логіки чи логіки предикатів. Тим самим, таке визначення висловлювання веде до *лінгвістично-базованого підходу*, тобто традиційна логіка будується на засадах іншої науки – лінгвістики, а не засадах самої логіки; на труднощі такого підходу вже в античності звертали увагу софісти. Крім того, традиційна логіка визначається як бівалента, обґрунтування багатозначних логік відсутнє.

- Особливістю традиційної логіки є її *формальність*, яка говорить про дослідження форми висловлювань в абстракції від їх змісту. Такий підхід викривляє тлумачення висловлювань, звужує їх до їх форми, тому він обмежує логіку, фактично зводячи її лише до теорії доведень.
- *Традиційні закони логіки не є науковими законами*, їх можна розглядати як нормативні закони, строгого обґрунтування вибору саме класичних законів немає.

Наведена критика традиційних методологічних засад актуалізує проблему розбудови власних методологічних засад математичної логіки та логіки в цілому. З цією метою пропонується діяльнісно-базований підхід.

2. Діяльнісно-базований підхід до методологічних засад математичної логіки

2.1 Методологічні принципи та схеми підходу

Базові принципи пов'язані з онтологією, гносеологією (епістемологією) та телеологією:

- принцип універсального взаємозв'язку;
- принцип сходження (розвитку) від абстрактного до конкретного;
- принцип орієнтації на математичну логіку.

Наведемо короткі пояснення цих принципів.

Загальний принцип універсального взаємозв'язку можна знайти в різних формулюваннях у філософії, науці та логіці (дивись, наприклад, [15]). Принцип стверджує, що ніщо у світі не існує саме по собі, ізольовано; кожен об'єкт, річ, процес пов'язані з усіма іншими, об'єднуючи їх в єдине ціле. Як наслідок, усі поняття логіки взаємопов'язані та впливають одне на одного. Цей принцип зазвичай трактується як *онтологічний*.

Загальний принцип сходження від абстрактного до конкретного був чітко сформульований та використаний Гегелем [13] як основний метод пізнання реальності. Тут «абстрактне» означає одностороннє, а «конкретне» — єдність багатьох сторін. Як наслідок, основні поняття логіки можна

віднести до певних рівнів абстракції та конкретизації. Цей принцип зазвичай трактується як *гносеологічний*.

Специфічний принцип орієнтації на математичну логіку означає, що серед десятків категорій та понять враховуються ті, які пов'язані з математичною логікою, її теорією та практикою. Цей принцип можна розглядати як *телеологічний*, оскільки він формулює цілі наших побудов.

Наведені принципи окреслюють загальні характеристики підходу та повинні бути підкріплені певними *конкретними схемами*. Багато таких схем можна знайти в літературі. Тут ми орієнтуємося на Гегеля [13], обираючи ті схеми, які використовуються в наших побудовах:

Зняття: це одна з основних категорій у системі логіки Гегеля. Інтуїтивно, зняття означає заперечення чогось, але збереження його в якійсь іншій формі. Примітивні приклади: вершки – це зняте молоко, масло – це зняті вершки. Для простішого розуміння терміну ‘зняття’ можна тлумачити його як побудову адекватної *моделі*.

Заперечення заперечення: щось, що заперечується, призводить до протилежності, яка, будучи запереченою, дає нове, яке зберігає основні характеристики початкового. Примітивний приклад: яблуна заперечується своїм насінням, яке, у свою чергу, заперечується новою яблунею.

Діалектична тріада: щось (теза) призводить до антитези, які об'єднуються у синтезі. Примітивний приклад: теща – це теза, зять – антитеза, а дочка тещі, яка є дружиною зятя, – синтез.

Діалектична тетрада: ціле розщеплюється на дві протилежності, які об'єднуються в конкретному. Примітивний приклад: висловлювання має синтаксичні та семантичні аспекти, формалізовані відповідно формулою та предикатом, які об'єднані відношенням інтерпретації. Перехід від цілого до конкретного можна розглядати як зняття. У лінійній формі тетраду будемо позначати таким чином:

ціле – (протилежа_{не1} + протилежа_{не2}) – конкретне

Подвійна діалектична тетрада: конкретне першої діалектичної тетради розглядається як ціле, що за допомогою другої тетради призводить до нового конкретного. Відповідні протилежності двох тетрад пов'язані зняттям.

2.2 Основні поняття логіки та їх взаємозв'язки

Сформульовані принципи та схеми дозволяють перейти до ідентифікації основних понять логіки (детальніше в [16]).

Можна стверджувати, що найважливіше для людини у світі – це *діяльність*, а в діяльності найважливіше – її *істина*, а для істини найважливіше – її *закони*, а закони істини – це *логіка*. Це означає, що поняття *Світ*, *Діяльність*, *Істина* та *Логіка* пов'язані відношенням зняття. Тому наступний крок полягає в поданні понять *Діяльність*, *Істина* та *Логіка* як певних конкретних за допомогою схеми діалектичної тетради. Більше того, оскільки ця схема застосовується послідовно, можна говорити про подвійну, а потім про потрійну тетраду. Залишається виявити протилежні поняття в кожній з тетрад.

Доцільно почати з розкриття поняття *Діяльність*. Дійсно, щоб зрозуміти логіку, потрібно знати, що таке істина, а оскільки істина — це знята діяльність, то повинні почати саме з діяльності.

Згідно зі словниковим визначенням, діяльність — це процес активної взаємодії між *суб'єктом* та *об'єктом*. Суб'єкт — це той, хто виконує дію або від кого спрямована певна діяльність. Відповідно, ця діяльність спрямована на об'єкт. Об'єкт — це та частина об'єктивної реальності, яку суб'єкт вже залучив до своєї діяльності.

Зазначені характеристики дозволяють нам застосувати наступну схему діалектичної тетради для побудови поняття *Діяльності* як конкретного у тетраді

$$\text{Світ} - (\text{Суб'єкт} + \text{Об'єкт}) - \text{Діяльність}.$$

Наступний крок – уточнення істини. Цей крок є вирішальним, оскільки він уточнює *основну характеристику (предмет) логіки*. Тут можемо використовувати загальноприйняте визначення істини як відповідності мислення реальності (об'єкту), а також той факт, що головною характеристикою суб'єкта є мислення (як знятого суб'єкта). Але що буде знятим об'єктом? Оскільки головними характеристиками об'єкта є дії,

пов'язані з ним, то знятий об'єкт повинен представляти такі дії. Ми пропонуємо назвати це поняття *Роблення*.

Ці міркування можна подати наступною діалектичною тетрадою:

Діяльність – (Мислення + Роблення) – Істина.

Нарешті, слід здійснити конкретизацію відношення *Істина – Логіка*. Оскільки Логіка є наукою, то переходимо від Істини (як єдності Мислення та Роблення) до їхніх наук. Отримуємо нову діалектичну тетраду:

Істина – (Наука Мислення + Наука Роблення) – Наука Істини (Логіка).

Зауважимо, що подальша конкретизація Науки Мислення та Науки Роблення веде до таких розділів математичної логіки як Теорія доведень та Теорія моделей.

Поняття *Діяльності* є центральним; тому запропонований підхід можна назвати *діяльнісно-базованим*.

Підсумовуючи, можна сформулювати такі взаємозв'язки між поняттями логіки:

- Логіка – це Наука про Істину (є знятою Істиною); Істина – це знята Діяльність; Діяльність – це знятий Світ.
- Діяльність – це знятий Світ, який проявляється як єдність Суб'єкта та Об'єкта.
- Істина – це знята Діяльність, яка проявляється як єдність Мислення та Роблення.
- Логіка (як Наука Істини) – це єдність Науки Мислення та Науки Роблення.
- Наука Мислення – це зняте Мислення, Мислення – це знятий Суб'єкт.
- Науки Роблення – це знята Діяльність, Роблення – це знятий Об'єкт.

Ще раз відмітимо, що для розуміння наведених формулювань філософський термін 'зняття' можна тлумачити як побудову загальної моделі, скажімо, формулювання «Логіка є знятою Істиною» означає, що в Логіці моделюються (уточнюються) основні закони пізнання та оперування з Істиною.

Для подальшого розвитку понять логіки слід більш детально описати діяльність.

2.3 Рівні та типи діяльності

Дослідження поняття діяльності розпочнемо з ідентифікації його рівнів та типів. Рівні діяльності в першу чергу визначаються рівнями розгляду об'єктів. Серед таких рівнів виокремлюємо *наявний рівень* (об'єкти характеризуються їх *якостями*), *реальний рівень* (об'єкти характеризуються їх *властивостями*), *дійсний рівень* (об'єкти характеризуються їх *діями*).

Наведені рівні індукують відповідні системи математичної логіки, а саме:

- наявний рівень індукує *пропозиційну логіку* (деталі описано в [17; 18]),
- реальний рівень – *логіку предикатів* (деталі описано в [19]),
- дійсний рівень – *модальну логіку, програмну логіку* тощо (формальні визначення наведено в [20]).

Отже, виокремлення рівнів діяльності природним чином веде до конвенційних логік. Зауважимо, що при побудові відповідних логік немає потреби в аналізі різних класів висловлювань, як це відбувається згідно лінгвістично-базованого підходу, навпаки, аналіз діяльності веде до висловлювань та їх структури.

Що стосується типів діяльності, то основними з них є *розпізнавання* та *створення* об'єктів. Діяльність розпізнавання може бути визначена як діяльність *прийняття* (акцептації) або *класифікації*. Почнемо з діяльності прийняття як відносно абстрактної (простої), та яку можна асоціювати з раніше описаними рівнями діяльності. Інтуїтивно, діяльність прийняття виокремлює підклас прийятних об'єктів в заданому класі об'єктів. Уточнення такої діяльності дозволяє побудувати певну формальну логічну систему (логіку), яка матиме адекватне обґрунтування в рамках діяльнісно-базованого підходу. Продемонструємо таке обґрунтування для діяльності прийняття.

2.4 Діяльність прийняття та її експлікація

Інтуїтивно, діяльність прийняття полягає в відборі прийнятних об'єктів в класі заданих об'єктів. Незважаючи на простоту такого формулювання, є певні моменти, які потребують уточнення. Розглянемо простий приклад: треба купити кислі яблука для компоту. Можливі такі варіанти:

- ви вибираєте кілька кислих яблук. Інші яблука вас не цікавлять, тобто діяльність задається лише позитивним описом;
- ви вибираєте кілька яблук кислих сортів, але не берете перезрілих. Тут діяльність задається позитивно-негативним описом;
- ви вибираєте кілька кислих яблук, але їх недостатньо для компоту, перезрілих яблук брати не можна, але ви готові взяти кілька не кислих й не перезрілих яблук (слабкий позитивно-негативний опис, тобто прийнятними є також об'єкти, які не підпадають під позитивний та негативний опис).

Яким же чином можна формалізувати різні варіанти діяльності прийняття?

Серед багатьох можливостей оберемо просту формалізацію семантичного (екстенсіонального) типу, використовуючи *тотальні* та *часткові предикати*. Для заданого класу об'єктів *OBJ* позитивний опис семантично задається частковим відображенням з *OBJ* в множину $\{T\}$, яка складається з однієї константи *T*; негативний опис семантично задається частковим відображенням з *OBJ* в множину $\{F\}$, яка складається з однієї константи *F*; слабкий позитивно-негативний опис семантично задається частковим відображенням з *OBJ* в множину $\{T, F\}$. Зауважимо, що невизначеність предиката (позначаємо символом *u*) означає, що відповідний об'єкт не підпадає під прийняття чи неприйняття. Тому для останнього випадку частковість предиката може бути подана тотальним тризначним предикатом.

Проілюструємо формалізацією таких випадків в рамках діяльнісно-базованого підходу для сполучника природньої мови 'якщо...то'.

Почнемо з прикладу: робимо обумовлений вибір (прийняття) кросівок таким чином: якщо у кросівках білий верх, то подошва має бути чорною. Тут

є два описи: 1) білий верх, 2) чорна підошва. Позитивна частина полягає в тому, що кросівки вибираємо, якщо одночасно вони мають білий верх і чорну підошву. Що можна сказати про негативну складову? Мабуть доречно вважати, що якщо у кросівках білий верх, а підошва не чорна, то такі кросівки не приймаються (бо в цьому суть обумовленого прийняття). Залишаються два випадки, коли умова (білий верх) не виконується. В описі про такий випадок нічого не говориться, тобто можна вважати, що цей випадок не описаний. Це означає, що обумовлений вибір задається такою частковою (тризначною) таблицею істинності:

A	B	Обумовлене прийняття (в випадку A має бути B)
T	T	T
T	F	F
F	T	и
F	F	и

Ця таблиця достатньо адекватно формалізує обумовлене прийняття. Проблема виникає тоді, коли *приймається постулат (закон) виключеного третього* (як у традиційній логіці). Це означає, що в таблиці значення *и* треба замінити на *T* або *F*. Виникають наступні чотири випадки:

A	B	(1)
T	T	T
T	F	F
F	T	F
F	F	F

A	B	(2)
T	T	T
T	F	F
F	T	F
F	F	T

A	B	(3)
T	T	T
T	F	F
F	T	T
F	F	F

A	B	(4)
T	T	T
T	F	F
F	T	T
F	F	T

Випадок (1) задає кон'юнкцію *A* та *B*, випадок (2) задає еквівалентність *A* та *B*, а випадок (3) співпадає з *B*. Ці випадки не відповідають ідеї обумовленого прийняття. Залишається випадок (4), який можна прийняти як формалізацію обумовленого прийняття (об'єкт з ознакою *A* приймається, якщо об'єкт також має ознаку *B*). Але цей випадок є матеріальною імплікацією. Тим самим надано обґрунтування матеріальної імплікації як формалізації обумовленого прийняття для двозначної діяльності.

Проблема з матеріальною імплікацією в традиційній логіці виникає тому, що сполучник '*якщо...то*' пробують тлумачити як відношення

слідування, а це призводить до парадоксів істинності. Таким чином треба розрізняти два різних поняття: 1) операцію обумовленого прийняття (формалізується матеріальною імплікацією) та 2) відношення слідування, яке, до речі, може мати багато різних визначень, скажімо слідування за істиною, слідування за хибністю тощо (дивись деталі в [20]).

Висновки

Коротко опишемо розглянуті відмінності *традиційного (ТП)* та *діяльнісно-базованого (ДП) підходів* до методологічних засад математичної логіки.

- А. За ТП об'єктом математичної логіки є *міркування*, предметом – *правильні міркування* (для логіки в цілому об'єктом є *мислення*, предметом – *правильне мислення*).
В. За ДП об'єктом є *діяльність*, предметом – *істина* діяльності. Оскільки мислення є складовою діяльності, то ДП можна розглядати як збагачення ТП. Тим самим ДП не протистоїть ТП, але більш адекватно відображає засади логіки. Безумовно, ДП виглядає складніше ніж ТП, скажімо, як автомобіль виглядає складніше велосипеда, але надає значно більше можливостей.
- А. Базовим поняттям ТП є *висловлювання* – речення з істиннісними значеннями. Це призводить до того, що логіка базується на засадах лінгвістики, тому ТП є лінгвістично-базованим. Підхід також можна назвати *синтактико-семантичним*, бо перевага віддається формі висловлювань. Оскільки природна мова є надзвичайно складною для формалізації, це призводить до складнощів в розбудові логіки.
В. Базовим поняттям ДП є поняття *дії*, яке з'єднує мислення та роблення, тут перевага віддається семантиці, а це означає, що ДП є *семантико-синтаксичним* підходом. Додатковою перевагою є те, що дії гарно класифікуються за рівнями та типами, чого немає в ТП; відповідно, ДП надає адекватне обґрунтування таким системам математичної логіки як пропозиційна логіка, логіка предикатів, модальна логіка, програмна логіка тощо.

- А. ТП формулює принцип *формальності* логіки, концентруючись на формі висловлювання в абстракції від його змісту. Таке абстрагування збіднює логіку, зводячи її до теорії доведень.

В. За ДП формальність логіки є похідною від властивостей дій, тому формальність знаходить своє адекватне обґрунтування в ДП.

- А. За ТП закони логіки фактично є *постулатами*, які не знаходять свого обґрунтування в предметних областях. Такі закони можна назвати *нормативними*.

В. За ДП закони логіки дійсно є *законами областей* діяльності, бо ДП надає достатньо потужні методи опису таких областей.

Безумовно, є ще багато відмінностей у підходах, які не розглядались у рамках цієї статті.

Вказані особливості та переваги діяльнісного-базованого підходу доцільно втілити у курс «Математична логіка», що дозволить зробити його більш прозорим, зв'язним та обґрунтованим.

Список використаних джерел

1. Church Alonzo. Introduction to mathematical logic. Volume I. Second printing. Princeton University Press, Princeton 1958, x + 378 pp.
2. Kleene Stephen Cole. Mathematical logic. John Wiley & Sons, Inc., New York, London, and Sydney, 1967, xiii + 398 pp.
3. Mendelson Elliott. Introduction to mathematical logic, sixth edition. CRC Press, 2015, 499 pp.
4. Shoenfield Joseph R. Mathematical Logic. Addison-Wesley, 1967, 334 pp.
5. Copi Irving M. Introduction to Logic. 2nd edition, New York: Macmillan Company, 1962, 512 pp.
6. Freund A. An Introduction to Mathematical Logic, 2023, 86 pp. arXiv preprint arXiv:2310.09921, 2023•arxiv.org
7. Bass Sam. Introduction to Mathematical Logic, 2023, 294 pp. Режим доступу https://mathweb.ucsd.edu/~sbuss/IntroMathLogic/Fullbook_Draft.pdf
8. Gensler Harry J. Introduction to Logic. New York, Routledge, 3rd Edition, 2017, 428 pp.
9. Bhattacharyya S. The Concept of Logic. Philosophy and Phenomenological Research, Vol. 18, No. 3 (Mar., 1958), pp. 326-340.

10. Robison Gerson B. An introduction to mathematical logic. Prentice-Hall, Inc., Engle wood Cliffs, N.J., 1969, xii + 212 pp.
11. Плющ М.Я. Граматика української мови. Морфеміка. Словотвір. Морфологія. Підручник. 2-е видання, доповнене. – К.: Видавничий Дім «Слово», 2010. – 328 с.
12. Feys R., Fitch F.B. Dictionary of Symbols of Mathematical Logic. North-Holland, 1969, 171 pp.
13. Hegel G. W. F. The Science of Logic / translated and edited by George di Giovanni, Cambridge: Cambridge University Press, 2010.
14. Конверський А.Є. Логіка (традиційна та сучасна): Підручник. – Київ: Центр навчальної літератури, 2004. – 535 с.
15. Spirkin A. Dialectical Materialism. Progress Publisher, Moscow, 1983, 343 pp.
16. Nikitchenko M. Revisiting Data-Information-Knowledge-Wisdom Hierarchy from a Logic Perspective. 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), 2021. – PP. 2159-2165, doi:10.1109/HPCC-DSS-SmartCity-DependSys53884.2021.00322.
17. Nikitchenko M. Towards methodological foundations of propositional logic. Logic & Artificial Intelligence / Vladimir Andrunachievici Institute of Mathematics and Computer Science; edited by Svetlana Cojocaru [et al.]. – Chişinău : S. n., 2023 (Valinex). – PP. 171-185.
18. Нікітченко М. Трирівнева схема експлікації базових понять у викладанні курсу «Математична логіка». Математична логіка та програмування. Досвід викладання: Колективна монографія. – Одеса: Видавничий дім «Гельветика», 2022. – С. 7–20.
19. Нікітченко М. Трирівнева схема експлікації базових понять логіки предикатів у викладанні курсу «Математична логіка». Математична логіка та програмування. Досвід викладання. 2024–2025 навчальний рік: Збірник праць. – Видавничий дім «Гельветика», 2025. – С. 7–17.
20. Нікітченко М. С., Шкільняк С. С. Прикладна логіка. – К.: ВПЦ "Київський університет", 2013.

Шкільняк Степан Степанович, д.ф.-м.н, професор

НЕТРАДИЦІЙНІ ТА ПРОГРАМНО-ОРІЄНТОВАНІ ЛОГІКИ В КУРСІ «МАТЕМАТИЧНА ЛОГІКА»

Робота присвячена опису програми навчальної дисципліни «Математична логіка» для студентів спеціальності F3 «Комп'ютерні науки» освітньо-професійної програми «Інформатика». Характерною ознакою цієї програми є приділення особливої уваги нетрадиційним логікам – багатозначним, модальним, темпоральним, епістемічним, програмно-орієнтованим логікам часткових квазіарних предикатів. Розширення сфери застосування математичної логіки зумовлює залучення до практичного використання все нових і нових її розділів. Це обґрунтовує включення відповідних тем до робочої програми навчальної дисципліни «Математична логіка». В роботі стисло описано зміст тем, пов'язаних з нетрадиційними логіками. Ознайомлення з такими логіками та можливостями їх практичного застосування дуже важливе для формування висококваліфікованих фахівців в галузі інформатики.

Ключові слова: багатозначна логіка, модальна логіка, темпоральна логіка, епістемічна логіка, логіка часткових квазіарних предикатів

The paper outlines the syllabus for the «Mathematical Logic» course designed for students majoring in «Computer Science» (specialty F3) within the «Informatics» educational-professional program. A distinctive feature of this syllabus is the special emphasis placed on non-traditional logics – such as many-valued, modal, temporal, and epistemic logics, as well as program-oriented logics of partial quasiary predicates. The expanding scope of mathematical logic's application necessitates the practical utilization of an ever-growing number of its branches. This justifies the inclusion of relevant topics in the syllabus of the «Mathematical Logic» course. This paper briefly outlines the content of topics related to non-traditional logics. Familiarity with such logics and their potential practical applications is crucial for training highly qualified specialists in the field of computer science.

Keywords: many-valued logic, modal logic, temporal logic, epistemic logic, logic of partial quasiary predicates

Вступ

Логіка тісно пов'язана з інформатикою та програмуванням. В основі схемотехніки комп'ютерів лежить апарат математичної логіки (алгебра логіки). Саме в межах математичної логіки запропоновано перші формалізації поняття алгоритму, мови програмування базуються на тих чи інших уточненнях цього поняття.

Розвиток інформаційних технологій та їх проникнення в усі сфери діяльності людини веде до невинного розширення сфери застосування математичної логіки. Апарат математичної логіки належить до базових засобів моделювання різноманітних предметних областей, він є основою сучасних систем штучного інтелекту, інформаційних і програмних систем (див., напр., [1; 2]).

Розмаїття задач, для розв'язку яких використовується математична логіка, зумовлює залучення до практичного використання все нових і нових її розділів. Особливе місце серед них посідають нетрадиційні логіки. В першу чергу серед них можна виділити багатозначні логіки, модальні логіки, програмно-орієнтовані логіки часткових квазіарних предикатів (див., напр., [2–6]). Ознайомлення з такими логіками та можливостями їх практичного застосування дуже важливе для формування фахівців з інформаційних технологій. Тому робоча програма навчальної дисципліни «Математична логіка» (див. [7]) включає відповідні теми.

Програмно-орієнтовані логіки часткових квазіарних предикатів

На даний момент розроблено багато різноманітних логічних систем, які успішно використовуються в інформатиці. Основою цих систем зазвичай виступає класична логіка. Проте ця логіка має принципові обмеження, що ускладнює її використання. Класична логіка недостатньо враховує структурованість, неповноту, частковість інформації про предметну область. Тому набуває актуальності задача побудови нових, програмно-орієнтованих логік. Таку побудову природно вести на базі спільного для логіки й програмування композиційно-номінативного підходу. Логіки, побудовані на основі такого підходу, названо композиційно-номінативними (КНЛ). Передумовою виникнення цих логік стала необхідність посилення можливостей класичної логіки для розв'язку нових задач штучного інтелекту та інформатики. КНЛ базуються на загальних класах часткових відображень, заданих на наборах іменованих значень. Такі відображення названо квазіарними. Тому КНЛ називають логіками квазіарних предикатів.

Основні поняття і визначення, пов'язані з квазіарними функціями і предикатами та засобами їх побудови висвітлені в першій частині дисципліни. Такі засоби трактуються як алгебраїчні операції над предикатами – композиції. Такими є традиційні логічні зв'язки і квантори, а також композиції реномінації (перейменування), суперпозиції та спеціальні предикати і композиції рівності. Обґрунтовано необхідність побудови програмно-орієнтованих логік. Наведено спектр логік квазіарних предикатів. Також розглянуто реноминативні й безкванторні логіки. Описано семантичні моделі та мови цих логік.

Друга частина дисципліни присвячена логікам першого порядку. Окрім класичних логік першого порядку, розглядаються першопорядкові логіки монотонних (еквітонних) предикатів. Такі логіки зберігають основні властивості класичної логіки при істотному розширенні класу семантичних моделей, тому їх названо неокласичними. Для цих логік визначається відношення неспростовнісного логічного наслідку для пар формул та для пар множин формул, наводяться властивості цього відношення. Описуються нормальні форми для формул мови неокласичних логік.

В третій частині дисципліни розглядаються числення логік першого порядку. Для класичних логік описуються числення Гільбертівського типу, розглядаються фундаментальні поняття несуперечливості та повноти, теорема Гьоделя про повноту та її наслідки. Для класичних логік розглянуто найпопулярніший метод пошуку виведень – метод резолюцій, також описано числення Генценівського типу – секвенційні числення. В неокласичних логіках метод резолюцій не працює, тому ефективний пошук виведень в неокласичних логіках дають змогу робити саме секвенційні числення. Розглядаються приклади побудови виведень в секвенційних численнях, для таких числень доводяться теореми про побудову контрмоделі, теореми коректності та повноти.

Четверта частина дисципліни цілком присвячена неокласичним логікам. Розглядаються першопорядкові логіки квазіарних предикатів без обмежень монотонності. Описуються різновиди квазіарних предикатів, визначаються їх мови та класи інтерпретацій (семантики). Вводиться низка відношень логічного наслідку $P \models_{IR}, P \models_T, P \models_F, P \models_{TF}, R \models_{TF}$ для пар формул та для пар множин формул, визначаються властивості цих відношень та співвідношення між ними. Також розглядаються логіки вищих порядків, інтуїціоністські логіки. Особлива увага приділена багатозначним логікам та логікам модального типу.

Багатозначні логіки

Традиційні логіки двозначні, або бівалентні: множина істиннісних значень таких логік двоелементна. Цю множину традиційно позначають $\{T, F\}$; також її позначають $\{1, 0\}$.

Багатозначні логіки вперше з'явилися на початку 20 ст. В 1920 році Ян Лукасевич на основі аналізу Аристотелевої проблеми про істиннісний статус майбутніх подій прийшов до висновку, що принцип бівалентності для цього не підходить. Він вводить *третє* істиннісне значення, трактоване як "нейтрально", "невизначено", пізніше як "можливо". Далі Я. Лукасевич визначає логічні зв'язки 3-значної логіки L_3 :

Опис модальних висловлень за допомогою 3-значної логіки виявився не зовсім адекватним, далі розвиток модальних логік пішов іншими шляхами. Найважливішим є той факт, що Я. Лукасевич довів саму *можливість* існування неklasичних логік, що за значимістю сумірне з відкриттям неевклідових геометрій.

В 1921 р. Е. Пост запропонував низку n -значних логік P_n . На відміну від підходу Я. Лукасевича, це було зроблено формально, без семантичного обґрунтування. Пост зробив припущення, що логічні функції (предикати) визначені й набувають значення на n -елементній множині $\{t_1, t_2, \dots, t_n\}$. При цьому Поста не цікавив логічний смисл виразу "значення t_k ". Зазвичай $\{t_1, t_2, \dots, t_n\}$ – це $\{0, 1, \dots, n-1\}$. При $n=2$ логіка P_n стає класичною пропозиційною логікою C_2 .

Пост запропонував два варіанти заперечення – традиційне $\neg_p$ (як у Лукасевича) і циклічне $\sim_p$. Основною властивістю логік P_n з базовими зв'язками $\vee_p$ та циклічним $\sim_p$ є їх функціональна повнота: через $\vee_p$ та $\sim_p$ можна виразити кожну функцію n -значної логіки.

До найвідоміших 3-значних належать сильна логіка Кліні K_3 та слабка логіка Кліні K_3^w . Вони запропоновані для використання в теорії рекурсії. Сильна логіка Кліні застосовується в системах алгоритмічних алгебр, мовах табличних баз даних.

Логічні зв'язки сильної логіки Кліні фактично такі ж, як логічні зв'язки традиційної (бівалентної) логіки часткових однозначних предикатів.

Цікавою є 3-значна логіка Бочвара (логіка абсурду): тут третє істиннісне значення трактується як "беззмістовно". Це фактично слабка логіка Кліні, розширена так званими зовнішніми логічними зв'язками, в яких для результатів ототожнено хибність та беззмістовність.

Серед 3-значних варто відзначити проміжні логіки Кліні $K_3 \rightarrow$ (**Lisp**) та $K_3 \leftarrow$, логіку Гейтінга-Гьоделя G_3 , низку параконсистентних логік.

Найвідомішими параконсистентними 3-значними логіками є K_3^2 та **PCont**. K_3^2 – це Клінієва K_3 з виділеними істиннісними значеннями $\{T, \perp\}$.

Параконсистентність означає, що не діє закон *ex falso quodlibet*, тому не завжди маємо $A, \neg A \models B$. -Це дає змогу будувати нетривіальні теорії із суперечливою інформацією.

Проміжні логіки Кліні $K_3^{\rightarrow}$ та $K_3^{\leftarrow}$ цікаві ти, що в них логічні зв'язки $\vee$ та $\&$ стають не комутативними.

Цікавим є той факт, що різні мови програмування використовують різні логіки: K_3 – це логіка Prolog, K_3^w – це логіка Pascal; $K_3^{\rightarrow}$ – логіка Lisp.

Дослідження 4-значних логік започаткував Я. Лукасевич. Найцікавішою з них є **L-модальна** логіка, базована на логіці C_4 .

Особливої уваги серед 4-значних заслуговує логіка Белнапа **B₄**. Це зумовлено її застосуванням для опису інформаційних систем із неповною та суперечливою інформацією

Варто відзначити особливу роль логіки Клівні K_3 серед 3-значних логік та логіки Белнапа **B₄** серед 4-значних: алгебра Кліні логіки K_3 ізоморфна алгебрі часткових однозначних бівалентних предикатів; алгебра Белнапа логіки **B₄** ізоморфна алгебрі часткових неоднозначних бівалентних предикатів реляційного типу.

Якщо множина істиннісних значень нескінченна, то логіку називають *нескінченнозначною*. Такі логіки запропонував Я. Лукасевич, він розглядав можливість градації істиннісних значень від 0 до 1 з неперервного інтервалу $[0, 1]$ (зліченнозначні та континуальні логіки). Цей підхід привів до ймовірнісних та до нечітких логік.

Поширеним різновидом нескінченнозначних логік є узагальнення класичної бівалентної C_2 на неперервний інтервал істиннісних значень $[0, 1]$.

Логічні зв'язки такої нескінченнозначної логіки C_∞ вводяться так:

$$\begin{aligned}\neg_c P &= 1 - P; & P \rightarrow_c Q &= \max(1 - P, Q); \\ P \vee_c Q &= \max(P, Q); & P \&_c Q &= \min(P, Q)\end{aligned}$$

Модальні логіки

На відміну від традиційних логік, які орієнтовані на опис одного конкретного стану світу, модальні логіки враховують можливість його зміни і розвитку, що дає змогу застосовувати модальні логіки для аналізу та моделювання різноманітних аспектів діяльності людини. Тому модальні логіки в останній час мають велике застосування в прикладних галузях, найперше це темпоральні логіки та епістемічні логіки. Темпоральні логіки застосовуються для опису та моделювання складних динамічних систем,

специфікації та верифікації програм; епістемічні логіки використовуються в системах штучного інтелекту та інформаційних системах, в експертних системах.

Розглядаються основні різновиди модальних логік: атлетичні, темпоральні, епістемічні, а також деонтичні. Описуються мови цих різновидів модальних логік на пропозиційному рівні, особлива увага приділена реляційній семантиці цих логік (семантика можливих світів). Наводяться аксіоматичні системи атлетичних, темпоральних, епістемічних логік пропозиційного рівня. Розглядається зв'язок між модальними аксіомами та властивостями відношення досяжності на світах.

Виділяються різновиди темпоральних логік: лінійні (ЛТЛ) та розгалужені (РТЛ), описуються аксіоматичні системи пропозиційної ЛТЛ. Відзначається роль темпоральних логік для специфікації та верифікації програм.

Розглядаються системи епістемічної логіки знання з одним агентом (експертом) знання та мультиагентні (мультіекспертні) системи. Відзначається роль епістемічних логік для розробки мультиагентних систем штучного інтелекту, здатних ефективно координувати дії та приймати рішення.

Можливості класичних модальних логік та логік часткових квазіарних предикатів поєднують транзиційні модальні логіки часткових квазіарних предикатів, вони враховують аспект зміни та розвитку предметних областей. Семантичною основою таких логік є транзиційні модальні системи (ТМС).

Виділяються різновиди транзиційних модальних систем: загальні, темпоральні, мультимодальні. Описуються мови першопорядкових ТМС, дається визначення формули та відображення інтерпретації. Досліджується взаємодія модальних композицій $\Box$ та $\Diamond$ із реномінаціями та кванторами. Взаємодія $\Box$ та $\Diamond$ із кванторами буде різною для ТМС еквітонних предикатів та ТМС предикатів без обмежень еквітонності.

Висновки

Розширення сфери застосування математичної логіки зумовлює необхідність забезпечення належного рівня викладання цієї дисципліни із врахуванням останніх наукових досягнень в галузі математичної логіки, адаптації матеріалу до потреб інформатики та штучного інтелекту.

Навчальна дисципліна «Математична логіка» дає змогу набуту базові знання цієї галузі науки, знати основні поняття, засоби і методи математичної

логіки, їх застосування в штучному інтелекті, інформатиці й програмуванні; мати сучасні уявлення про основні методи пошуку доведень та засоби логічного виведення. Все нові розділи математичної логіки залучаються до практичного використання, тому в програмі навчальної дисципліни особлива увага приділена нетрадиційним логікам – багатозначним, модальним, темпоральним, епістемічним, програмно-орієнтованим логікам часткових квазіарних предикатів. Оволодіння основними поняттями і методами математичної логіки та можливостями їх практичного застосування вкрай важливе для формування висококваліфікованих фахівців в галузі інформатики.

Список використаних джерел

1. S. Abramsky, D.M. Gabbay and T.S.E. Maibaum (eds), Handbook of Logic in Computer Science, Vol. 1-5, Oxford University Press, 1993-2000.
2. M. Huth and M. Ryan, Logic in Computer Science, Second Edition, Cambridge University Press, 2004.
3. J.-J. Ch. Meyer, W. Van der Hoek, Epistemic logic for AI and Computer Science, Cambridge University Press, 2009.
4. V. Goranko, Temporal Logics, Cambridge University Press, 2023.
5. Шкільняк С.С. Математична логіка. Приклади й задачі. Навчальний посібник. – Київ: ВПЦ Київський університет, 2022. – 304 с.
6. Шкільняк С.С. Програмно-орієнтовані логіки квазіарних предикатів. Навчальний посібник. – Київ: ВПЦ Київський університет, 2025. – 423 с.
7. Шкільняк С.С. Робоча програма навчальної дисципліни «Математична логіка» для студентів спеціальності ФЗ «Комп'ютерні науки», освітня програма «Інформатика». – Київ: Київський національний університет імені Тараса Шевченка, 2026. – 11 с.

РОЗДІЛ 2. ПРАКТИЧНІ АСПЕКТИ ВИКЛАДАННЯ ТА ІНТЕГРАЦІЇ КУРСІВ ПРОГРАМУВАННЯ

Криволап Андрій Володимирович, к.ф.-м.н

ОСОБЛИВОСТІ ВИКОРИСТАННЯ МОВИ ПРОГРАМУВАННЯ PYTHON У КУРСІ «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

Роботу присвячено дослідженню можливості та доцільності використання мови програмування Python для практичних демонстрацій та основи практичних програмних проєктів у рамках дисципліни «Об'єктно-орієнтоване програмування», зокрема розглянуто основні принципи ООП, шаблони проектування та принципи SOLID.

Ключові слова: парадигми програмування, мови програмування, об'єктно-орієнтований

The work is dedicated to the study of the possibility and feasibility of using the Python programming language for practical demonstrations and as the basic language for practical program projects within the discipline "Object-Oriented Programming", in particular, the basic principles of OOP, design patterns, and SOLID principles are considered.

Keywords: paradigms of programming, programming languages, object oriented

Вступ

У той час, як вивчення принципів парадигми ООП[1] є основою теоретичного аспекту дисципліни «Об'єктно-орієнтоване програмування», практичний аспект передбачає використання однієї, або декількох мов програмування для формулювання прикладів та виконання практичних програмних проєктів. Вибір такої мови є критичним, та повинен враховувати різні критерії, серед яких, зокрема, підтримка основних концепцій ООП; залучення в межах інших дисциплін; складність вивчення та застосування; можливості подальшого використання поза навчальним контекстом; перспективи розвитку мови та її популярність. Зважаючи на вивчення мови Python у рамках дисципліни «Програмування» певних освітніх програм, доцільним є дослідження можливості використання даної мови в тому числі і для дисципліни «Об'єктно-орієнтоване програмування».

Для розв'язання означеної проблеми, потрібно відповісти на наступні питання: чи достатньо можливостей Python[2] для демонстрації ключових концепцій ООП[1] у повній мірі; розгляду прикладів шаблонів та анти-шаблонів дизайну програмних систем[3]; застосування на практиці принципів SOLID[4] для побудови надійних та масштабовних ПС. У випадку якщо наявні певні обмеження, важливо розуміти, чи є вони критичними і переважають користь від послідовного вивчення мови протягом декількох курсів, та врахувати в відповідних складових курсу.

Основна частина

Найперше, варто відмітити ключову особливість Python[2], що є одночасно і перевагою і недоліком — це те, що вона підтримує декілька парадигм. Серед них не лише об'єктно-орієнтовна, а й, наприклад, функціональна. Тому мова може бути використана також у дисциплінах, присвячених функціональному програмуванню. Також можливим є розгляд шаблонів рішень, що поєднують обидва підходи, використовуючи найкраще, що є в них. Але саме тому не може вважатися типовим прикладом чистої об'єктно-орієнтованої мови. Разом з тим варто також відмітити, що Python є мовою з динамічною типізацією, на відміну від класичного сімейства об'єктно-орієнтовних мов зі статичною типізацією C++, Java, C#, через призму яких часто розглядаються основні концепції ООП. Це зумовлює особливість реалізації цих концепцій в Python. Серед основних концепцій ООП відмітимо і розглянемо наступні: інкапсуляція, наслідування, поліморфізм, абстрагування[1].

Інкапсуляція полягає в тому, що дані та методи обробки цих даних можуть бути поєднанні в єдиній сутності (класі), за потреби особливості реалізації можуть бути приховані від зовнішнього коду[1]. Звичними механізмами реалізації у мовах програмування є можливість задавати у описі класів як складову, що відповідає даним, або стану, так і складову, що відповідає за поведінку, обробку даних. Таким чином маємо поля і методи класів, що мають місце і в Python. Іншим аспектом інкапсуляції, яку іноді називають строгою, є наявність механізмів регулювання строгого доступу зовнішнього коду до елементів класу. Прикладом є модифікатори доступу для полів та методів. Особливістю Python є відсутність таких модифікаторів, лише домовленість, що назви таких компонент будуть починатися на подвійне підкреслення і вони будуть особливим чином перетворюватися та трактуватися компілятором. Це не гарантує заборону доступу, та не приховує повністю.

Наслідування полягає в тому, що нові класи можуть бути створені на основі вже існуючих, успадковуючи зазначені компоненти батьківського класу чи класів[1]. Це дозволяє ефективно використовувати існуючий код. Також у теорії вирізняють різні типи успадкування, зокрема множинне. Варто відмітити, що різні об'єктно-орієнтовані мови підтримують різні типи наслідування, мотивуючи це складністю деяких типів і ризиком внесення помилок. В свою чергу Python підтримує наслідування, причому також і множинне. Наявність множинного наслідування дозволяє розглянути основні ризики його використання, та механізми запобігання помилкам в такому разі.

Поліморфізм полягає в тому, що об'єкти з однаковим інтерфейсом, або специфікацією, можуть мати різну реалізацію[1]. Це дозволяє зовнішньому коду працювати однаково з різними об'єктами, а інформацію про свою поведінку визначає власне об'єкт. Варто зазначити, що механізми реалізації поліморфізму суттєво відрізняються в мовах з динамічною типізацією та мовах зі статичною типізацією. Тому доцільним є розгляд в дисципліні «Об'єктно-орієнтоване програмування» також прикладів з мов зі статичною типізацією, як Java, або C#, адже в Python механізм віртуальних, перевантажених методів не є релевантним. Поліморфізм в Python природним чином впливає з так званої «качиної» типізації, що є прикладом динамічної типізації. Адже в цьому випадку коректність типу змінної визначається її інтерфейсами, тобто методами та властивостями, що використовуються зовнішнім кодом. В свою чергу в мовах зі статичною типізацією об'єкти з однаковим інтерфейсом з точки зору поліморфізму розуміються як ті, що реалізують один і той же інтерфейс, або можуть вважатися екземплярами одного й того ж класу (наприклад, нащадками одного й того ж батьківського класу).

Абстракція полягає в тому, що приховуються надлишкові деталі реалізації, залишаючи тільки необхідні компоненти[1]. Зазвичай забезпечується механізмом абстрактних класів та інтерфейсів. Зважаючи на динамічну природу Python та підтримку функціональної парадигми, спеціальний модуль Abstract Base Classes дозволяє описати відповідні механізми в термінах Python і гарантувати дотримання контрактів у вигляді інтерфейсів, не змінюючи загального підходу до типізації.

Крім основних концепцій в ООП важливу роль також відіграють приклади шаблонів та анти-шаблонів проектування, зокрема набір загальноприйнятих GoF шаблонів[3]. Саме вони свого часу слугували доказом переваги ООП над попередніми парадигмами, такими як структурне програмування. Тому з точки зору використання мови Python у межах

дисципліни «Об'єктно-орієнтоване програмування», важливим є питання, наскільки релевантними є такі шаблони, чи деякі з них взагалі можна реалізувати за допомогою даної мови. Розглянемо шаблони GoF, найперша версія була запропонована з прикладами мовами C++ та Smalltalk й враховували можливості та обмеження відповідних мов[3]. У свою чергу мова Python має вбудовані декоратори, що робить шаблон декоратор зайвим, також підтримка функціональної парадигми програмування та динамічна типізація дозволяє значно спростити такі шаблони як стратегія, адаптер. Це не применшує важливості даних шаблонів для інших мов, проте дозволяє продемонструвати, що проблема, яку вони вирішували була актуальною і тому засоби її вирішення стали частиною базових можливостей нових мов програмування.

Більш високорівневим питанням присвячені принципи SOLID[4], що були запропоновані як рекомендації, дотримання як дозволить отримати в результаті розробки програмну систему, що буде пристосованою для впровадження нових змін, що є неминучим. Доцільним є дослідження того, наскільки дані принципи застосовні до розробки з використанням мови Python та чи є вони універсальними для будь-якої мови, що підтримує об'єктно-орієнтовну парадигму.

Принцип єдиної відповідальності є одним з найбільш загальним і в ньому йдеться про те, що клас повинен мати єдину причину для внесення змін, єдину відповідальність, та уособлювати собою структурну одиницю системи відповідальну за єдину функцію[4]. Враховуючи дане формулювання, принцип є релевантним для будь-якої мови, що підтримує класи. Насправді, він може бути узагальненим на довільну парадигму програмування, що має виокремлені структурні одиниці, які потенційно не обмежені однією функцією. Тому можна сказати, що принцип єдиної відповідальності є актуальним і для мови Python, не обов'язково виконується за замовчуванням.

Принцип відкритості/закритості полягає в тому, що елементи програм мають бути відкритими для розширення, та закритими для змін[4]. У термінах мови програмування це означає, що повинен існувати механізм за допомогою якого функціональні можливості класів, методів можуть бути розширені без зміни їх коду. Тоді даний принцип є релевантним для мови. Прикладами таких механізмів в об'єктно-орієнтованій мові є можливості за допомогою яких реалізовано абстрагування, наслідування, поліморфізм. Функції існуючих класів можуть бути розширені в класах нащадках, а поліморфізм гарантує те, що для зовнішнього коду не буде різниці між існуючим класом

та тим, що його розширює. Зважаючи, що все вищезазначене релевантно для мови Python, принцип відкритості/закритості теж таким є.

Принцип підстановки Лісков полягає у тому, що класи можуть бути замінені підкласами не порушуючи роботу програми та не призводячи до помилок[4]. Так як мова Python має динамічну типізацію, то даний принцип може бути застосовним і має сенс і для програм написаних нею. Варто відмітити, що з точки зору синтаксису та можливостей об'єктно-орієнтовних мов, однією зі складових наслідування є можливість використовувати екземпляри класів нащадків всюди де очікується батьківський клас. Отже, принцип підстановки Лісков застосовний до більшості мов в яких реалізовано наслідування, або динамічну типізацію. Загалом даний принцип є спробою врегулювати змістовний аспект поняття наслідування і узгодити очікування.

Принцип розділення інтерфейсу полягає у наданні переваги декільком простішим та більш специфічним інтерфейсам перед одним загальним[4]. Наявність модуля, що дозволяє задавати інтерфейси та абстрактні класи, дає змогу говорити про даний принцип з точки зору мови Python. Такий модуль є ключовим з точки зору абстрагування.

Принцип інверсії залежностей полягає в тому, що абстракції не повинні залежати від деталей, або сутності верхніх рівнів від сутностей нижніх рівнів деталізації[4]. Як і у випадку з попереднім принципом саме наявність засобів для абстрагування та наслідування разом з відсутністю відповідних обмежень робить його застосовним до певної мови програмування.

Таким чином всі принципи SOLID застосовні до програм мовою Python і тому вона може бути використана для демонстрації впливу їх порушення на практиці. Варто зазначити, що підтримка функціонального програмування має бути врахована в цьому випадку, адже самі принципи були сформульовані як рекомендації до програм реалізованих в чистій об'єктно-орієнтованій парадигмі. Підсумовуючи все можна відмітити, що враховуючи певні обмеження, мова Python є релевантною з точки зору практичного аспекту дисципліни «Об'єктно-орієнтоване програмування», також зважаючи на її використання у суміжних дисциплінах.

Висновки

Досліджено реалізацію основних принципів об'єктно-орієнтованого програмування в мові Python. Серед переваг визначено наявність складної системи наслідування, що дозволяє продемонструвати потенційні ризики множинного успадкування. Динамічна типізація в свою чергу не дозволяє

продемонструвати складніші механізми забезпечення поліморфізму. Також як обмеження відмічено відсутність явних модифікаторів доступу.

Описано яким чином принципи SOLID застосовні до програм написаних мовою Python та чому приклади їх порушення все ще залишаються релевантними для даної мови.

Описано особливості висвітлення шаблонів проектування в рамках курсу та ілюстровано прикладами потребу в адаптації класичних шаблонів, що не применшує проблеми, які вони вирішують, а лише демонструє їх важливість та можливості адаптації мови програмування. Окрему увагу приділено підтримці мовою Python інших парадигм програмування.

Список використаних джерел

1. Musch O. Object-Oriented Programming and Design Principles. Design Patterns with Java. Wiesbaden : Springer Vieweg, 2023. С. 13–21. URL: https://doi.org/10.1007/978-3-658-39829-3_2.
2. *Python documentation*. URL: <https://www.python.org/doc/> (дата звернення: 21.08.2026).
3. Head First. Патерни проектування / Е. Фрімен та ін. Харків : Фабула, 2020. 672 с. ISBN 978-617-09-6159-4.
4. Мартін Р. Чиста архітектура. Харків : Фабула, 2019. 368 с. ISBN 978-617-09-5286-8.

Панченко Тарас Володимирович, к.ф.-м.н, доцент

ПОБУДОВА КУРСІВ СИСТЕМНОГО ПРОГРАМУВАННЯ ТА ОПЕРАЦІЙНИХ СИСТЕМ У ПІДГОТОВЦІ ІТ-ФАХІВЦІВ: ВІД НИЗЬКОРІВНЕВИХ МЕХАНІЗМІВ ДО ДОСЛІДНИЦЬКИХ ПРАКТИК

У роботі узагальнено досвід побудови та викладання взаємопов'язаного комплексу дисциплін із системного програмування та операційних систем на бакалаврському й магістерському рівнях підготовки фахівців з комп'ютерних наук. Окрему увагу приділено методичним особливостям організації навчання за умов, коли обов'язкова дисципліна «Системне програмування» вивчається раніше, ніж вибіркова дисципліна «Unix-подібні операційні системи». Описано структуру курсу системного програмування, що поєднує три взаємопов'язані компоненти: програмування в командному середовищі UNIX, основи автоматизованої побудови трансляторів та низькорівневе програмування й оптимізацію.

Ключові слова: системне програмування, операційні системи, Linux, UNIX, оптимізація програм, профілювання, енергоефективність, паралелізм, дослідницьке навчання, Flame Graph, eBPF, perf.

The paper summarizes the experience of designing and teaching an interconnected set of courses in system programming and operating systems at the bachelor's and master's levels of computer science education. Particular attention is paid to the methodological aspects of organizing the learning process in a curriculum where the compulsory course "System Programming" is taught before the elective course "Unix-like Operating Systems". The structure of the System Programming course is described as comprising three interrelated components: programming in the UNIX command-line environment, fundamentals of automated compiler construction, and low-level programming and optimization. Particular emphasis is placed on the practical analysis of program performance, the use of profiling, Flame Graphs, perf, and eBPF, machine-code analysis, as well as the measurement of computational energy consumption using hardware counters available in modern processors. The paper also examines the design of a bachelor-level operating systems course centered on fundamental resource-management mechanisms and a master-level course, «Modern Operating Systems», characterized by a strong research component. The proposed approach is presented as a description of practical experience in organizing the content of these related courses.

Keywords: system programming, operating systems, Linux, UNIX, program optimization, profiling, energy efficiency, parallelism, research-based learning, Flame Graph, eBPF, perf.

Вступ

Системне програмування та операційні системи належать до фундаментальних напрямів підготовки фахівців з комп'ютерних наук, оскільки забезпечують розуміння взаємодії прикладного програмного забезпечення, системного програмного забезпечення та апаратної платформи. Вивчення процесів і потоків, механізмів синхронізації, керування пам'яттю, введення-виведення, системних викликів, трансляції програм і виконання машинних команд формує той рівень системного мислення, який необхідний не лише розробникам операційних систем, а й фахівцям, що працюють з високопродуктивними, серверними, розподіленими та ресурсоемними програмними системами.

Актуальні міжнародні рекомендації з формування програм Computer Science підтверджують фундаментальний статус відповідної предметної області. У рекомендаціях Computer Science Curricula 2023, розроблених спільною робочою групою ACM, IEEE Computer Society та AAAI, операційні системи виокремлено як самостійну область знань (Operating Systems Knowledge Area). Частина її змісту віднесена до CS Core, тобто мінімального ядра знань, яке рекомендації розглядають як необхідне для підготовки випускника Computer Science. До нього належать, зокрема, роль і принципи роботи операційної системи, процеси, планування, конкурентність, керування пам'яттю, захист, введення-виведення та файлові системи [1]. Водночас CS2023 не встановлює вимоги реалізовувати цей зміст виключно у вигляді однієї окремої дисципліни: конкретне komponування змісту залишається завданням освітньої програми.

Подібна варіативність спостерігається і в освітніх програмах провідних університетів. У поточній структурі бакалаврської програми Computer Science Стенфордського університету дисципліна CS111 Operating Systems Principles представлена в ядрі підготовки. Її зміст охоплює конкурентність і синхронізацію, планування, процеси, віртуальну пам'ять, введення-виведення, файлові системи та механізми захисту [2; 3]. При цьому більш поглиблений CS112 Operating Systems Kernel Implementation Project належить уже до системного напрямку підготовки. Така структура демонструє розподіл між базовим розумінням принципів ОС та подальшим поглибленим системним проєктуванням.

Іншу модель реалізовано в програмі Computer Science and Engineering (Course 6-3) Массачусетського технологічного інституту. У межах вимог до спеціальності студент має опанувати один із системних компонентів: 6.1800 Computer Systems Engineering, 6.1810 Operating System Engineering або 6.5831

Database Systems. Курс 6.1810 зосереджений безпосередньо на проєктуванні й реалізації операційних систем і охоплює віртуальну пам'ять, файлові системи, потоки, перемикання контексту, ядро, переривання, системні виклики, IPC та взаємодію програмного й апаратного забезпечення [4; 5]. Отже, конкретна форма інтеграції операційних систем у навчальний план може відрізнятися, однак системна складова посідає суттєве місце в підготовці фахівця з комп'ютерних наук.

Водночас наведені міжнародні приклади не слід інтерпретувати як нормативну вимогу до конкретної української освітньої програми. Стандарт вищої освіти України зі спеціальності 122 «Комп'ютерні науки» встановлює компетентності та результати навчання, але не визначає обов'язкових назв і послідовності конкретних освітніх компонентів. Тому порівняння зі Stanford, MIT та рекомендаціями ACM/IEEE-CS/AAAI у цій роботі використовується винятково як контекст для аналізу сучасних підходів до системної підготовки студентів, а не як аргумент щодо правильності чи неправильності конкретного навчального плану [22].

У межах освітньої програми, досвід викладання якої розглядається у цій роботі, дисципліна «Системне програмування» є обов'язковою та викладається у першому семестрі третього курсу, тоді як «Unix-подібні операційні системи» пропонуються пізніше як вибіркова дисципліна. Це визначає важливу методичну умову: при побудові курсу системного програмування не можна виходити з припущення, що студент уже пройшов окремий систематичний курс операційних систем.

При цьому вибірковий курс «Unix-подібні операційні системи» не розглядається як формальна передумова для вивчення обов'язкового курсу «Системне програмування». Навпаки, завданням викладача є така організація останнього, за якої необхідні для нього поняття вводяться безпосередньо в контексті відповідних практичних задач, а подальший курс операційних систем дає можливість систематизувати й поглибити розуміння внутрішніх механізмів ОС.

Описана в подальшому структура дисципліни «Системне програмування» відповідає її актуальній робочій програмі [6]. Саме ця редакція програми та досвід її практичної реалізації є предметом аналізу в статті.

Стаття має характер узагальнення та аналізу педагогічного досвіду, а не контрольованого педагогічного експерименту. Її метою не є доведення статистичної переваги описаної структури курсу над іншими моделями. Для такого висновку були б необхідні щонайменше порівнювані навчальні групи,

попередньо визначені критерії результативності, вимірювання результатів навчання та відповідний статистичний аналіз. Предметом цієї роботи є інше питання – яким чином можна змістово пов'язати системне програмування, архітектуру обчислювальних систем та операційні системи і водночас наблизити практичні завдання до актуальних інженерних задач.

Побудова курсу «Системне програмування»

Курс «Системне програмування» побудовано навколо трьох взаємопов'язаних тематичних блоків: програмування в UNIX-середовищі та командному рядку; принципів побудови трансляторів; низькорівневого програмування, аналізу машинного коду та оптимізації.

Перший блок продовжує традицію класичного UNIX-програмування, сформовану, зокрема, у роботах Б. Кернігана та Р. Пайка [7]. Його метою є не стільки вивчення набору команд операційної системи, скільки формування розуміння UNIX як програмного середовища, в якому невеликі інструменти можуть об'єднуватися в складніші обчислювальні ланцюги за допомогою стандартних потоків, каналів, перенаправлення введення-виведення та сценаріїв командної оболонки.

У цьому контексті shell розглядається як середовище програмування. Студенти працюють зі стандартними потоками, файловими дескрипторами, змінними середовища, процесами та композицією програм. Одночасно вводиться мінімальний набір понять операційної системи, необхідний для розуміння поведінки відповідних механізмів. Таким чином забезпечується автономність дисципліни від попереднього проходження окремого курсу ОС.

Другий блок присвячено принципам побудови трансляторів. Розглядаються лексичний та синтаксичний аналіз, контекстно-вільні граматики, а також принципово різні способи синтаксичного розбору. LL(1)-аналіз використовується як приклад низхідної (top-down) побудови дерева розбору, тоді як LALR – як представник висхідних (bottom-up) методів [8].

Важливою методичною задачею є показати студентам не лише формальні правила роботи аналізатора, а весь базовий ланцюг трансляції: лексичний аналіз, синтаксичний розбір, формування семантики, проміжне представлення, оптимізацію та генерацію коду. Практична реалізація базується на класичному інструментарії lex/yacc та його сучасних аналогах [9].

Як компактний приклад використовується традиційний для навчання компіляторів «калькулятор». Попри обмеженість предметної області, він

дозволяє пройти основні етапи побудови транслятора та показати зв'язок між граматикою мови, семантикою виразів і результатом трансляції.

Обсяг суто формальної теорії синтаксичного аналізу протягом розвитку курсу поступово скорочується на користь тем, пов'язаних із семантикою, генерацією коду, оптимізацією та аналізом кінцевого машинного представлення. Така зміна акценту не означає відмови від теоретичних засад побудови компіляторів, а відображає необхідність розглянути весь цикл перетворення програми в умовах обмеженого семестрового часу.

Третій тематичний блок охоплює асемблерне програмування, архітектуру процесора та низькорівневу оптимізацію. Його завдання полягає у формуванні зв'язку між програмою мовою високого рівня, діями компілятора та фактичним виконанням команд апаратною платформою.

Розглядаються регістри, система команд, конвеєрне виконання, залежності між командами, кеш-пам'ять, прогнозування переходів, позачергове виконання та інші механізми сучасних процесорів. Асемблерний код при цьому використовується передусім як засіб аналізу, а не як основна мова розробки.

Практичним інструментом є Compiler Explorer – середовище, яке дає змогу зіставити вихідний код із машинним представленням, дослідити роботу різних компіляторів і рівнів оптимізації та безпосередньо побачити, які конструкції мови високого рівня породжують певні послідовності машинних команд [10].

Такий підхід дає змогу пов'язати теми компіляторів та архітектури процесорів. Студент спостерігає не абстрактну «оптимізацію компілятора», а конкретні зміни машинного коду і може досліджувати їхній вплив на характеристики виконання програми.

Профілювання, оптимізація та енергетична ефективність програм

Окремим наскрізним компонентом курсу є аналіз продуктивності програм. Його ключовим принципом є вимога переходити до оптимізації лише після вимірювання та локалізації реального вузького місця.

Такий підхід відповідає сучасній практиці системного аналізу продуктивності, систематизованій, зокрема, у роботах Brendan Gregg [11; 12]. Для профілювання використовуються утиліти perf, інструментарій eBPF/BPF, а для візуалізації отриманих стеків викликів – Flame Graph.

Flame Graph дозволяє представити статистику виконання програми у формі, придатній для швидкого виявлення фрагментів, на які припадає значна

частина процесорного часу. При цьому важливим результатом лабораторної роботи є не сама побудова графіка, а його інтерпретація: студент має пояснити, чому певний шлях виконання є витратним і чи можна вплинути на нього алгоритмічно або на рівні реалізації.

Практичне завдання організовується як ітеративний цикл: вимірювання → локалізація вузького місця → формування гіпотези → оптимізація → повторне вимірювання → аналіз результату.

Такий цикл привчає розділяти припущення щодо продуктивності та результати реальних вимірювань. Зокрема, зміна, яка інтуїтивно сприймається як оптимізація, може не давати вимірюваного ефекту або переносити навантаження до іншої частини системи.

Додатковим напрямом лабораторних досліджень є оцінювання енергетичної ефективності програм.

Традиційний аналіз продуктивності переважно відповідає на питання, який обсяг роботи виконується за одиницю часу. Для сучасних серверних і високопродуктивних систем дедалі актуальнішим є також співвідношення виконаної роботи та витраченої енергії.

При цьому фізично доцільно розрізняти потужність та енергію. Потужність вимірюється у ватах, енергія – у джоулях. Тому одним із природних показників ефективності є кількість роботи або операцій на одиницю енергії, наприклад operations/J. Поширений показник performance per watt математично характеризує ту саму взаємозалежність продуктивності й енергетичних витрат.

Сучасні процесорні платформи надають апаратні механізми оцінювання споживаної енергії. У Linux доступ до них може здійснюватися, зокрема, через Power Capping Framework і механізми на кшталт Intel RAPL. Залежно від апаратної платформи доступними можуть бути показники процесорного пакета та окремих енергетичних доменів [13]. Такі вимірювання не слід ототожнювати з точним визначенням споживання всієї системи або системної плати – для цього може бути необхідне зовнішнє вимірювальне обладнання. Проте внутрішні лічильники є придатним засобом для порівняльного експерименту між різними варіантами однієї програми.

Практична актуальність такого підходу простежується і в розвитку інфраструктур високопродуктивних обчислень. У Worldwide LHC Computing Grid показник HEPScore23 використовується для оцінювання обчислювальних ресурсів, зокрема для планування потужностей і оцінювання нового обладнання. Сучасний HEP Benchmark Suite доповнює традиційне вимірювання продуктивності збором системних характеристик, включно зі

споживанням потужності [14]. У 2026 році NEPiX Benchmarking Working Group окремо визначає розвиток засобів вимірювання завантаження, частоти, I/O та енергоспоживання як напрям, необхідний для одночасної оцінки performance і power efficiency обчислювальної інфраструктури [15].

У межах навчальної роботи студент отримує можливість дослідити програму одночасно з двох позицій – часової та енергетичної. Спочатку формується початковий профіль, визначаються найбільш витратні ділянки, фіксується час виконання та доступні енергетичні показники. Після оптимізації експеримент повторюється.

Це дозволяє показати, що мінімальний час виконання та мінімальні енергетичні витрати не завжди є тотожними критеріями. У такий спосіб класична задача оптимізації розширюється до багатокритеріального інженерного аналізу.

У сукупності програмування в UNIX-середовищі, побудова трансляторів, аналіз машинного коду, профілювання та експериментальна оптимізація формують цілісний, хоча через часові обмеження неминуче різний за глибиною, огляд основних рівнів системного програмування.

Курс «Unix-подібні операційні системи»

Якщо курс системного програмування орієнтований передусім на взаємодію програміста з системним середовищем і аналіз виконання програм, то курс операційних систем має інше завдання – сформувати цілісну модель внутрішньої організації ОС. Курс за будовою та наповненням схожий зі згаданими вище Stanford CS111 і MIT 6.1810 [3; 5].

Теоретичною основою курсу є класична структура викладання операційних систем, представлена, зокрема, в Modern Operating Systems Е. Таненбаума та Г. Боса [16]. У сучасному п'ятому виданні розглядаються процеси й потоки, пам'ять, файлові системи, введення-виведення, тупики, віртуалізація, багатопроцесорні системи, безпека та архітектура сучасних ОС.

Водночас метою курсу є не лише відтворення класичної теорії. Для кожного фундаментального механізму розглядається зв'язок між вихідною інженерною проблемою, класичним способом її вирішення та сучасним станом відповідної підсистеми.

Управління процесорним часом аналізується через концепцію time sharing, механізми планування та критерії справедливого розподілу ресурсів. Керування пам'яттю розглядається через віртуальний адресний простір, сторінкову організацію, memory mapping та dynamic address translation. Аналогічно досліджуються введення-виведення, файлові та мережеві підсистеми.

Важливою частиною курсу є питання про причини виникнення конкретних архітектурних рішень. Студент має розуміти не тільки алгоритм, але й проблему, яка до нього привела, наявні компроміси та обмеження.

Показовим прикладом такого підходу є розгляд мікроархітектурних вразливостей Spectre і Meltdown [17; 18]. Прогнозування переходів, спекулятивне та позачергове виконання створювалися для підвищення продуктивності процесорів, однак у певних сценаріях вони порушили традиційні припущення про ізоляцію програмних контекстів. Аналіз таких випадків дозволяє продемонструвати студентам, що рішення в операційних системах, компіляторах, апаратній архітектурі та інформаційній безпеці не існують ізольовано.

Практична частина курсу значною мірою концентрується на конкурентності та паралелізмі. Розглядаються *race conditions*, взаємне виключення, синхронізація потоків, блокування, тупики та нескінченне відкладання.

Для формування відповідних навичок використовуються класичні модельні задачі: *producer-consumer*, *bounded buffer*, *dining philosophers* та інші. Їхня перевага полягає в тому, що порівняно невеликі за обсягом програми відтворюють фундаментальні класи проблем, характерні для значно складніших реальних систем.

Інший напрям практичних завдань передбачає взаємодію з механізмами реальних операційних систем – обробкою подій, системними інтерфейсами, засобами перехоплення й аналізу системної активності, механізмами пам'яті. При цьому практична робота будується так, щоб студент не лише отримав працездатний код, а й пояснив системний механізм, на якому базується отриманий результат.

Таким чином, курс «Unix-подібні операційні системи» не виступає попередньою умовою курсу «Системне програмування», а виконує іншу функцію – систематизує та поглиблює модель внутрішньої будови ОС, частина проявів якої вже була доступна студентові через попередню практику системного програмування. Разом з тим, курс операційних систем фактично формує не вузьку спеціалізацію розробника ядра, а фундамент для більш зрілого розуміння поведінки програмних систем.

Магістерський курс «Сучасні операційні системи»

Інша логіка використовується для дисципліни «Сучасні операційні системи», що викладається магістрам освітньо-наукової програми «Інформатика».

Студенти магістерської програми можуть мати різний попередній освітній досвід, тому початкова частина дисципліни виконує функцію систематизації та вирівнювання базових понять. Водночас повторення бакалаврського курсу не є її основним завданням.

Після базової частини акцент переноситься на поглиблене дослідження сучасних механізмів операційних систем. Однією з форм роботи є підготовка студентами тематичних міні-досліджень із подальшою доповіддю та обговоренням.

Викладач пропонує перелік орієнтовних тем, однак студент може запропонувати власний напрям за умови попереднього погодження. До таких тем належать Read-Copy-Update (RCU), futex, сучасні алгоритми планування процесорного часу, контейнеризація, Docker, Kubernetes та інші системні механізми.

RCU розглядається як приклад механізму синхронізації, оптимізованого для сценаріїв із великою кількістю операцій читання [19]. Futex демонструє інший важливий принцип – перенесення uncontended-випадку синхронізації до user space із залученням ядра насамперед тоді, коли потоку необхідно перейти до очікування [20].

Окремим прикладом розвитку системних механізмів є еволюція планувальника Linux. Традиційний Completely Fair Scheduler дає можливість розглянути поняття virtual runtime та справедливого розподілу процесорного часу, тоді як сучасний перехід до EEVDF дозволяє аналізувати, чому навіть усталені фундаментальні механізми ОС продовжують змінюватися [21].

Методика роботи з такими темами не обмежується реферативною доповіддю. Після представлення матеріалу проводиться спільне обговорення причин появи механізму, його сильних та слабких сторін і можливих альтернатив.

Важливу роль відіграє формат «чому саме так?» та what-if analysis: що зміниться, якщо відмовитися від конкретної властивості алгоритму, перемістити механізм між user space і kernel space, використати інший тип синхронізації або змінити характер навантаження.

Дослідницький характер має й лабораторна частина дисципліни. Один із напрямів пов'язаний із міжпроцесною взаємодією та пам'яттю. Особливу увагу приділено mmap як одному з ключових POSIX-механізмів відображення файлів і shared memory у віртуальний адресний простір процесу.

Другий напрям стосується socket-взаємодії. Студенти порівнюють Internet sockets та UNIX domain sockets, blocking і non-blocking режими, синхронні й асинхронні способи організації роботи.

На магістерському рівні важливо, щоб лабораторна робота не завершувалася демонстрацією працездатної програми. Вона має містити дослідницьке питання, опис експериментальної конфігурації, результати вимірювань або спостережень, їхню інтерпретацію та висновки.

Усі роботи підлягають усному захисту. Захист організовується як коротке Q&A щодо сутності проведеного експерименту та звіту. Студент повинен пояснити вибрану методику, основні результати, обмеження експерименту та обґрунтувати зроблені висновки.

Таким чином, магістерський курс відрізняється від бакалаврського не лише складністю тематики, але й характером навчальної діяльності: від засвоєння й застосування системних механізмів здійснюється перехід до їх критичного аналізу та експериментального дослідження.

Обговорення

Розглянуті три дисципліни утворюють взаємопов'язану систему, але виконують різні освітні функції.

«Системне програмування» формує зв'язок між програмою, системним середовищем, компілятором та апаратною платформою. «Unix-подібні операційні системи» систематизують знання про внутрішні механізми ОС і дозволяють пояснити поведінку системи через її архітектуру. «Сучасні операційні системи» на магістерському рівні переносить акцент на самостійний аналіз і дослідження сучасних механізмів.

Відсутність попереднього окремого курсу ОС при вивченні системного програмування в такій моделі не інтерпретується як формальна суперечність навчального плану. Вона виступає вихідною умовою побудови дисципліни: системне програмування має бути самодостатнім щодо необхідних базових понять, а подальший курс ОС – поглиблювати й систематизувати їх.

Порівняння зі Stanford, MIT та рекомендаціями CS2023 також показує, що універсальної моделі «одна конкретна дисципліна – одне фіксоване місце у навчальному плані» не існує. Stanford виділяє базові принципи ОС у Core та має окремі поглиблені системні курси; MIT дозволяє реалізувати обов'язкову системну складову різними курсами; CS2023 визначає обов'язковість певного ядра знань стосовно операційних систем, залишаючи конкретне curricular packaging освітній програмі [1–5]. Саме тому предметом аналізу доцільно

робити не назву або статус окремої дисципліни самі по собі, а те, яким чином у сукупності освітніх компонентів формуються необхідні системні компетентності.

Описаний досвід також показує можливість пов'язати класичні теми системного програмування з актуальними задачами аналізу програм. Застосування Compiler Explorer, perf, eBPF, Flame Graph та апаратних засобів вимірювання енерговитрат створює для студента єдиний експериментальний контекст: від вихідного коду – до машинних команд, профілю виконання, часових характеристик та енергетичної ефективності.

Однак на підставі наведеного досвіду не можна робити висновок про перевагу цієї методики над альтернативними моделями викладання. У межах роботи не проводилося порівняльного педагогічного експерименту з контрольними групами та стандартизованими показниками результатів навчання. Тому отримані положення слід розглядати як опис реалізованого підходу та формулювання гіпотез для подальшого емпіричного дослідження.

Перспективним продовженням роботи є кількісна перевірка окремих складових запропонованої моделі: порівняння результатів засвоєння системних концепцій за різних послідовностей їх введення; оцінка впливу практик профілювання на здатність студентів локалізувати проблеми продуктивності; дослідження того, наскільки виконання експериментів з енергоспоживанням змінює підхід студентів до оптимізації програм.

Висновки

Досвід викладання системного програмування та операційних систем свідчить про необхідність розглядати відповідні дисципліни не як ізольований набір курсів, а як послідовну систему формування системного мислення майбутнього ІТ-фахівця.

Запропонована структура курсу «Системне програмування» охоплює три взаємопов'язані рівні: програмування в UNIX-середовищі, автоматизовану побудову трансляторів та низькорівневий аналіз і оптимізацію програм. Інтеграція профілювання, Flame Graph, perf/eBPF, Compiler Explorer та вбудованих апаратних засобів вимірювання енерговитрат переводить навчальну роботу від розгляду абстрактних прикладів до задач, характерних для реального системного програмування.

Особливо перспективним видається доповнення класичного критерію продуктивності критерієм енергетичної ефективності. Формування у студентів звички оцінювати не лише час виконання, а й витрати енергії дозволяє пов'язати низькорівневу оптимізацію з актуальною проблематикою sustainable та green computing.

Бакалаврський курс операційних систем, своєю чергою, має формувати фундаментальне розуміння внутрішньої логіки системи: не лише того, як працюють розподіл часу (time sharing), віртуальна пам'ять, введення-виведення, синхронізація або файлові системи, але й причин, через які відповідні архітектурні рішення виникали та змінювалися. Курс, схожий за будовою і змістом до аналогічних від світових лідерів, Stanford та MIT, будучи в категорії обов'язкових, допоміг би формуванню комплексного погляду майбутніх розробників, розвитку системного мислення та навичок аналізу причинно-наслідкового зв'язку у практиці IT.

На магістерському рівні зміщується сам характер навчальної діяльності. Дисципліна «Сучасні операційні системи» будується як поєднання систематизації фундаментальних знань із дослідницькими семінарами, самостійним опрацюванням сучасних механізмів та експериментальними лабораторними роботами. Обов'язковий усний захист результатів доповнює програмну реалізацію перевіркою здатності студента інтерпретувати та аргументувати отримані результати.

У сукупності такий підхід дозволяє сформувати хоча й широкий і неминуче обмежений за глибиною в межах окремих тем, проте цілісний погляд на системне програмування. Студент проходить шлях від командної оболонки й системних інтерфейсів через компіляцію та машинний код до мікроархітектури процесора, профілювання, енергоспоживання та внутрішніх механізмів операційної системи. Саме цей зв'язок між фундаментальними принципами, низькорівневою реалізацією та експериментальним аналізом, на думку автора, є одним із ключових результатів сучасної підготовки фахівців напряму комп'ютерних наук та інформаційних технологій із системним мисленням та комплексним поглядом на задачі.

Водночас наведені узагальнення характеризують конкретний досвід організації навчання і не є доказом переваги описаної методики над іншими підходами. Для оцінювання її педагогічної результативності необхідне окреме порівняльне дослідження з формалізованими критеріями, контрольованими умовами та кількісним аналізом результатів навчання.

Список використаних джерел

1. Kumar A. N., Raj R. K., Aly S. G., Anderson M. D., Becker B. A., Blumenthal R. L., Eaton E., Epstein S. L., Goldweber M., Jalote P., Lea D., Oudshoorn M., Pias M., Reiser S., Servin C., Simha R., Winters T., Xiang Q. Computer Science Curricula 2023. ACM Press, IEEE Computer Society Press, AAAI Press, 2024. ISBN 979-8-4007-1033-9. DOI: 10.1145/3664191.
2. Stanford University. Department of Computer Science. BS Degree Requirements. URL: <https://www.cs.stanford.edu/bs-degree-requirements>.
3. Stanford University. CS111: Operating Systems Principles. URL: <https://explorecourses.stanford.edu/>.

4. Massachusetts Institute of Technology. Bachelor of Science in Computer Science and Engineering, Course 6-3. MIT Course Catalog. URL: <https://catalog.mit.edu/degree-charts/computer-science-engineering-course-6-3/>.
5. Massachusetts Institute of Technology. 6.1810 Operating System Engineering. MIT Course Catalog. URL: <https://catalog.mit.edu/subjects/6/>.
6. Робоча програма навчальної дисципліни «Системне програмування». URL: https://tp.csc.knu.ua/wp-content/uploads/2026/09/БІНФОП23ПЛ23_OK_19_СП-Панченко.pdf
7. Kernighan B. W., Pike R. The UNIX Programming Environment. Englewood Cliffs, NJ: Prentice-Hall, 1984.
8. Aho A. V., Lam M. S., Sethi R., Ullman J. D. Compilers: Principles, Techniques, and Tools. 2nd ed. Boston: Addison-Wesley, 2006.
9. Levine J., Mason T., Brown D. lex & yacc. 2nd ed. Sebastopol, CA: O'Reilly Media, 1992.
10. Compiler Explorer. Interactive compiler exploration environment. URL: <https://compiler-explorer.com/>.
11. Gregg B. Systems Performance: Enterprise and the Cloud. 2nd ed. Pearson, 2020.
12. Gregg B. BPF Performance Tools: Linux System and Application Observability. Boston: Addison-Wesley Professional, 2019.
13. The Linux Kernel Documentation. Power Capping Framework. URL: <https://docs.kernel.org/power/powercap/powercap.html>.
14. Menéndez Borge G. HEP Benchmark Suite: The Centralized Future of WLCG Benchmarking. EPJ Web of Conferences. 2024. Vol. 295. 07023. DOI: 10.1051/epjconf/202429507023.
15. Giordano D., Michelotto M. HEPiX Benchmarking Working Group Report. HEPiX Spring 2026 Workshop. Lisbon, 22 April 2026. CERN Indico.
16. Tanenbaum A. S., Bos H. Modern Operating Systems. 5th ed. Pearson, 2022.
17. Kocher P., Horn J., Fogh A., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Y. Spectre Attacks: Exploiting Speculative Execution. 2019 IEEE Symposium on Security and Privacy. 2019. P. 1–19. DOI: 10.1109/SP.2019.00002.
18. Lipp M., Schwarz M., Gruss D. et al. Meltdown: Reading Kernel Memory from User Space. 27th USENIX Security Symposium. 2018. P. 973–990.
19. The Linux Kernel Documentation. RCU Concepts and Documentation. URL: <https://docs.kernel.org/RCU/>.
20. Linux man-pages project. futex(2) – Fast User-Space Locking. URL: <https://man7.org/linux/man-pages/man2/futex.2.html>.
21. The Linux Kernel Documentation. EEVDF Scheduler. URL: <https://docs.kernel.org/scheduler/sched-eevdf.html>.
22. Стандарт вищої освіти України. Перший (бакалаврський) рівень. Спеціальність 122 «Комп'ютерні науки». Затверджено наказом Міністерства освіти і науки України від 10.07.2019 № 962.

Русіна Наталія Геннадіївна, к.пед.н, доцент

ПРАКТИКО-ОРІЄНТОВАНЕ НАВЧАННЯ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ ОСНОВАМ ПРОГРАМУВАННЯ

Робота присвячена основам практико-орієнтованого навчання програмуванню здобувачів вищої освіти ОПП «Прикладна (комп'ютерна) лінгвістика та англійська мова». Обґрунтовано доцільність використання базового інструменту мови програмування Python для опанування програмування та формування загальних і фахових компетентностей майбутніх прикладних лінгвістів. Визначено основні напрями професійного застосування програмування у прикладній лінгвістиці, наприклад, автоматизоване опрацювання текстів, аналіз і структурування мовних даних, корпусну лінгвістику та реалізацію алгоритмів комп'ютерного лінгвістичного аналізу. Представлено та систематизовано приклади відповідності програмних конструкцій Python прикладним лінгвістичним завданням і визначено їхній зв'язок із фаховими компетентностями та програмними результатами навчання, передбаченими освітньою програмою. Представлено приклади практичних кейсів. Обґрунтовано педагогічні переваги практико-орієнтованого підходу із підвищенням мотивації до вивчення програмування, формування професійних компетентностей та усвідомлення практичної цінності програмування у професійній діяльності прикладного лінгвіста.

Ключові слова: практико-орієнтоване навчання, Python, прикладна лінгвістика, основи програмування, мовні дані, комп'ютерна лінгвістика, програмні компетентності.

The paper focuses on the foundations of practice-oriented programming education for higher education students enrolled in the educational programme «Applied (Computational) Linguistics and English Language.» The relevance of using Python as a basic programming language is substantiated as a tool for mastering programming fundamentals and developing the general and professional competencies of future applied linguists. The main areas of professional application of programming in applied linguistics are identified, including automated text processing, analysis and structuring of linguistic data, corpus linguistics, and implementation of algorithms for computational linguistic analysis. A practice-oriented approach to organizing the students' learning process is proposed, whereby each Python topic is accompanied by practical tasks whose content is directly related to students' future professional activities. Examples

demonstrating the correspondence between Python programming constructs and applied linguistic tasks are presented and systematized, and their relationship with the professional competencies and programme learning outcomes specified in the educational programme is established. Practical cases illustrating the application of programming to professional linguistic tasks are also presented. The pedagogical advantages of the practice-oriented approach are substantiated, including increased motivation to learn programming, development of professional competencies, and a better understanding of the practical value of programming in the professional activities of an applied linguist.

Keywords: practice-oriented learning, Python, applied linguistics, programming fundamentals, linguistic data, computational linguistics, programming competencies.

Вступ

Важливим завданням професійної підготовки здобувачів вищої освіти є не лише використання сучасних цифрових платформ і програмних засобів для навчання, а їх змістовна інтеграція в освітній процес [1]. У контексті вивчення програмування передбачено поєднання традиційних педагогічних методів із практико-орієнтованими цифровими інструментами, які позитивно впливають на формування алгоритмічного мислення, розвиток навичок роботи з даними та розв'язання прикладних лінгвістичних завдань засобами мови Python.

Цифровізація освітнього середовища є важливим чинником модернізації професійної підготовки здобувачів вищої освіти ОПІ “Прикладна (комп’ютерна) лінгвістика та англійська мова”, оскільки створює нові можливості для підвищення якості, доступності та практичної спрямованості навчання. Особливе значення у викладанні основ програмування є те, що цифрові технології забезпечують інтерактивність, гнучкість і наближеність навчальних завдань до реальних професійних потреб майбутніх фахівців. Водночас упровадження цифрових рішень має здійснюватися послідовно, з урахуванням специфіки гуманітарної підготовки здобувачів, особливостей освітніх програм та академічних традицій закладів вищої освіти.

Традиційне навчання програмуванню для гуманітарних спеціальностей нерідко супроводжується труднощами: абстрактністю змісту, низькою мотивацією здобувачів, відсутністю зв'язку між навчальними завданнями та майбутньою професією. Унаслідок цього програмування

сприймається як другорядна дисципліна, що ускладнює формування необхідних професійних компетентностей.

Одним із шляхів розв'язання вищевказаної проблеми є впровадження практико-орієнтованого навчального підходу, у якому теоретичний матеріал пов'язується з реальними професійними завданнями здобувачів освіти.

Професійно спрямоване навчання програмування майбутніх прикладних лінгвістів доцільно будувати на поєднанні програмних і лінгвістичних завдань. Практична спрямованість та міждисциплінарна інтеграція узгоджуються із сучасними засадами STEM-освіти [2]. У працях [3; 4] розглянуто можливості використання Python у підготовці здобувачів-лінгвістів.

У сучасних навчальних матеріалах Python може використовуватися для виконання широкого спектра завдань, пов'язаних із опрацюванням мовних і структурованих даних. У контексті прикладної лінгвістики його застосування охоплює, зокрема, роботу з текстами, корпусами та іншими типами мовних даних [4-6]. Для реалізації таких завдань використовуються спеціалізовані програмні засоби: бібліотека NLTK призначена для виконання завдань обробки природної мови [7], а Stanza забезпечує програмний інструментарій для автоматизованого лінгвістичного аналізу текстів [8]. Водночас модуль re дає змогу працювати з регулярними виразами, бібліотека pandas — опрацьовувати та аналізувати структуровані й табличні дані, а matplotlib — візуалізувати результати їх опрацювання [4; 9]. Таке поєднання інструментів дає змогу формувати практичні завдання, у яких базові програмні конструкції застосовуються до професійно орієнтованих мовних даних.

Одним із ключових питань під час проєктування змісту дисципліни «Основи програмування» для здобувачів ОПП «Прикладна (комп'ютерна) лінгвістика та англійська мова» є вибір мови програмування. Ураховуючи зміст майбутньої професійної діяльності здобувачів, для навчання основ програмування доцільно використовувати Python, оскільки його синтаксис дає змогу зосередити увагу на базових алгоритмічних конструкціях, а екосистема бібліотек забезпечує можливість переходу від навчальних завдань до практичного опрацювання мовних і структурованих даних.

Насамперед мова програмування загального призначення Python характеризується простим і зрозумілим синтаксисом, що суттєво знижує поріг входження для студентів гуманітарного профілю. На відміну від мов програмування C++ або Java, де значна увага приділяється технічним аспектам роботи з пам'яттю, типізацією чи складною структурою програм,

Python надає можливість зосередитися на алгоритмічному мисленні та розв'язанні прикладних задач. Саме тому Python розглядається як доступний інструмент для початківців, які не мають попереднього досвіду програмування [10].

Додатковою перевагою є те, що мова Python широко застосовується у сфері комп'ютерної лінгвістики, аналізі даних та в роботі зі штучним інтелектом. У книзі Майкла Хаммонда «Python for Linguists» зазначається, що ця мова спеціально адаптована для роботи з мовними даними, а більшість прикладів програмування можуть безпосередньо використовуватися для проведення лінгвістичних досліджень. Автор підкреслює, що Python забезпечує ефективну роботу з текстами, пошуком інформації, кодуванням символів та інтернет-даними, які є важливими складниками діяльності сучасного прикладного лінгвіста [4].

Важливим аргументом на користь використання Python є наявність бібліотек для автоматизованої обробки природної мови. Однією з найвідоміших є бібліотека NLTK, яка надає інструменти для токенізації, морфологічного аналізу, класифікації текстів, синтаксичного розбору та роботи з мовними корпусами. Розробники NLTK визначають її як платформу для створення програм, що працюють із людською мовою, а також як засіб навчання комп'ютерної лінгвістики та програмування одночасно [7].

Важливим є міждисциплінарний характер мови програмування Python. Здобуті навички можуть використовуватися не лише в комп'ютерній лінгвістиці, а й у суміжних напрямках: аналізі даних, машинному навчанні, автоматизації роботи з документами, веброзробці та створенні цифрових мовних ресурсів. Таким чином, опанування мови програмування Python сприяє підвищенню конкурентоспроможності випускників на сучасному ринку праці.

Отже, вибір Python як базової мови програмування для здобувачів вищої освіти ОПП «Прикладна (комп'ютерна) лінгвістика та англійська мова» є педагогічно та професійно обґрунтованим. Простота синтаксису, орієнтація на роботу з текстовими даними, широкий спектр бібліотек для NLP, інтеграція з технологіями штучного інтелекту та можливість використання у прикладних лінгвістичних дослідженнях роблять Python доцільним інструментом для реалізації практико-орієнтованого навчання майбутніх фахівців.

Разом із тим проблема адаптації змісту курсу основи програмування саме до потреб студентів спеціалізації «Прикладна лінгвістика» потребує подальшого дослідження.

Основна частина

Python є однією з найпоширеніших мов програмування у сфері аналізу даних та роботі зі штучним інтелектом. Для здобувачів гуманітарного профілю мова програмування є зручною завдяки простому та зрозумілому синтаксису, невисокому порогу входження, можливості швидко отримувати результат, наявності великої кількості готових бібліотек, застосуванню у сфері NLP.

Відповідно до підходу, запропонованого М. Хаммондом, вивчення програмування здобувачами-лінгвістами повинно базуватися на виконанні завдань, пов'язаних з аналізом текстів, пошуком мовних закономірностей, опрацюванням корпусних даних та автоматизацією лінгвістичних досліджень. Така організація навчання надає можливість одночасно формувати алгоритмічне мислення та професійні компетентності майбутніх фахівців [4].

Після опанування базових конструкцій мови Python доцільно поступово переходити від абстрактних навчальних вправ до професійно спрямованих завдань, пов'язаних з автоматизованим опрацюванням мовних даних. На початковому етапі це можуть бути операції з текстовими рядками та словниковими структурами, підрахунок частотності лексем, пошук заданих мовних одиниць і шаблонів. Надалі практичні завдання доцільно ускладнювати шляхом роботи з колекціями текстів, регулярними виразами, структурованими даними та спеціалізованими бібліотеками для обробки природної мови [4; 7; 9].

Така послідовність допомагає реалізувати принцип поступового ускладнення навчальної діяльності: від опанування окремої програмної конструкції – до її застосування для розв'язання конкретного професійного завдання. Однак, здобувач не лише засвоює синтаксис Python, а й усвідомлює функціональне призначення відповідного інструмента в контексті прикладної лінгвістики.

На нашу думку, кожна тема дисципліни «Основи програмування» має супроводжуватися практичним завданням, зміст якого безпосередньо пов'язаний із майбутньою професійною діяльністю здобувача. Відповідно, ефективність такого завдання визначається не складністю програмного коду, а його здатністю продемонструвати практичну цінність програмування для розв'язання лінгвістичного завдання (табл. 1).

Таблиця 1. Приклади практико-орієнтованих завдань на мові Python для здобувачів ОПП “Прикладна (комп’ютерна) лінгвістика та англійська мова”

Тема Python	Практичне лінгвістичне завдання	Професійна спрямованість
Змінні та типи даних	Опрацювання окремих мовних одиниць: слово, речення, кількість символів, довжина слова	Представлення мовних даних у програмі
Рядки	Пошук заданих слів, підрядків та мовних конструкцій у тексті	Автоматизація пошуку мовних одиниць
Списки	Формування та опрацювання списку лексем тексту або невеликого корпусу	Структурування мовних даних
Словники	Створення частотного словника тексту	Кількісний аналіз лексики
Умовні оператори	Класифікація текстових елементів за визначеними ознаками	Реалізація правил лінгвістичного аналізу
Цикли	Автоматичний підрахунок частотності слів у тексті	Автоматизація повторюваних операцій
Функції	Створення функцій для токенизації, очищення або нормалізації тексту	Модульна організація програмного рішення
Регулярні вирази	Вилучення дат, електронних адрес, URL, номерів та інших структурованих елементів	Автоматичне вилучення інформації з тексту
Робота з файлами	Пакетне опрацювання колекції текстових файлів	Корпусне опрацювання мовних даних
CSV / JSON	Зберігання текстів та їхніх метаданих у структурованому вигляді	Створення текстоорієнтованих масивів даних
pandas	Порівняння частотності лексем у двох корпусах	Кількісний та статистичний аналіз
matplotlib	Візуалізація частотності слів або статистичних характеристик тексту	Представлення результатів лінгвістичного дослідження
NLTK / spaCy	Токенізація, лематизація, частиномовний аналіз	Автоматична обробка природної мови
Функції та модулі	Створення окремих компонентів текстового аналізатора	Розроблення модульного програмного забезпечення

Завдяки практико-орієнтованим завданням на мові Python здобувачі зможуть зосереджуватися не лише на технічних аспектах кодування, а й на розв'язанні змістових професійних завдань.

Даний практико-орієнтований підхід доцільно реалізовувати через такі принципи:

- професійна релевантність завдань;
- навчання через проєкти;
- поетапне ускладнення завдань.

Здобувачі впродовж навчання курсу зможуть виконувати наступні мініпроєкти, наприклад:

- програма обчислення частоти слів у тексті;
- автоматичне очищення текстових даних;
- генератор тестових завдань з лексики;
- простий чат-бот для мовної практики;
- класифікація текстів за тематикою.

На початку вивчення дисципліни “Основи програмування” засвоюються базові конструкції, далі – робота з файлами, регулярними виразами, бібліотеками для аналізу тексту.

Практико-орієнтоване навчання основ програмування передбачає спрямування змісту навчальної діяльності на формування не лише предметних знань і навичок програмуванню, а й загальних та фахових компетентностей, необхідних майбутньому фахівцеві для розв'язання прикладних завдань. Такий підхід набуває особливого значення у підготовці здобувачів ОПП “Прикладна (комп'ютерна) лінгвістика та англійська мова”, оскільки сучасна професійна діяльність фахівця передбачає застосування інформаційних технологій для розв'язання практичних завдань у галузі опрацювання мовної інформації та автоматичного аналізу тексту [11; 12].

До основних переваг практико-орієнтованого підходу до навчання програмуванню доцільно віднести використання професійно спрямовані завдання, які впливають на підвищення мотивації до навчання здобувачів, формуванню загальних та професійних компетентностей, розвитку алгоритмічного й логічного мислення, усвідомленню практичної цінності програмування та підготовці до використання сучасних технологій автоматичного опрацювання природної мови. Для здобувачів прикладної лінгвістики принципово важливо, щоб програмні конструкції вивчалися не ізольовано, а застосовувалися для розв'язання задач, пов'язаних із майбутньою професійною діяльністю.

Практична спрямованість навчання надає можливість встановлювати безпосередній зв'язок між абстрактними поняттями програмування та конкретними лінгвістичними завданнями. Наприклад, вивчення циклів і словників може реалізовуватися через створення частотного словника тексту; опанування рядкових операцій – через пошук і нормалізацію мовних одиниць; регулярних виразів – через автоматичне вилучення дат, електронних адрес, термінів та інших структурованих елементів тексту; функцій – через розроблення окремих програмних компонентів системи автоматичного аналізу тексту. Така організація навчальної діяльності сприяє усвідомленню здобувачами того, що програмування є не самоцільлю, а інструментом професійної діяльності прикладного лінгвіста.

Водночас упровадження практико-орієнтованого підходу супроводжується низкою труднощів, серед яких різний початковий рівень цифрової та математичної підготовки здобувачів, психологічні бар'єри та невпевненість гуманітарно орієнтованих здобувачів в процесі опанування програмування, обмежений обсяг навчального часу, а також необхідністю інтеграції програмного та лінгвістичного змісту. Для подолання зазначених труднощів доцільно застосовувати: диференційовані та адаптивні завдання, поступове ускладнення практичних кейсів, роботу в парах і малих групах, інтерактивні середовища програмування, використання готових наборів мовних даних та поетапну систему професійної підтримки.

З огляду на зазначене, зміст дисципліни «Основи програмування» доцільно структурувати навколо професійно орієнтованих практичних кейсів. Кожна тема вивчення мови Python має бути пов'язана з конкретною лінгвістичною або програмно-інженерною задачею, а її виконання – із формуванням визначених освітньою програмою компетентностей і досягненням відповідних програмних результатів навчання. Такий підхід забезпечує прозорий зв'язок між змістом дисципліни, професійними завданнями майбутнього фахівця та вимогами освітнього стандарту [11].

Для демонстрації зазначеного взаємозв'язку розроблено матрицю відповідності «тема Python – практичний кейс – фахова компетентність – програмний результат навчання» (табл. 2).

Таблиця 2. Матриця відповідності тем мови Python, практичних кейсів, фахових компетентностей та програмних результатів навчання

Практичний кейс / професійна задача	Тема Python	Застосування	Компетентності	ПРН
Частотний аналіз тексту	Цикли, словники	Підрахунок частотності слів, формування частотного словника, визначення найбільш уживаних лексем	ЗК 6, ЗК 10; ФК 8, ФК 21, ФК 23	ПРН 6, ПРН 29
Пошук ключових слів у тексті	Рядки, списки, словники	Виявлення та підрахунок заданих лексичних одиниць	ЗК 4, ЗК 6, ЗК 10; ФК 8, ФК 21	ПРН 6
Визначення мови тексту	Умовні оператори, словники	Порівняння частотності символів/слів та визначення ймовірної мови тексту	ЗК 6, ЗК 10; ФК 21, ФК 23	ПРН 6, ПРН 27, ПРН 29
Вилучення e-mail, дат, URL	Регулярні вирази	Автоматичне вилучення структурованої інформації з тексту	ЗК 6, ЗК 10; ФК 8, ФК 18, ФК 21	ПРН 6, ПРН 31
Автоматизований аналіз корпусу	Файли, списки, словники, цикли	Опрацювання колекції текстових файлів, агрегація та систематизація мовних даних	ЗК 5, ЗК 6, ЗК 10; ФК 8, ФК 18	ПРН 6, ПРН 31
Створення текстоорієнтованої бази даних	CSV, JSON, SQLite	Зберігання, пошук, фільтрація та оновлення текстових даних	ЗК 10; ФК 8, ФК 18, ФК 19	ПРН 26, ПРН 31
Простий чат-бот	Функції, умовні оператори, словники	Реалізація діалогу на основі ключових слів та сценаріїв	ЗК 6, ЗК 7, ЗК 10; ФК 16, ФК 21	ПРН 6, ПРН 26, ПРН 28, ПРН 29
Аналіз тональності тексту	Списки, словники, функції	Визначення позитивної/негативної лексики та розрахунок умовної тональності	ЗК 6, ЗК 9, ЗК 10; ФК 11, ФК 21, ФК 23	ПРН 6, ПРН 27
Порівняльний аналіз двох корпусів	pandas, словники, функції	Порівняння частотності лексики та статистичних	ЗК 5, ЗК 6, ЗК 10; ФК 1,	ПРН 7

		характеристик двох наборів текстів	ФК 8, ФК 11, ФК 23	
Візуалізація мовних даних	matplotlib, pandas	Побудова графіків частотності, розподілу довжини речень, лексичного різноманіття	ЗК 5, ЗК 10, ЗК 11; ФК 8, ФК 23	ПРН 6
Розроблення мінісистеми NLP	Функції, модулі, класи	Об'єднання окремих програмних компонентів у систему автоматичного аналізу тексту	ЗК 6, ЗК 7, ЗК 10; ФК 14–19, ФК 21	ПРН 23, ПРН 24, ПРН 26, ПРН 29, ПРН 30, ПРН 32
Тестування лінгвістичної програми	Функції, винятки, модульне тестування	Перевірка коректності роботи програми на різних типах текстових даних	ЗК 6, ЗК 10; ФК 17, ФК 19, ФК 21	ПРН 26, ПРН 32

Наведені приклади демонструють, що практико-орієнтоване навчання програмуванню здобувачів спеціалізації «Прикладна лінгвістика» може бути побудоване на послідовному переході від базових програмних конструкцій до професійно значущих лінгвістичних завдань. Вивчення циклів, умовних операторів, структур даних, функцій і регулярних виразів набуває прикладного змісту через їх використання для аналізу текстів, опрацювання корпусів, вилучення структурованої інформації та створення прототипів мовних технологій.

Отже, практичний кейс у матриці відповідності «тема Python – практичний кейс – фахова компетентність – програмний результат навчання» є не лише способом закріплення теоретичного матеріалу, а механізмом інтеграції програмних, лінгвістичних і дослідницьких знань. Результатом виконання кейсу буде не тільки розроблений здобувачем програмний код, а завершений продукт діяльності здобувача – алгоритм, програмний модуль, база мовних даних, прототип інформаційної системи або результати автоматизованого лінгвістичного дослідження.

Висновки

Практико-орієнтоване навчання основ програмування здобувачів спеціалізації «Прикладна лінгвістика» є ефективним засобом формування загальних та фахових компетентностей. Використання мови програмування Python надає можливість поєднувати вивчення базових алгоритмічних конструкцій із розв'язанням реальних задач обробки мовних даних.

Професійно спрямовані завдання сприяють підвищенню мотивації до навчання здобувачів, полегшують засвоєння складних тем та створюють підґрунтя для подальшого вивчення штучного інтелекту, машинного перекладу та комп'ютерної лінгвістики.

Перспективою подальших досліджень є розроблення методики використання Python під час проєктного навчання в професійній підготовці майбутніх прикладних лінгвістів.

Список використаних джерел

1. Digital Education Action Plan 2021-2027. *European Education Area Quality education and training for all*. URL: <https://education.ec.europa.eu/focus-topics/digital-education/actions/plan>.
2. Про схвалення Концепції розвитку природничо-математичної освіти (STEM-освіти) : Розпорядження Кабінету Міністрів від 05.08.2020, № 960-р. URL: <https://zakon.rada.gov.ua/laws/show/960-2020-p#Text>.
3. Ivashkevych L. Teaching Programming With Python for Linguists: Whys and How-Tos. *Advanced Linguistics*. 2019. Т. 3. URL: <https://doi.org/10.20535/2617-5339.2019.3.165687>.
4. Hammond M. Python for Linguists. Cambridge : Cambridge University Press, 2020. 300 p.
5. Eisenstein J. Introduction to Natural Language Processing. Cambridge, MA: MIT Press, 2019. 544 p.
6. Weisser M. Python Programming for Linguistics and Digital Humanities: Applications for Text-Focused Fields. Hoboken : Wiley, 2024. 288 p.
7. Loper E., Bird S. NLTK: The Natural Language Toolkit // Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia, 2002. P. 63–70. DOI: <https://doi.org/10.3115/1118108.1118117>.
8. Qi P., Zhang Y., Zhang Y., Bolton J., Manning C. D. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations. 2020. P. 101–108. DOI: <https://doi.org/10.18653/v1/2020.acl-demos.14>.
9. Unpingco J. Python Programming for Data Analysis. Cham : Springer, 2021. 263 p. DOI: 10.1007/978-3-030-68952-0.
10. Sweigart A. Automate the Boring Stuff with Python: Practical Programming for Total Beginners. 2nd ed. San Francisco : No Starch Press, 2019. 592 p.
11. Наказ МОН № 869 Про затвердження стандарту вищої освіти за спеціальністю 035 «Філологія» для першого (бакалаврського) рівня вищої освіти від 20.06.2019 року URL: <https://mon.gov.ua/static-objects/mon/sites/1/vishcha-osvita/zatverdzeni%20standarty/2021/07/28/035-Filolohiya-bakalavr.28.07-1.pdf>
12. Освітньо-професійна програма «Прикладна (комп'ютерна) лінгвістика та англійська мова» URL: <https://drive.google.com/file/d/1aJ0dtHBTzfKYqddzgnEuuJK-dFfFDwey/view>

**Свистунов Антон Олександрович, асистент
Омельчук Людмила Леонідівна, к.ф.-м.н, доцент**

**ІМПЛЕМЕНТАЦІЯ ПОЛОЖЕННЯ ПРО ВИКОРИСТАННЯ
ШТУЧНОГО ІНТЕЛЕКТУ ПРИ РЕАЛІЗАЦІЇ ОСВІТНІХ ПРОГРАМ У
ЗМІСТІ НАВЧАЛЬНОЇ ДИСЦИПЛІНИ «WEB-ТЕХНОЛОГІЇ»**

Роботу присвячено практичному досвіду впровадження Положення про використання штучного інтелекту при реалізації освітніх програм на прикладі вибіркової дисципліни «WEB-технології» для здобувачів першого (бакалаврського) рівня вищої освіти, що реалізується за спеціальністю «Комп'ютерні науки». Розглянуто обґрунтування обраного Рівня 3 «Переважно дозволене використання генеративного штучного інтелекту», особливості інтеграції інструментів штучного інтелекту в освітній процес, а також підходи до забезпечення прозорості та академічної доброчесності при їх використанні. Окрему увагу приділено застосуванню платформи NotebookLM для систематизації навчальних матеріалів та підготовки до контрольних робіт, а також використання шаблонів Git-репозиторіїв із механізмами автоматичного логування дій ШІ-агентів і контролю декларування використання штучного інтелекту за допомогою pre-commit хуків.

Ключові слова: штучний інтелект, WEB-технології, NotebookLM, Git, академічна доброчесність, генеративний ШІ, pre-commit хуки.

The paper is devoted to the practical experience of implementing the Regulation on the Use of Artificial Intelligence in the Implementation of Educational Programmes, using the elective discipline “Web Technologies” as an example. The discipline is offered to first-cycle (Bachelor’s) higher education students enrolled in the Computer Science speciality. The paper discusses the rationale for selecting Level 3, “Predominantly permitted use of generative artificial intelligence”, the specific features of integrating artificial intelligence tools into the educational process, and approaches to ensuring transparency and academic integrity in their use. Special attention is paid to the use of the NotebookLM platform for systematising learning materials and preparing students for tests and assessments, as well as to the use of Git repository templates with mechanisms for automatically logging the actions of AI agents and monitoring the declaration of artificial intelligence use through pre-commit hooks.

Keywords: artificial intelligence, WEB technologies, NotebookLM, Git, academic integrity, generative AI, pre-commit hooks.

Вступ

Стрімкий розвиток технологій генеративного штучного інтелекту (далі – ШІ) суттєво змінює освітнє середовище закладів вищої освіти: трансформується характер навчальних завдань, з'являються нові інструменти для самостійної роботи студентів, а викладачі отримують можливість переосмислити підходи до організації освітнього процесу [1; 2]. У відповідь на ці виклики Київський національний університет імені Тараса Шевченка розробив і затвердив Положення про використання штучного інтелекту при реалізації освітніх програм (далі – Положення) [3], яке вводить чітку систему рівнів дозволу використання генеративного ШІ в освітньому процесі.

Дисципліна «WEB-технології» для студентів спеціальності «Комп'ютерні науки» освітньо-професійної програми «Інформатика», що реалізується факультетом комп'ютерних наук та кібернетики Київський національний університет імені Тараса Шевченка є практично орієнтованою та безпосередньо пов'язаною з інструментами розроблення ПЗ, зокрема, ШІ-асистентами, що займають дедалі важливіше місце в роботі розробника програмного забезпечення [4; 5]. Ці тенденції викликають необхідність інтеграції ШІ-інструментів у освітній процес із чітким дотриманням принципів академічної доброчесності та прозорості [3; 6].

У даній роботі представлено практичний досвід імплементації університетського положення про ШІ у робочій програмі дисципліни «WEB-технології». Зокрема, розглядаються: обґрунтування обраного рівня використання генеративного ШІ; шляхи застосування платформи NotebookLM для систематизації матеріалу та підготовки студентів до контрольних заходів; шаблони Git-репозиторіїв з автоматизованими механізмами контролю декларування використання ШІ.

Основна частина

1. Обґрунтування рівня використання генеративного штучного інтелекту

Відповідно до Положення [3], для кожного освітнього компонента визначається один із трьох рівнів дозволу використання генеративного ШІ: Рівень 1 (повна заборона), Рівень 2 (обмежено дозволене використання з обов'язковою авторською адаптацією) та Рівень 3 («Переважно дозволене використання генеративного штучного інтелекту»). Для дисципліни «WEB-

технології» обрано Рівень 3. Обґрунтування цього вибору базується на двох ключових аргументах.

По-перше, генеративний ШІ суттєво зменшує час переходу від теорії до практичного результату. У сучасній веброзробці фахівці широко використовують ШІ-асистенти (GitHub Copilot, Claude, ChatGPT тощо) для прискорення написання коду, відлагодження, генерації тестових даних, документування API [4; 5]. Дослідження GitHub свідчать, що розробники, які використовують ШІ-асистенти, виконують типові завдання на 55% швидше порівняно з тими, хто не використовує таких інструментів [4]. Студенти повинні опанувати навички ефективної взаємодії з такими інструментами, критично оцінювати результати їх роботи та інтегрувати у власний робочий процес. Штучна заборона ШІ створювала б навчальне середовище, яке не відповідає умовам фахової діяльності.

По-друге, використання ШІ дозволяє студентам абстрагуватися від деталей конкретної реалізації та сконцентруватися на архітектурних принципах і концептуальному розумінні предметної галузі. Дисципліна «WEB-технології» охоплює широкий технологічний стек: від протоколів HTTP та принципів побудови REST API до React-фреймворку та контейнеризації (Docker/Kubernetes). Для студентів принципово важливіше розуміти концепцію розподілених систем чи принцип CAP-теореми, ніж витрачати значний час на синтаксичні деталі конкретної мови програмування або конфігураційні особливості конкретного фреймворку. ШІ-асистент виступає у ролі «розумного автодоповнення», що вивільняє когнітивні ресурси для глибшого осмислення фундаментальних концепцій [7].

Разом з тим, Рівень 3 передбачає обов'язкове декларування факту використання ШІ: студенти зобов'язані фіксувати у своїх роботах та репозиторіях інформацію про застосовані ШІ-інструменти. Це формує навички відповідального й прозорого використання технологій, що є ключовою вимогою Положення [3] та відповідає міжнародним рекомендаціям у сфері освіти та ШІ [2] та вимогам Закону України про академічну доброчесність [6].

2. Механізми формування фахових компетентностей в умовах дозволеного використання генеративного штучного інтелекту

Запровадження Рівня 3 використання генеративного ШІ актуалізує проблему відповідності навчального середовища задекларованим результатам навчання. Положення КНУ [3] прямо зазначає, що «використання генеративного штучного інтелекту саме по собі не змінює

змісту критеріїв оцінювання» та що оцінюванню підлягає «досягнення запланованих результатів навчання, рівень самостійності, обґрунтованість висновків, коректність застосування методів». Отже, перед викладачем постає задача проєктування форм контролю, що залишаються валідними і диференційованими в умовах широкої доступності ІІІ-інструментів. Доступність генеративного ІІІ зміщує пізнавальне навантаження від синтезу та відтворення інформації – на її критичну оцінку, адаптацію та застосування в нових контекстах. Це відповідає верхнім рівням таксономії Блума (аналіз, оцінювання, творення) [8] і вимагає переорієнтації методів оцінювання з продукту діяльності на процес і розуміння.

Усний захист як інструмент верифікації розуміння. Основною формою оцінювання в дисципліні є усний захист кожного виконаного етапу проєкту. Ця форма контролю є методично обґрунтованою в умовах використання ІІІ, оскільки перевіряє не артефакт (код, документацію), а здатність студента обґрунтувати прийняті архітектурні та технологічні рішення, пояснити альтернативи, що розглядались, та виявити обізнаність із потенційними обмеженнями реалізованого підходу. Відповідно до критеріїв оцінювання, зафіксованих у робочій програмі [9], повний бал присвоюється студенту, який «логічно та аргументовано пояснює реалізацію, демонструє результат і впевнено відповідає на запитання». Таким чином, незалежно від ступеня залучення ІІІ до написання коду, верифікація розуміння здійснюється через безпосередню взаємодію з викладачем.

Документування як складова особистого внеску. Журнали змін та інші форми звітування фіксують не лише факт використання ІІІ, а й характер взаємодії студента з інструментом: сформульовані запити, отримані результати, внесені корективи. Таке документування є операціоналізацією поняття «особистий внесок» у контексті людино-машинної взаємодії та відповідає вимозі Положення КНУ [3] щодо «розмежування власного внеску здобувача освіти та згенерованих елементів». Ведення такого журналу також є відтворенням реальних практик промислової розробки програмного забезпечення, де аудит ІІІ-взаємодій стає дедалі поширенішою вимогою [10].

Формування навичок критичної оцінки результатів генерації. Ефективне використання генеративного ІІІ є самостійною фаховою компетентністю, формування якої має бути педагогічно керованим. У межах курсу студентам пояснюються принципи формулювання запитів до ІІІ-систем, методи верифікації згенерованих результатів, зокрема, перевірка на типові вразливості, відповідність прийнятим архітектурним рішенням, правильність обробки граничних випадків та механізми документованої

адаптації результатів до вимог конкретного проєкту. Така підготовка узгоджується з рекомендаціями ЮНЕСКО щодо формування «цифрової грамотності, критичного мислення і навичок відповідального використання ШІ» [2].

Оцінювання процесу розроблення поряд із результатом. Поряд із функціональністю кінцевого застосунку, що є результатом проєкту, критеріями оцінювання передбачено: якість архітектури та обґрунтованість структурних рішень; повноту і коректність документації; якість взаємодії клієнт–сервер та обробки виняткових ситуацій; динаміку роботи над проєктом, що відстежується через історію комітів у репозиторії. Зазначені показники є слабо замінними через пряме використання виводу ШІ без глибокого залучення студента в процес розроблення, що забезпечує диференційованість оцінювання і в умовах широкого застосування ШІ-інструментів.

Таким чином, описана система оцінювання спрямована на формування фахівця, здатного не лише застосовувати ШІ-інструменти, але й розуміти, проєктувати, критично аналізувати та відповідально використовувати програмні рішення, що відповідає програмним результатам навчання дисципліни [9]:

- ПРН18.2. Аналізувати, оцінювати і вибирати інструментальні та обчислювальні засоби, парадигми, технології, алгоритмічні і програмні рішення при проєктуванні та розробці програмних систем.
- ПРН20.2. Мотивовано обирати технології програмування для розв’язання завдань створення і супроводження програмного забезпечення.

3. Платформа NotebookLM як інструмент систематизації навчальних матеріалів та підготовки до контрольних робіт

Платформа NotebookLM від Google є ШІ-інструментом для роботи зі знаннями, що дозволяє завантажувати різноманітні джерела інформації – документи, презентації, відео, посилання на веб-сторінки – та взаємодіяти з ними через природномовний інтерфейс [11]. У межах курсу «WEB-технології» запроваджена наступна практика.

Перед кожною лекцією або контрольною роботою викладач формує в NotebookLM тематичну колекцію («notebook»), до якої завантажуються: лекційні слайди та матеріали; рекомендовані наукові статті та технічна документація (RFC, W3C-специфікації, MDN); відеозаписи лекцій та

семінарів; посилання на відповідні розділи Google Classroom курсу. Студенти отримують спільний доступ до цих колекцій і можуть опрацьовувати матеріал у зручний для них спосіб та в зручний час.

Важливою педагогічною перевагою NotebookLM є підтримка різних форматів взаємодії з матеріалом, що забезпечує інклюзивність та доступність навчання для студентів з різними навчальними стилями та потребами. Зокрема, платформа пропонує: аудіо- та відеопереказ ключових тез (зручно для студентів, що краще сприймають інформацію на слух, або для осіб з вадами зору); автоматично згенеровані флеш-картки для самоперевірки знань термінів і понять; мапи думок для візуалізації зв'язків між концепціями (наприклад, між протоколами, архітектурними патернами або компонентами мікросервісної архітектури); інтерактивний режим запитань та відповідей на основі завантажених джерел із посиланнями на конкретні місця в матеріалах. Приклад тематичної колекції в середовищі NotebookLM для Темі 10 «Тема JavaScript Features. Web APIs. Navigation & other Browser APIs. Third-party APIs. Graphics (2D, 3D). Closures. Classes & Objects (OOP). The scope and “this” keyword. Threads. Asynchronous JavaScript: Promises, async+await. Timers (timeout, interval)» показано на Рисунку 1.

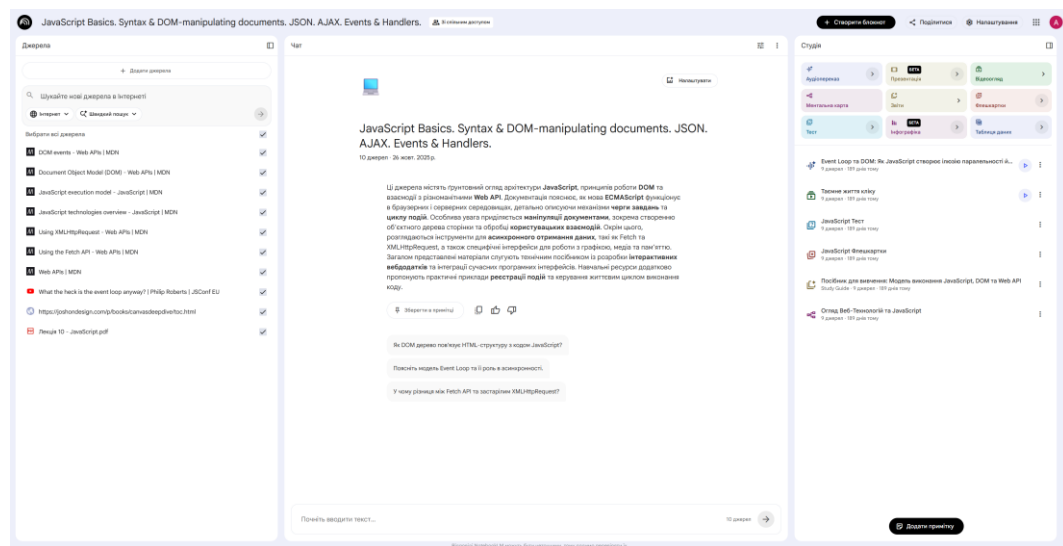


Рис.1. Приклад тематичної колекції в середовищі NotebookLM. Зліва направо: панель джерел – конспекти лекцій та посилання на статті чи відео, чат-інтерфейс, «студія» – згенеровані аудіо- та відеоперекази, мапи думок, тести та інше

Напередодні контрольних робіт студенти мають можливість проходити симуляції тестування: NotebookLM генерує питання на основі матеріалів теми, пояснює правильні відповіді з посиланням на конкретні джерела. Це дозволяє студентам здійснювати ефективну та верифіковану самопідготовку. Описаний підхід узгоджується з принципом академічної свободи у використанні ШІ поза межами оцінюваних завдань, що передбачений Положенням [3], і одночасно формує у студентів навички критичного опрацювання навчальних матеріалів за допомогою ШІ-інструментів.

4. Шаблони Git-репозиторіїв для контролю декларування використання ШІ

Проектна складова дисципліни «WEB-технології» передбачає публікацію всіх результатів у Git-репозиторії (GitHub, GitLab або аналог), причому історія комітів (внесених змін) та активність роботи враховуються під час оцінювання [9]. З метою технічної підтримки вимог Положення щодо обов'язкового декларування використання генеративного ШІ розроблено стандартизований шаблон репозиторію. Шаблон реалізований у вигляді GitHub Template Repository і містить декілька ключових компонентів.

Файл AGENTS.md розміщується у кореневій директорії репозиторію та містить інструкції для ШІ-агентів – описи дозволених дій, очікуваного формату виведення, вимог до документування та обмежень, які студент встановлює для ШІ-інструментів у межах свого проєкту. Концепція файлів AGENTS.md (або аналогічних SKILL.md) набула поширення як стандарт керування поведінкою автономних ШІ-агентів у репозиторіях [12].

Ключовою вимогою, зафіксованою в AGENTS.md, є обов'язкове ведення структурованого журналу використання ШІ у файлі AI_USAGE.md. Кожного разу, коли ШІ-агент виконує будь-яку дію в репозиторії (генерує код, вносить зміни, формує коміт) він зобов'язаний додати запис до AI_USAGE.md із зазначенням: дати та часу дії; назви використаного інструменту (моделі / сервісу); короткого опису виконаної операції та її результату; посилання на відповідний коміт або файл. Файл AI_USAGE.md є структурованим журналом взаємодії з ШІ-інструментами впродовж усього проєкту та одночасно слугує декларацією про використання ШІ у відповідності до вимог Положення [3].

Для автоматизованого контролю дотримання вимог щодо декларування в шаблон репозиторію включено pre-commit хуки. Pre-commit

хук – це скрипт, що автоматично виконується перед кожним `git commit` і може відхилити коміт у разі виявлення порушень встановлених правил [13]. Хуки реалізовані з використанням фреймворку `pre-commit` і налаштовуються через конфігураційний файл `.pre-commit-config.yaml` у кореневій директорії репозиторію.

У шаблоні реалізовано наступні автоматичні перевірки: наявність файлу `AI_USAGE.md` у репозиторії; відсутність порожнього файлу `AI_USAGE.md`; відповідність формату записів у `AI_USAGE.md` встановленому шаблону (дата, назва інструменту, опис, посилання на коміт/файл). У разі невиконання будь-якої з умов коміт відхиляється з відповідним повідомленням-підказкою для студента.

Такий підхід має декілька взаємодоповнюючих переваг. По-перше, він автоматизує рутинну перевірку, звільняючи час викладача для змістовного зворотного зв'язку. По-друге, формує у студентів розуміння того, що документування ІІІ-взаємодій є невід'ємною складовою сучасної культури розробника програмного забезпечення [10]. По-третє, забезпечує прозорий і технічно верифікований аудит використання ІІІ в межах освітнього компонента, що відповідає принципу підзвітності, закладеному в Положенні [3].

Висновки

У роботі представлено три взаємодоповнюючі практичні підходи до імплементації Положення про використання ІІІ при реалізації освітніх програм при викладанні дисципліни «WEB-технології». Обраний Рівень 3 дозволу використання генеративного ІІІ узгоджується з реальними вимогами фахової діяльності розробника програмного забезпечення та орієнтує освітній процес на розвиток критичного мислення й відповідального застосування ІІІ-інструментів. Платформа NotebookLM забезпечує диференційований та інклюзивний доступ студентів до навчального контенту, підтримуючи різні навчальні стилі та потреби. Шаблони Git-репозиторіїв із механізмами автоматичного логування (`AI_USAGE.md`) та `pre-commit` хуками перетворюють вимогу декларування використання ІІІ на природну та технічно підкріплену складову робочого процесу студента.

Описані підходи є відтворюваними та можуть бути адаптовані для інших дисциплін технічного спрямування, де використання ІІІ є релевантним та педагогічно виправданим. Подальші дослідження можуть бути спрямовані на кількісну оцінку впливу застосованих підходів на якість навчання студентів та рівень їхньої академічної доброчесності.

Список використаних джерел

1. Mollick E., Mollick L. Instructors as Innovators: a Future-focused Approach to New AI and Tech-enhanced Teaching Models with Agency and Efficacy. *Research Papers in Education*. 2023. Vol. 38, Is. 5. P. 790–808.
2. UNESCO Recommendation on the Ethics of Artificial Intelligence. Paris : UNESCO, 2021. URL: <https://unesdoc.unesco.org/ark:/48223/pf0000381137> (дата звернення: 20.04.2026).
3. Положення про використання штучного інтелекту при реалізації освітніх програм у Київському національному університеті імені Тараса Шевченка : проєкт, схвалений на засіданні Науково-методичної ради КНУ, протокол № 03-26 від 19 лют. 2026 р.
4. Kalliamvakou E. Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness. *GitHub Blog*. 2022. URL: <https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/> (дата звернення: 20.04.2026).
5. Liang J. T., Yang C., Myers B. A. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE '24)*, April 14–20, 2024, Lisbon, Portugal. New York : ACM, 2024. P. 1–13. DOI: 10.1145/3597503.3608128.
6. Про академічну доброчесність : Закон України від 18 груд. 2025 р. № 4742-IX. URL: <https://zakon.rada.gov.ua/laws/show/4742-20#Text> (дата звернення: 02.05.2026).
7. Bender E. M., Gebru T., McMillan-Major A., Shmitchell S. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FACCT '21)*. New York : ACM, 2021. P. 610–623.
8. Taxonomy of Educational Objectives: The Classification of Educational Goals. *Handbook I: Cognitive Domain* / ed. B. S. Bloom. New York : David McKay Company, 1956. 207 p.
9. Свистунов А. О. Робоча програма навчальної дисципліни «WEB-технології». Київ : КНУ імені Тараса Шевченка, факультет комп'ютерних наук та кібернетики, кафедра теорії та технології програмування, 2026.

10. Floridi L., Cows J., Beltrametti M., Chatila R., Chazerand P., Dignum V., Luetge C., Madelin R., Pagallo U., Rossi F., Schafer B., Valcke P., Vayena E. An Ethical Framework for a Good AI Society: Opportunities, Risks, Principles, and Recommendations. *Minds and Machines*. 2018. Vol. 28. P. 689–707.
11. NotebookLM : [вебзастосунок]. Google LLC, 2023. URL: <https://notebooklm.google.com> (дата звернення: 20.04.2026).
12. Anthropic. CLAUDE.md / AGENTS.md Specification for AI Agent Instructions in Repositories. Anthropic Developer Documentation. 2024–2025. URL: <https://docs.anthropic.com/en/docs/claude-code/memory> (дата звернення: 20.04.2026).
13. pre-commit : a framework for managing and maintaining multi-language pre-commit hooks : [програмний засіб]. URL: <https://pre-commit.com> (дата звернення: 20.04.2026).

Ткаченко Олексій Миколайович, к.т.н, доцент

ДОСВІД ВИКОРИСТАННЯ ІІІ ДЛЯ ФОРМАЛЬНОЇ СПЕЦИФІКАЦІЇ В КУРСІ "ФОРМАЛЬНІ МЕТОДИ РОЗРОБКИ ПРОГРАМНИХ СИСТЕМ"

У статті проаналізовано досвід використання ІІІ для формальної специфікації програмних систем (Dafny, TLA+) магістрантами у 2025 та 2026 роках. Досліджено еволюцію використання ІІІ-моделей та перехід від інтерактивного генерування коду до розробки автономних агентів. Виявлено зростання рівня успішності верифікації, а також зміну характеру помилок моделей від простих синтаксичних до глибоко семантичних. Обґрунтовано зміну парадигми розробки: ключовою навичкою фахівця стає інженерія формальних вимог, тоді як ІІІ виконує автоматизоване генерування та верифікацію коду.

Ключові слова: формальна специфікація, ІІІ

The work analyzes the use of AI for formal software specification (Dafny, TLA+) by M.S. students in 2025 and 2026. The evolution of AI models and the transition from interactive code generation to autonomous agents are investigated. An increase in verification success rate and a shift in model errors from simple syntactic to deeply semantic are revealed. A development paradigm shift is substantiated: the specialist's key skill becomes formal requirements engineering, while AI performs automated code generation and verification.

Keywords: formal specification, AI

Вступ

Впровадження формальних методів розробки програмного забезпечення історично розглядалося як ресурсоємний процес, виправданий лише у сферах з критичними вимогами до безпеки.

Згідно з даними NASA і програмами досліджень DARPA, верифікація ПЗ може складати до 88% загальних витрат на створення систем високої надійності [1]. Тривалий час фундаментальні підходи до проектування, розроблення і тестування залишалися незмінними, що призводило до стрімкого зростання собівартості розробки у поєднанні з ускладненням самих систем. Ще відносно недавно частка програмних систем у світі, при розробці яких застосовувалися формальні методи та інструменти, становила менше

відсотка, і ці системи були переважно в аерокосмічній, оборонній та ядерній галузях.

Однак динаміка за останні 10 років суттєво змінилася. Сучасні апаратні та програмні архітектури досягли рівня складності, який не дозволяє забезпечити ефективне покриття тестами. У 2025 році глобальний ринок інструментів формальної верифікації оцінювався у 1,8 мільярда доларів США, а до 2034 року очікується його зростання до 6,2 мільярда доларів при середньорічному темпі зростання 14,7%. Сьогодні середньостатистична команда розробників, наприклад, напівпровідників витрачає від 28% до 34% загального бюджету верифікації саме на формальні методи, що свідчить про перехід цих технологій з академічної ніші до індустріального стандарту. Окрім галузі напівпровідників (31,5% ринку), драйверами є аерокосмічна (22,4%) та автомобільна галузь (19,8%) [2].

Очевидним паралельним трендом індустрії ПЗ є розвиток і використання великих мовних моделей (LLM). У 2023 році близько 70% розробників повідомляли про використання інструментів ШІ, у 2024 році цей показник зріс до 76%, а згідно з глобальним опитуванням Stack Overflow, 84% розробників почали використовувати ШІ у своєму робочому процесі, при цьому 51% професійних інженерів застосовують інструменти штучного інтелекту щодня [3]. У той же час, 46% розробників зазначають, що не довіряють точності згенерованого коду, що створює потребу у формальних методах його перевірки [4]. Це посилює потребу у вивченні відповідних методів та інструментів на IT-спеціальностях.

Основна частина

У 2025 та 2026 роках в межах магістерського курсу "Формальні методи розробки програмних систем", який викладається для студентів освітньо-наукової програми "Інформатика" спеціальності 122/F3 "Комп'ютерні науки" на факультеті комп'ютерних наук та кібернетики в Київському національному університеті імені Тараса Шевченка, у практичне завдання формальної специфікації було включено дослідницький компонент, відповідно до якого студенти повинні були виконати завдання самостійно та з використанням обраної системи ШІ, зробити порівняння обох підходів та виконати рефлексію з обґрунтуванням проблем, що виникли та описом досвіду їх усунення. Формальні мови специфікацій та ШІ для виконання завдання студенти могли обирати на свій розсуд.

Аналіз студентських робіт продемонстрував, що найбільш популярною мовою формальних специфікацій була Dafny [5], дещо меншою

мірою – TLA+ [6]. Еволюція використання ними моделей ШІ подано в табл. 1, а розподіл популярності відображено на рис. 1.

Таблиця 1. Використання ШІ-моделей для формальної специфікації

Сімейство моделей	Версії 2025 року	Частка 2025	Версії 2026 року	Частка 2026
Anthropic Claude	3.5 Sonnet / Opus	26%	3.7 Sonnet / Opus	38%
OpenAI GPT/o-series	GPT-4o / GPT-5.2	42%	OpenAI o1 / GPT-5.5	32%
DeepSeek	DeepSeek V3.2 / R1	21%	DeepSeek-V4 / R2-Pro	22%
Відкриті моделі	Llama 3.1 / OLMo	11%	Llama 3.3 / Mistral Large 3	8%

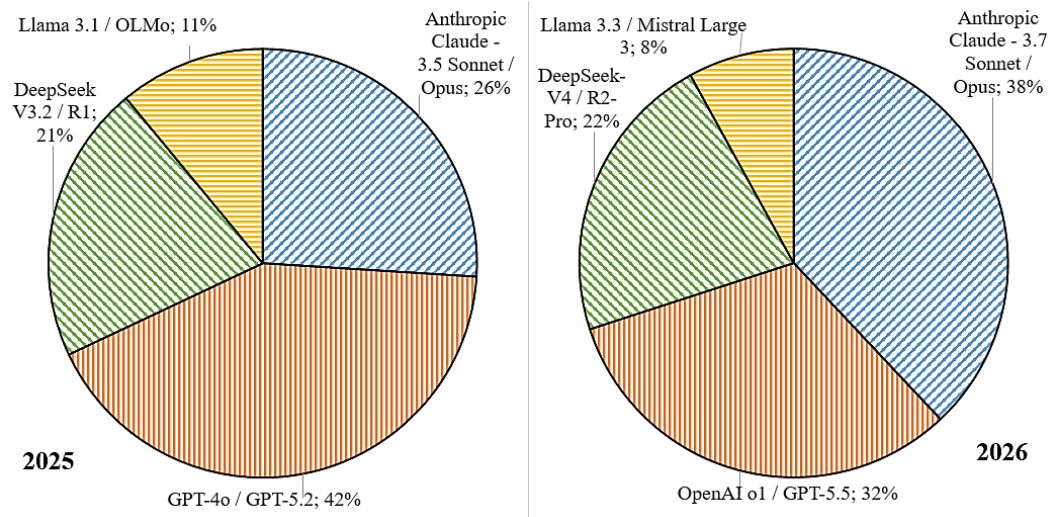


Рис.1. Частка використання студентами ШІ-моделей для формальної специфікації у 2025 та 2026 році

У 2026 році моделі сімейства Claude 3.7 витіснили продукти OpenAI з першого місця. У студентських звітах це пояснюється розширеним вікном контексту Claude та кращою здатністю дотримуватися строгих синтаксичних правил математичних мов. Також 2026 рік відзначився переходом на моделі з

внутрішнім циклом міркування (наприклад, OpenAI o1 та нові ітерації DeepSeek-R2).

Перехід на "reasoning" моделі у 2026 році обумовив покращення здатності ШІ писати специфікації.

Успішність (здатність згенерувати код, який проходить верифікацію) оцінювалася у двох режимах: без втручання людини ("one-shot") та з ітеративним зворотним зв'язком (REPL):

- середній показник "One-shot" успішності: зріс із 28-35% (у 2025) до 48-52% (у 2026). Хоча моделі стали якіснішими, вони все ще не здатні надійно видавати коректний результат з першої спроби;
- середній показник успішності з REPL (ітеративний): зріс із 65-72% (у 2025) до 88-92% (у 2026).

Процент успішності моделей для обраних мов:

- Dafny (82% успіху у 2026 проти 65% у 2025): стрибок успішності зумовлений тим, що нові моделі (Claude 3.7 та o1) були спеціально дотреновані розробниками на синтаксисі Dafny, що відзначили студенти у своїх рефлексіях;
- TLA+ (40% успіху у 2026 проти 29% у 2025): Темпоральна логіка все ще алгоритмічно "незручна" для текстових моделей.

Порівняння виявлених помилок за два роки ілюструє перехід проблем ШІ від синтаксичних до глибоко семантичних.

У 2025 році студенти відзначали:

- грубі синтаксичні помилки (змішування мов);
- плутанина з базовими кванторами (загальності та існування);
- нездатність написати навіть базовий інваріант циклу.

Що змінилося у 2026 році? Кількість синтаксичних помилок суттєво зменшилася. Однак у 2026 році студенти зіткнулися з новими типами помилок:

- "читерство" при доведенні (Proof Cheating): студенти виявили, що стикаючись зі складною теоремою, нові моделі ШІ іноді додають суперечливі аксіоми (наприклад, конструкції типу `assume false`; у Dafny), щоб обдурити солвер і змусити його прийняти доведення;
- деградація інваріантів у складних структурах даних: ШІ у 2026 році навчився писати інваріанти для масивів та простих циклів, але при роботі з графами чи складними деревами знову скочується до генерування сильних умов, які обумовлюють для солвера труднощі з індуктивним доведенням;

- переускладнення специфікацій (Over-specification): нові "reasoning" моделі схильні описувати кожен дрібний стан системи, навіть коли це не вимагається контрактом, що призводить до зависання SMT-солверів.

Висновки

На основі аналізу студентських звітів про виконання практичної частини формальної специфікації самостійно та за допомогою ШІ можна зробити кілька висновків.

1. Бачення студентів щодо готовності ШІ генерувати формальну специфікацію.

У 2025 році студенти робили висновок, що ШІ – просто "розумний автокомпліт", який вимагає інтерактиву з розробником та ручне втручання. У 2026 році понад 60% студентів не використовували веб-чати, а вже почали розробляти агентні пайплайни (Agentic Workflows) через API. Деякі здобувачі писали скрипти, де ШІ-агент самостійно генерував специфікацію, викликав компілятор (наприклад, Dafny), читав помилки, аналізував їх і переписував код, доки не отримував успішну верифікацію.

Загальний висновок 2026 року: ШІ частково готовий до написання формальних специфікацій. Роль людини зводиться до формулювання математично точної початкової вимоги, а рутинну роботу виконує агент.

2. Було проаналізовано рефлексії студентів щодо майбутнього індустрії розробки ПЗ та ролі розробника і ШІ.

У 2025 році студенти зазначали зниження вартості написання коду і вважали формальну специфікацію просто "корисною навичкою" для тестування того, що нагенерував ШІ. Інженерія вимог розглядалася як убезпечення від ШІ-галюцинацій. У звітах 2026 року з'явилося усвідомлення зміни парадигми. Оскільки ШІ-агенти тепер здатні самостійно писати складний прикладний код і виправляти його, популярний раніше "Prompt Engineering" визнано менш перспективним. Деякі студенти 2026 року зазначають, що в найближчі 5 років професія "програміст" значною мірою трансформується у професію "Інженер-специфікатор" (Specification Engineer). Головним завданням людини буде написання контрактів в математично-орієнтованій нотації. Коли контракт написаний людиною і є формально доведеним, наступне генерування коду, його оптимізація і верифікація стають автоматизованим процесом, який виконують ШІ-агенти.

3. Загальні висновки у порівнянні 2025 та 2026 року.

Порівняльний аналіз студентських звітів курсу "формальні методи розробки програмних систем" за 2025 та 2026 роки показує не просто

еволюцію інструментів, а якісний перехід у методології розробки. За один рік студенти перейшли від виправлення згенерованих III синтаксичних помилок до побудови автономних довідних агентів. Мови типу Dafny вийшли на перший план, вони краще пристосовані для машинної обробки. Головний висновок студентів 2026 року полягає в тому, що формальна специфікація більше не є "нішовим академічним предметом", а еволюціонує у напрямі промислового стандарту у поєднанні з використанням III.

Список використаних джерел

1. Reducing Verification Costs through Practical Formal Methods: A Survey - KnowledgeBytes, http://www.knowledgebytes.net/downloads/Practical_Formal_Methods_A_Survey_2011_v1.pdf (дата звернення: 24.08.2026).
2. Formal Verification Copilot Market Research Report 2034 - Dataintelo, <https://dataintelo.com/report/formal-verification-copilot-market> (дата звернення: 24.08.2026).
3. AI | 2025 Stack Overflow Developer Survey, <https://survey.stackoverflow.co/2025/ai> (дата звернення: 24.08.2026).
4. 84% of developers use AI, yet most don't trust it! - ShiftMag, <https://shiftmag.dev/stack-overflow-survey-2025-ai-5653/> (дата звернення: 24.08.2026).
5. The Dafny Programming and Verification Language. <https://dafny.org/> (дата звернення: 24.08.2026).
6. Learn TLA+. <https://learntla.com/> (дата звернення: 24.08.2026).

Шишацька Олена Володимирівна, к.ф.-м.н., доцент

ІНТЕГРАЦІЯ ТЕОРЕТИЧНОГО ТА ПАРАДИГМАЛЬНОГО ПІДХОДІВ У ПІДГОТОВЦІ БАКАЛАВРІВ ІНФОРМАТИКИ

У статті висвітлено досвід поєднання теоретичного та парадигмального підходів у підготовці бакалаврів комп'ютерних наук (ОПП «Інформатика») в КНУ імені Тараса Шевченка. На прикладі вибіркової дисципліни «Парадигми та технології програмування» продемонстровано методичні можливості практичної реалізації теоретичного апарату обов'язкового курсу «Теорія програмування» (формалізація мов, алгебра програм, коректність) у функціональному, логічному й об'єктному стилях програмування. Проаналізовано зміст курсів, зв'язок між ними, форми інтеграції та результати навчання. Наведено таблиці відповідності понять і зіставлення результатів обох дисциплін.

Ключові слова: теорія програмування, парадигми програмування, інтеграція підходів, бакалавр інформатики, компетентності, результати навчання, формалізація програм, коректність програм.

The article highlights the experience of combining theoretical and paradigmatic approaches in training Bachelor students of Computer Science (EPP "Informatics") at Taras Shevchenko National University of Kyiv. Using the elective course 'Programming Paradigms and Technologies' as a case study, the paper explores methodological approaches to putting the theoretical framework of the mandatory 'Theory of Programming' course (language formalization, program algebra, correctness) into practice within functional, logic, and object-oriented programming paradigms. The paper analyzes course contents, the connection between them, forms of integration, and learning outcomes, providing tables of conceptual correspondence and a comparison of the results of both disciplines.

Keywords: theory of programming, programming paradigms, integration of approaches, Bachelor of Informatics, competences, learning outcomes, program formalization, program correctness.

Вступ

Підготовка фахівців у галузі інформаційних технологій – одна з найдинамічніших і водночас найбільш суперечливих сфер сучасної університетської освіти. З одного боку, практикоорієнтованість і швидка зміна технологічного ландшафту вимагають від випускника здатності

негайно вирішувати прикладні задачі. З іншого – фундаментальна природа програмування як наукової дисципліни вимагає від фахівця глибокого розуміння теоретичних основ, без яких неможлива ані обґрунтована архітектура складних систем, ані критична оцінка нових обчислювальних парадигм.

Ця суперечність добре відома у світовій педагогіці комп'ютерних наук. Ще наприкінці ХХ ст. Е. Дейкстра та Ч. Хоар наголошували, що програмування є математичною дисципліною і що без відповідної теоретичної культури фахівець приречений на помилки, яких принципово не може виявити інтуїтивне мислення [5]. Водночас ізольоване викладання теорії без чіткого зв'язку з практикою перетворює формальні курси на незрозумілі та демотивуючі для студентів.

У сучасному контексті ця проблема набуває нового виміру: поряд із класичними парадигмами програмування – об'єктно-орієнтованою, функціональною, логічною – активно розвиваються реактивне, агентне, квантове програмування. Фахівець в галузі комп'ютерних наук, що не отримав системного уявлення про природу парадигм і їхній теоретичний фундамент, виявляється беззахисним перед цим різноманіттям: він може «вивчити» чергову мову, але не здатний обрати адекватну парадигму для задачі і обґрунтувати свій вибір.

В освітньо-професійній програмі «Інформатика» обов'язковий курс «Теорія програмування» (ТП) забезпечує фундаментальну математично-методологічну культуру мислення, яка знаходить пряме прикладне розгортання в дисциплінах спеціалізації. Зокрема, у вибіркового блоці кафедри теорії та технології програмування дисципліна «Парадигми та технології програмування» (ПТП) слугує показовим майданчиком для систематичної матеріалізації категоріального апарату теорії.

Мета статті: висвітлення досвіду та обґрунтування методичної інтеграції теоретичного та парадигмального підходів у підготовці бакалаврів інформатики. Завдання статті: охарактеризувати сутність і змістове наповнення кожного з підходів; виявити структурно-логічні зв'язки між дисциплінами на рівні конкретних понять і конструкцій; описати методичну реалізацію інтеграції в навчальному процесі; проаналізувати результати навчання та компетентності, що формуються на перетині двох підходів.

Теоретичний підхід у підготовці бакалаврів: сутність і роль

Дисципліна «Теорія програмування» як обов'язкова складова освітньо-професійної програми «Інформатика» викладається у 5-му семестрі третього курсу бакалаврату в обсязі 4 кредитів ЄКТС (120 годин): 28 годин лекцій, 28

годин практичних занять та 62 години самостійної роботи [1]. Курс структурований у дві змістовні частини: перша охоплює формалізацію мови програмування та основні програмні поняття, друга – синтаксис і семантику програм, рекурсивні програми та огляд формальних підходів до проектування програмних систем.

Концептуальна основа курсу – авторський навчальний посібник [2], методологічна база якого полягає в наступному твердженні: суть теорії програмування – це категоріальний (поняттєвий) аналіз процесу програмування. Поняття визначаються у єдності двох моментів: змісту (інтенціоналу) та обсягу (екстенціоналу). Тому для теорії програмування найважливішим є не конкретна програма, а відношення між програмними поняттями – логічна, інтенціональна складова [2].

Теоретичний підхід ґрунтується на трьох фундаментальних принципах. По-перше, принцип єдності теорії та практики: розробка теорії ґрунтується на науковому дослідженні та аналізі практики, після чого наукові теорії втілюються в практику через відповідні технології. Цей цикл – практика → наукове дослідження → теорія → технологія → практика – не лише наукова схема, а й дидактичний орієнтир [2]. По-друге, принцип сутнісного підходу: на відміну від феноменологічного підходу, що описує зовнішні прояви процесів, сутнісний підхід спрямований на розкриття їх внутрішніх причин і зв'язків. По-третє, принцип від абстрактного до конкретного: побудова теорії здійснюється від простих абстракцій до складних програмних систем, що відповідає основним методам побудови програм – систематичному, зверху-вниз, структурному [2].

Практична реалізація цих принципів у курсі починається з формалізації простої мови програмування SIPL (Simple Imperative Programming Language). Мова SIPL – варіант класичних мов структурного програмування та охоплює присвоєння, послідовне виконання, розгалуження й цикл. Для SIPL у курсі будується повна формалізація:

- 1) синтаксис мови задається за допомогою форми Бекуса–Наура (БНФ); розглядаються дерева синтаксичного виводу, питання однозначності граматик, пріоритет операцій;

- 2) семантика мови задається через алгебру даних і функціональні алгебри; операціями алгебр є композиції, що формалізують засоби конструювання програм: суперпозиція (S_n), присвоєння (AS_x), послідовне виконання ($\bullet$), умовний оператор (IF), цикл (WH);

- 3) для кожної програми будується семантичний терм – її образ в алгебрі програм A_Prog ;

4) на основі побудованих формалізацій доводиться часткова та повна коректність програм.

Ключовий результат цього аналізу – усвідомлення студентом того, що «без наявності формальної семантики та синтаксису ми не могли б навіть говорити про доведення різних властивостей програм та, зокрема, їх коректності. На інтуїтивному рівні розуміння програм може бути лише інтуїтивне обґрунтування властивостей програм» [2]. Це твердження методологічно принципове: воно переводить поняття коректності з рівня інтуїції на рівень строгої математичної теорії.

Центральний математичний результат другої частини курсу: теорема про найменшу нерухому точку – фундаментальний результат, що лежить в основі семантики рекурсивних програм. Студент опановує поняття повних впорядкованих множин, монотонних і неперервних операторів, ієрархію граматики Хомського, автоматні формалізми сприйняття мов, контекстно-вільні граматики та методи семантичного аналізу – аксіоматичну, денотативну, композиційну семантики [1].

Схема формалізації мови SIPL узагальнюється до загальної схеми: мова L задається як трійка (Synt, Sem, interpretation), де Synt – опис синтаксичного аспекту, Sem – опис семантичного аспекту (смислу програм), а interpretation: $\text{Synt} \rightarrow \text{Sem}$ – інтерпретація, що кожній програмі зіставляє її смисл. Саме ця схема, оволодівши якою студент дійсно розуміє, що таке мова програмування, стає містком до глибокого розуміння будь-якої парадигми.

Важлива педагогічна особливість курсу – форма проведення практичних занять. Зважаючи на складність матеріалу та необхідність формування навичок математичного мислення, практичні заняття передбачають обговорення розв'язань задач з залученням студентів до дискусії. У зв'язку з цим практичні заняття доцільно проводити з академічною підгрупою чисельністю до 15 осіб. Такий формат дає змогу реалізувати сократівський метод: задачі не лише розв'язуються, а й обговорюються, аналізуються з різних боків, обґрунтовуються.

Базою для опанування ТП є попередні математичні дисципліни циклу загальної та професійної підготовки («Дискретна математика», «Математична логіка», «Теорія алгоритмів»). Своєю чергою, категоріальний апарат курсу ТП виконує інтегративну роль на завершальному етапі бакалаврату: він забезпечує формально-семантичну основу для виконання курсових та кваліфікаційних робіт, дисциплін фахового вибору комп'ютерного спрямування (зокрема парадигмальних та архітектурних спецкурсів), а також створює наукове підґрунтя для дисциплін другого

(магістерського) рівня вищої освіти, пов'язаних із формальними методами верифікації та прикладною логікою [2].

Парадигмальний підхід: багатовимірне мислення програміста

Поняття «парадигма програмування» введено Робертом Флойдом у 1978 р. та розвинуто Томасом Куном у контексті наукових революцій. У програмуванні парадигма – спосіб концептуалізації обчислень, що визначає, як програміст описує задачу, які абстракції він використовує і як вони відображаються на обчислювальний процес. Різні парадигми не конкуруючі – вони є взаємодоповнювальними, кожна з яких найкраще підходить для певного класу задач.

Дисципліна «Парадигми та технології програмування» – вибіркова в рамках блоку «Теорія та технологія програмування» і викладається у 6-му семестрі третього курсу в обсязі 4 кредитів ЄКТС (120 годин): 28 годин лекцій, 14 годин лабораторних робіт та 76 годин самостійної роботи [3]. Курс завершується екзаменом і передбачає три змістові частини та три контрольні роботи у форматі тестів.

Мета курсу – засвоєння базових знань щодо використання різних парадигм програмування: функціональної, логічної, декларативної, об'єктно-орієнтованої; оволодіння навичками проєктування та розробки програмних систем із застосуванням різноманітних парадигм з відповідними структурами даних, механізмами управління та технологіями [3].

Частина 1. Функціональна парадигма (Haskell)

Теоретичною базою функціонального програмування є лямбда-числення Алонзо Черча (1936). Функціональне програмування – стиль, у якому програма є набором математичних функцій без побічних ефектів. Мова Haskell, обрана для вивчення цієї парадигми, є «чистою» функціональною мовою зі строгою статичною типізацією та лінівими обчисленнями. У курсі розглядаються: типізація в Haskell, визначення функцій та механізм карінгу (часткового застосування); визначення функцій кількома рівняннями та зіставлення зі зразком (pattern matching); рекурсивні функції та теорема про нерухому точку; поліморфні типи та параметричний поліморфізм; функції вищих порядків, лямбда-функції, функції `curry` та `uncurry`; лінійні обчислення, нескінченні списки та `list comprehension`; монадні обчислення, вводу-виводу та обробка виключень (монада `Maybe`) [3]. Ключова дидактична ідея: студент уперше бачить програму не як послідовність команд, що змінюють стан машини, а як систему математичних рівнянь. Це радикально змінює спосіб мислення і готує ґрунт для більш глибокого розуміння семантики, що вивчалася в ТП.

Частина 2. Логічна парадигма (Prolog)

Логічне програмування ґрунтується на формальній логіці предикатів першого порядку: програма – набір аксіом і правил, а виконання – пошук доведення цільового твердження. Мова Prolog, реалізована у варіанті SWI-Prolog, є найпоширенішою мовою логічного програмування.

У курсі вивчаються: синтаксис «чистого» Прологу – факти, правила та запити; декларативна (логічна) та операційна семантики Prolog-програм; метод резолюцій, алгоритм уніфікації, стратегія пошуку та дерево виводу; особливості Prolog-процесорів – механізм відкатів (backtracking) та керування ним; техніки логічного програмування – робота зі списками, моделювання циклів через рекурсію, стеки та черги; логіка в експертних системах – пряма та обернена стратегії пошуку відповіді, подання знань у вигляді правил; експертні системи з поясненнями – відповіді на питання «Як?» та «Навіщо?», робота в умовах неповної визначеності [3].

Методично ця частина найбільш безпосередньо пов'язана з теорією програмування: алгоритм уніфікації, що є операційним ядром Prolog-процесора, детально розглянутий у курсі математичної логіки та зустрічається в ТП при вивченні резолютивних методів. Студент природньо переходить від абстрактного формалізму до живої мови.

Частина 3. Декларативні засоби в ООП (Java / .NET)

Третя частина курсу поширює принципи декларативності на найбільш поширену об'єктно-орієнтовану парадигму. Студенти вивчають: метадані та анотації – рефлексія об'єктів, декларативний стиль у Java, створення та використання власних анотацій; декларативні фреймворки на основі анотацій для IoC (Inversion of Control) та DI (Dependency Injection); сервісно-орієнтована архітектура (SOA) – принципи побудови розподілених систем та еволюція стандартів від SOAP до REST; декларативний опис мережевих інтерфейсів – порівняння контрактного (WSDL) та ресурсного (OpenAPI/Swagger) підходів [3].

Ця частина – приклад того, як декларативні ідеї функціонального та логічного програмування проникають у мейнстрімне ООП. Студент бачить, що «парадигмальна чистота» не є вимогою реального програмування – важливіше розуміти, який саме декларативне виражальний засіб обирається і чому.

Попередні вимоги до ПТП прямо передбачають компетентності, здобуті у ТП та суміжних курсах: знання загальних понять мов програмування (синтаксис, семантика), основ теорії алгоритмів (рекурсивні функції та обчислення), математичної логіки (резольвента, резолютивний

вивід), а також уміння описувати на формальних мовах 1-го порядку твердження щодо предметних областей [3].

Структурно-логічний зв'язок між підходами

Послідовне розміщення дисциплін – ТП у 5-му семестрі, ПТП у 6-му – методично вмотивована архітектура навчального процесу. Теоретичний курс виступає концептуальним фундаментом, на якому парадигмальний курс зводить конкретні «будівлі» обчислювальних стилів.

Ключовим методологічним принципом, що об'єднує обидва курси, є ідея композиційного програмування – підходу, запропонованого В. Н. Редьком і розвиненого у Київській науковій школі [2]. Програма розуміється не як набір інструкцій для машини, а як алгебраїчний вираз у певній програмній алгебрі, де базові функції і правила їх композиції повністю визначають семантику мови. Цей погляд універсальний: і функціональна програма на Haskell, і логічна програма на Prolog, і анотований Java-клас – різні реалізації одного й того ж принципу – декларативного опису обчислень через систему взаємодіючих виразів. Зв'язок між двома підходами виявляється на рівні конкретних понять і конструкцій. Розглянемо ці зв'язки детально.

Теорема про нерухому точку, що є центральним результатом теорії рекурсивних програм, безпосередньо пояснює семантику рекурсивних функцій у Haskell. Функція $\text{fix} :: (a \rightarrow a) \rightarrow a$ – програмне втілення конструктивного методу знаходження найменшої нерухомої точки; лінійні обчислення Haskell забезпечують коректність цього методу навіть для нескінченних структур даних.

Часткова та повна коректність, що доводяться в ТП для програм мови SIPL, знаходять відображення у системі типів Haskell та монадному програмуванні. Монада Maybe – програмний аналог часткових функцій – явно виражає можливість невизначеності результату – саме те, що в ТП записується через позначку $\text{fs}(st) \uparrow$ (функція не визначена на стані st). Монада Either узагальнює це до обробки помилок.

Найбільш прямий зв'язок між курсами виявляється у логічній парадигмі. Алгоритм уніфікації – операційне ядром Prolog-процесора – конкретизує абстрактне поняття «розв'язання системи рівнянь у термовій алгебрі», з яким студенти ознайомлюються у ТП через формалізм алгебр. Механізм відкатів (backtracking) – реалізація стратегії пошуку в дереві виводу – поняття, безпосередньо пов'язане з методом резолюцій. Факти і правила Prolog – це аксіоми і правила виводу, що вивчалися у математичній логіці та застосовувалися у ТП.

Предметні області і мови специфікацій, що розглядаються у другій частині ТП, набувають конкретного програмного втілення в логіці експертних систем: база знань Prolog є формальним описом предметної області мовою логіки предикатів першого порядку. Механізм пояснень (відповіді на «Як?» та «Навіщо?») реалізує аксіоматичний підхід до верифікації висновків.

Поняття алгебри програм і рівнів абстракції програмних систем, що розглядаються у ТП, практично втілено у третій частині ПТП. Анотації Java і атрибути .NET – декларативний опис метарівня програмної системи; принцип IoC реалізує інверсію залежностей, що є архітектурним аналогом переходу від конкретної до абстрактної алгебри; OpenAPI-специфікація – формальний контракт у дусі мов специфікацій, що вивчалися в ТП (таблиця 1).

Таблиця 1. Відповідність понять теоретичного та парадигмального курсів

Поняття / курс ТП	Парадигма / курс ПТП	Конкретна конструкція або концепція
Теорема про найменшу нерухому точку	Функціональна (Haskell)	Рекурсивні функції, fix-комбінатор, лінійні обчислення
Часткові функції: $fs(st) \uparrow / fs(st) \downarrow$	Функціональна (Haskell)	Монади Maybe та Either; обробка виключень
Поліморфні функції та алгебри	Функціональна (Haskell)	Параметричний поліморфізм, класи типів (type classes)
Уніфікація, резолютивний вивід	Логічна (Prolog)	Механізм зіставлення (matching), запити до БЗ
Стратегія пошуку, дерево виводу	Логічна (Prolog)	Backtracking, керування відкатами (cut)
Предметні області, мови специфікацій	Логічна (Prolog)	Бази знань, ЕС, механізм пояснень («Як?»/«Навіщо?»)
Алгебра програм, метарівень систем	Декларативна в ООП	Анотації, рефлексія, IoC/DI-фреймворки
Формальний контракт / мови специфікацій	Декларативна в ООП	OpenAPI/Swagger, WSDL-специфікації
Рівні абстракції програмних систем	Усі три парадигми	Вибір парадигми залежно від рівня та класу задач

Наведена таблиця не вичерпний перелік, а ілюстрація загального принципу: кожне поняття, введене в ТП як абстрактний теоретичний об'єкт, знаходить конкретне програмне втілення в одній або кількох парадигмах, що

вивчаються в ПТП. Це забезпечує двоспрямований навчальний ефект: теоретичний курс готує студента до парадигмального, а парадигмальний – поглиблює і конкретизує засвоєне в теоретичному. Принципово важливо, що цей зв'язок не штучно сконструйований, а органічно впливає зі спільної математичної природи обох дисциплін: обидві спираються на формальну логіку, теорію множин і алгебраїчні структури як свій фундамент. Студент, що добре засвоїв ТП, приходить на курс ПТП із готовим понятійним апаратом – і не просто «вчить нові мови», а впізнає в нових конструкціях уже відомі теоретичні об'єкти.

Методична реалізація інтеграції в навчальному процесі

Інтеграція теоретичного та парадигмального підходів реалізується через систему взаємопов'язаних методичних механізмів, що охоплюють структуру навчального плану, форми та методи навчання, організацію самостійної роботи та систему оцінювання.

Наступність і спільна математична база

Обидва курси спираються на спільний математичний фундамент – дискретну математику, математичну логіку та теорію алгоритмів. Ця спільна основа унеможливорює сприйняття теоретичного і парадигмального курсів як розрізнених. Студент, що навчився формалізувати предметну область мовами логіки першого порядку та встановлювати істинність формул, ніколи не буде розглядати факти і правила Prolog як «щось нове» – він впізнає в них добре знайомі синтаксичні та семантичні структури.

Вхідні вимоги до ПТП прямо передбачають знання і вміння, сформовані у ТП: загальні поняття мов програмування (синтаксис, семантика, об'єктно-орієнтований стиль), основи теорії алгоритмів (рекурсивні функції та рекурсивні обчислення), основні поняття математичної логіки (зокрема поняття резольвенти та резолютивного виводу), вміння описувати на формальних мовах 1-го порядку твердження щодо предметних областей та властивостей програм [3].

Форми навчання і методи викладання

Обидва курси поєднують лекційну та практичну (лабораторну) складові, проте з різним акцентом і методичним наповненням.

У курсі ТП лекції присвячені систематичному викладенню теоретичного апарату – від формалізації SIPL до семантики рекурсивних програм і сучасних методів розробки. Практичні заняття мають принципово діалогічний характер: студенти не просто відтворюють зразки, а розв'язують задачі формалізації, будують семантичні терми, доводять коректність

програм – і захищають свої рішення перед аудиторією. Проведення практик з половиною групи є необхідною умовою такого діалогу [1].

Проектне завдання – інтегративний компонент курсу ТП: воно об'єднує знання з обох частин курсу (формалізація і семантика) і вимагає від студента самостійного дослідницького мислення. Студент може замінити проектне завдання на здачу відповідного сертифіката з курсу, попередньо узгодженого з викладачем [1].

У курсі ПТП лабораторні роботи організовані за принципом наростаючої складності: спочатку студент освоює базові конструкції мови (типи, функції, рекурсія у Haskell; факти та запити в Prolog; анотації в Java), потім переходить до повноцінних програмних рішень – реалізація алгоритмів на списках, побудова міні-ЕС, розробка власних декларативних фреймворків. Захист лабораторних робіт включає усне обговорення прийнятих рішень, що безпосередньо корелює з теоретичним умінням обґрунтовувати властивості програм [3].

Студент, що не виконав вчасно лабораторну роботу, втрачає один бал за кожен тиждень затримки [3] – це формує відповідальність за терміни, що є самостійним результатом навчання (РН4.1). Водночас, якщо роботу виконано неякісно, викладач має право її не зарахувати або знизити бали – формується відповідальність за якість.

Система оцінювання як інструмент інтеграції

Системи оцінювання обох курсів побудовані за компетентнісним принципом і передбачають чотири виміри результатів навчання: знання (РН1), вміння (РН2), комунікацію (РН3) та автономність і відповідальність (РН4) [1, 3]. Подана нижче таблиця 2 зіставляє ці результати для обох курсів.

Таблиця 2. Зіставлення результатів навчання
дисциплін ТП та ПТП

Вимір РН	Результати навчання – ТП (5 сем.)	Результати навчання – ПТП (6 сем.)
Знати (РН1)	РН1.1: основні поняття програмування РН1.2: методи формалізації мов РН1.3: методи моделювання предметних областей	РН1.1: суть і засоби функціональної парадигми (Haskell) РН1.2: суть і засоби логічної парадигми (Prolog) РН1.3: декларативні засоби в ООП (.NET/Java)

Вимір РН	Результати навчання – ТП (5 сем.)	Результати навчання – ПТП (6 сем.)
Вміти (РН2)	РН2.1: формалізувати мови, моделювати предметні області, аналізувати програми	РН2.1: застосовувати функціональний стиль (Haskell) РН2.2: застосовувати логічне програмування (SWI-Prolog) РН2.3: використовувати декларативні засоби ООП
Комунікація (РН3)	РН3.1: обґрунтовувати власний погляд на задачу; спілкуватися щодо специфікацій та програм	(формується при захисті лабораторних, аналізі та вибору парадигми)
Автономність (РН4)	РН4.1: самостійно організовувати роботу для досягнення результату	РН4.1: самостійно організовувати роботу, нести відповідальність за якість і терміни
Форми оцінювання	2 контр. роботи + проектне завдання + домашні завдання + поточне оцінювання + іспит (40 б.)	3 тести + захист 3-х блоків лабораторних (Haskell, Prolog, Java/.NET) + дотримання графіку + екзамен

В обох курсах вимір РН4 (автономність і відповідальність) оцінюється через дотримання студентом власного графіку роботи – а не лише через правильність виконаних завдань. Це цілеспрямоване формування відповідальності як особистісної якості фахівця.

У курсі ТП іспит являє собою письмову роботу з чотирма питаннями: два теоретичних (РН1) та два практичних (РН1.3, РН2.1). Критерії оцінювання кожного питання включають повноту розкриття (1–4 бали), логіку викладення (2 бали) та аналітичні міркування (1–4 бали) [1] – структура, що безпосередньо відображає теоретичний характер курсу і вимагає від студента не відтворення, а осмислення матеріалу.

Роль самостійної роботи

Частка самостійної роботи значна в обох курсах: 62 години в ТП та 76 годин у ПТП. Ця диспропорція на користь ПТП є методично обґрунтованою: якщо в ТП самостійна робота супроводжує та заглиблює матеріал лекцій і практик, то в ПТП вона є основним середовищем для вправлення у

парадигмальному мисленні – написання та налагодження програм на Haskell, Prolog, Java, відпрацювання патернів кожної парадигми.

Тематичний план самостійної роботи в ТП передбачає конкретні завдання по кожній темі: аналіз визначень мов програмування, побудова граматик та синтаксичних діаграм, доведення властивостей алгебр і функцій, побудова рекурсивних програм та дослідження їх властивостей, концептуальний аналіз підходів до формальної специфікації систем та ознайомлення з принципами побудови специфікацій у підходах модельно-орієнтованого (B, Z) та алгебраїчного (RAISE) проектування [1]. Ця деталізація забезпечує змістовну неперервність між аудиторними та позааудиторними формами навчання.

Результати навчання та сформовані компетентності

Інтеграція теоретичного та парадигмального підходів забезпечує формування компетентностей, що виходять за рамки кожного окремого курсу і являють собою якісно нову здатність – системне бачення програмування як наукової і технологічної дисципліни.

Загальні компетентності

Обидва курси спільно розвивають широкий спектр загальних компетентностей. Здатність до абстрактного мислення, аналізу та синтезу формується в ТП через побудову формальних моделей і доведення властивостей програм та в ПТП – через необхідність обирати між принципово різними способами опису однієї й тієї ж задачі. Здатність до математичного формулювання та дослідження дискретних математичних моделей є безпосередньою метою ТП і передумовою ПТП. Здатність обґрунтовувати вибір методів і підходів для розв'язування теоретичних і прикладних задач у галузі комп'ютерних наук формується саме на перетині двох курсів: теорія дає критерії вибору, а парадигмальна практика – досвід застосування [1].

Фахові компетентності

На рівні фахових компетентностей особливо важлива СК18.2 – здатність до алгоритмічного та логічного мислення [3]. Ця компетентність виявляється в кожній з трьох парадигм по-різному: рекурсивне мислення та еквіваленція функцій – у функціональній; логічне виведення та стратегії пошуку – у логічній; декларативний опис залежностей та метапрограмування – в декларативно-об'єктно-орієнтованій. Проходження всіх трьох забезпечує об'ємне, багатовимірне алгоритмічне мислення.

Програмний результат навчання ПРН19.2 – здатність аналізувати, оцінювати і вибирати інструментальні та обчислювальні засоби, парадигми, технології, алгоритмічні і програмні рішення при проектуванні та розробці

програмних систем [1, 3] – є «точкою зустрічі» обох підходів. Він потребує і теоретичного розуміння семантики та властивостей програм, і практичного досвіду роботи з різними парадигмами. Без теоретичного курсу цей результат залишається формальним – студент «знає парадигми», але не вміє обирати між ними аргументовано; без парадигмального курсу теоретик не має досвіду, де застосовувати свій апарат.

Не менш важливим є ПРН18.2 – здатність застосовувати підходи щодо оцінки та забезпечення якості програмного забезпечення [1, 3]. Саме теорія програмування надає для цього математичний апарат (доведення коректності, формальна верифікація), а парадигмальний курс – практичні засоби його реалізації (типові системи Haskell як інструмент статичного запобігання помилок; декларативні специфікації як контракти між компонентами системи).

Спостереження з педагогічної практики

Аналіз навчального процесу та захистів робіт у курсі ПТП показує якісні переваги, що виникають завдяки попередньому опануванню курсу ТП у 5-му семестрі. Спостереження за динамікою засвоєння тем фіксують такі характерні патерни студентського сприйняття:

- при вивченні Haskell вони значно природніше приймають відсутність стану і мутабельності – адже алгебра програм, з якою вони вже знайомі, описує програму як функцію, а не як послідовність змін стану. Поняття монади перестає бути «страшним» – це просто тип-обгортка з двома операціями, що відповідають певним алгебраїчним законам;

- при вивченні Prolog вони значно впевненіше орієнтуються в операційній семантиці: алгоритм уніфікації не є «магією інтерпретатора» – це конкретний алгоритм, схожий на розв'язання системи рівнянь, що вивчався ще в курсі математичної логіки. Механізм відкатів сприймається як природна реалізація пошуку в дереві виводу;

- при вивченні декларативних засобів у Java вони критично підходять до вибору між контрактним і ресурсним підходами до опису інтерфейсів, розуміючи різницю між специфікацією і реалізацією – що є прямим наслідком теоретичного розуміння різниці між синтаксисом і семантикою програм.

Водночас спостерігається і зворотний ефект: парадигмальний курс поглиблює розуміння теоретичних понять. Після практичного знайомства з монадами студент краще розуміє абстрактний зміст часткових функцій; після роботи з Prolog – яскравіше відчуває практичне значення алгоритму уніфікації.

Висновки та перспективи

Інтеграція теоретичного та парадигмального підходів у підготовці бакалаврів інформатики є не методичним прийомом, а концептуальною необхідністю, що випливає зі самої природи програмування як науки і практики. «Теорія програмування» і «Парадигми та технології програмування» є двома взаємодоповнювальними вимірами одного феномену. Перший курс формує поняттєвий фундамент: мову опису, інструменти формалізації, принципи доведення коректності – категоріальний апарат, що дозволяє говорити про програми строгою математичною мовою. Другий реалізує цей фундамент у різних обчислювальних стилях, показуючи, як одні й ті самі теоретичні конструкти – нерухомі точки, уніфікація, алгебри функцій, предметні специфікації – живуть у функціональному, логічному та декларативно-об'єктно-орієнтованому програмуванні.

Ефективна методична інтеграція забезпечується сукупністю умов, виявлених у педагогічній практиці кафедри. По-перше, логічна послідовність розміщення: теоретичний курс у 5-му семестрі передуює парадигмальному у 6-му. По-друге, спільна математична база: обидва курси спираються на дискретну математику, математичну логіку та теорію алгоритмів, що утворює єдине поняттєве поле. По-третє, цілеспрямоване відображення: кожне теоретичне поняття ТП знаходить конкретне програмне втілення в одній або кількох парадигмах ПТП, і цей зв'язок явно артикулюється у навчанні. По-четверте, компетентісно орієнтована система оцінювання, що охоплює всі чотири виміри – знання, вміння, комунікацію та автономність – і є узгодженою між курсами. По-п'яте, діалогічні форми навчання, що вимагають від студента обґрунтування, а не лише відтворення.

Описаний підхід демонструє, що принцип методологічного циклу – практика → наукове дослідження → теорія → технологія → практика [2] – є не лише описом наукового методу, а й продуктивним дидактичним принципом. Студент проживає цей цикл: спочатку зустрічається з практикою програмування у попередніх курсах, потім отримує теоретичні інструменти її аналізу (ТП), потім повертається до практики на якісно новому рівні (ПТП), розуміючи, чому та чи інша технологія працює саме так.

Перспективою подальшого розвитку описаного підходу є розробка міждисциплінарних проєктних завдань, де студент самостійно обирає парадигму та обґрунтовує вибір формальними аргументами, що дає змогу вивести інтеграцію підходів на рівень реального дослідницького мислення.

Підсумовуючи, можна стверджувати: бакалавр інформатики, що пройшов обидва курси в їхньому методичному взаємозв'язку, отримує не

просто набір знань і навичок, а цілісний фаховий світогляд – здатність бачити в кожній програмі теоретичний об'єкт зі строго визначеними властивостями і водночас – результат усвідомленого парадигмального вибору.

Список використаних джерел

1. Шишацька О. В., Криволап А. В. Робоча програма навчальної дисципліни «Теорія програмування» для студентів спеціальності 122 «Комп'ютерні науки», ОПП «Інформатика». – Київ : Київський національний університет імені Тараса Шевченка, 2025. – 18 с.
2. Нікітченко М. С. Теорія програмування : навчальний посібник. – Київ : Видавничо-поліграфічний центр «Київський університет», 2023.
3. Шишацька О. В. Робоча програма навчальної дисципліни «Парадигми та технології програмування» для студентів спеціальності 122 «Комп'ютерні науки», ОПП «Інформатика». – Київ : Київський національний університет імені Тараса Шевченка, 2026. – 20 с.
4. Стандарт вищої освіти за спеціальністю 122 «Комп'ютерні науки» для першого (бакалаврського) рівня вищої освіти / Міністерство освіти і науки України. – Київ, 2019.
5. Hoare C. A. R. An Axiomatic Basis for Computer Programming // Communications of the ACM. – 1969. – Vol. 12, No. 10. – P. 576–580.
6. Floyd R. W. The Paradigms of Programming // Communications of the ACM. – 1979. – Vol. 22, No. 8. – P. 455–460.
7. Редько В. Н. Композиційні бази даних і програмування. – Київ : Наукова думка, 1992.
8. Liskov B. and Guttag J. Abstraction and Specification in Program Development. Cambridge, MA, USA: MIT Press, 1986.

Галавай Олександр Миколайович, магістр комп'ютерних наук, асистент

ІНТЕГРАЦІЯ ІНДУСТРІАЛЬНИХ ПРАКТИК У ВИКЛАДАННЯ DATA MINING: НАВЧАЛЬНИЙ ХАКАТОН ЯК МІСТ ДО ПРОДУКТОВОГО ІТ

Роботу присвячено аналізу досвіду інтеграції індустріальних практик у викладання дисципліни «Основи Data Mining» для студентів четвертого курсу бакалаврату. Описано трикомпонентну модель організації курсу, що поєднує індивідуальні лабораторні роботи, онлайн-курс з продуктової аналітики від компанії Genesis та командний навчальний хакатон, побудований на реалістичному продуктовому кейсі аналізу відтоку користувачів мобільного застосунку. Наведено результати підсумкового анкетування студентів, які засвідчують високий рівень задоволеності форматом. Окреслено напрями подальшого впровадження даного підходу.

Ключові слова: Data Mining, навчальний хакатон, продуктова аналітика, продуктове мислення, практико-орієнтоване навчання.

The work is dedicated to analysing the experience of integrating industrial practices into the teaching of the Fundamentals of Data Mining course for fourth-year bachelor students. A three-component course model is described, combining individual laboratory work, an online product analytics course provided by the Genesis company, and a team-based educational hackathon built around a realistic product case of user churn analysis in a mobile application. The results of the final student survey are presented, indicating a high level of satisfaction with the format and the willingness of most students to recommend it. Directions for further implementation of the approach are outlined.

Keywords: Data Mining, educational hackathon, product analytics, product thinking, practice-oriented learning.

Вступ

В умовах стрімкого розвитку ІТ-індустрії, зокрема напрямку продуктової аналітики, зростає потреба у фахівцях, здатних не лише застосовувати алгоритми інтелектуальної обробки даних, але й інтерпретувати результати в бізнесовому контексті. Дисципліни, пов'язані з аналізом даних, зокрема Data Mining, традиційно орієнтовані на вивчення алгоритмів, моделей та методів обробки даних та часто не враховують специфіку їх застосування у реальному продуктовому середовищі.

Водночас у продуктовому ІТ ключову роль відіграє так зване продуктове мислення, що передбачає формування гіпотез, роботу з метриками, оцінювання впливу рішень на користувацький досвід та бізнес. Таким чином, це створює розрив між академічною підготовкою та потребами індустрії, який потребує нових підходів до організації навчального процесу.

Одним із перспективних напрямків подолання цього розриву є інтеграція індустріальних практик у навчальні курси, зокрема через використання практико-орієнтованих форматів навчання, таких як хакатони [1, 2] та зовнішні освітні ресурси. В цьому контексті особливого значення набуває поєднання класичного курсу Data Mining із елементами продуктової аналітики.

Інтеграція індустріальних практик у викладання дисципліни Data Mining через впровадження навчального хакатону та залучення елементів продуктової аналітики орієнтована на поєднання алгоритмічної підготовки з контекстом прийняття продуктових рішень і формування відповідного мислення, що відрізняє її від традиційних моделей навчання. Оцінювання ефективності такого підходу пов'язане з аналізом залученості студентів, глибини розуміння матеріалу та здатності застосувати методи аналізу даних до реальних продуктових задач.

Особливістю підходу є використання емпіричних даних, отриманих шляхом анкетування студентів, що дозволяє простежити сприйняття дисципліни та рівень сформованості відповідних компетентностей. Запропоноване поєднання навчального хакатону, елементів продуктової аналітики та зовнішніх освітніх ресурсів розглядається як один із можливих шляхів підвищення практичної спрямованості викладання Data Mining в умовах сучасної ІТ-освіти.

Основна частина

Структура дисципліни. Описаний підхід було реалізовано в межах дисципліни «Основи Data Mining», що викладається у сьомому семестрі бакалаврату на кафедрі теорії та технології програмування факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. На дисципліну відводиться 3 кредити ЄКТС, зокрема 20 годин лекцій та 8 годин семінарських занять. У 2025/2026 навчальному році дисципліну вивчали 33 студенти двох академічних груп. Описані компоненти реалізовано в межах самостійної роботи з виконання проєктів, передбаченої чинною робочою програмою дисципліни [3]. За результатами

семестру підготовлено пропозиції щодо безпосереднього закріплення описаного формату в робочій програмі.

Практична складова дисципліни охоплює три компоненти, кожен з яких відповідає окремому типу навчальної діяльності: індивідуальній, самоосвітній та командній. Склад компонентів наведено в таблиці 1.

Таблиця 1. Компоненти практичної складової курсу «Основи Data Mining»

Модуль	Зміст
Базова частина (лабораторна робота)	Індивідуальна лабораторна робота за однією з чотирьох тем: аналіз ринкового кошика, аналіз тональності текстів соціальних мереж, сегментація клієнтів, предиктивна аналітика для кредитного скорингу. Формат передбачає письмовий звіт та усний захист результатів.
Самоосвіта або друга лабораторна робота	На вибір студента: онлайн-курс «Аналітика в продуктовому IT» від компанії Genesis або друга лабораторна робота (аналіз великих даних, тематичне моделювання текстів, аналіз часових рядів, рекомендаційна система).
Хакатон (командна робота)	Командне розв'язання реалістичного продуктового кейсу з аналізу поведінки користувачів мобільного застосунку та прогнозування відтоку, з публічним захистом результатів.

Така структура забезпечує поступовий перехід від індивідуального опанування класичних методів Data Mining до їх командного застосування в умовах, наближених до індустріальних, що узгоджується з принципами емпіричного навчання [4].

Онлайн-курс з продуктової аналітики. Першим індустріальним компонентом дисципліни став онлайн-курс «Аналітика в продуктовому IT», розроблений українською продуктовою IT-компанією Genesis [5]. Курс було інтегровано в складову самоосвіти як альтернативу другій лабораторній роботі, і саме цей варіант обрала більшість студентів. Програма курсу охоплює ролі в продуктивній розробці, базові та похідні продуктові метрики, роботу з воронками, юніт-економіку, а також практичні інструменти аналітики: SQL, Python, засоби візуалізації даних.

Дидактична функція курсу в межах дисципліни полягала у формуванні понятійного апарату продуктової аналітики до проведення хакатону: студенти опановували термінологію метрик, логіку воронок конверсії та підходи до формулювання продуктивних гіпотез, які згодом безпосередньо застосовувалися під час роботи над кейсом. Таким чином, зовнішній освітній ресурс виконував роль мосту між академічним змістом лекційної частини та практичним контекстом командного завдання.

Навчальний хакатон. Центральним елементом інтеграції індустріальних практик став командний навчальний хакатон, проведений в останній тиждень теоретичного навчання. У хакатоні взяли участь 6 команд по 5–6 студентів. Датасет та опис кейсу команди отримали за три доби до початку події, що дало змогу провести попередній розвідувальний аналіз даних; очна фаза тривала близько чотирьох годин і завершилася публічними пітчами команд.

Кейс хакатону було побудовано на реалістичних даних мобільного застосунку з ШІ-асистентом, що надає персоналізовані рекомендації користувачам. Користувач після встановлення застосунку проходить онбординг-опитування, яке збирає інформацію про його вподобання, мотивацію, цілі та спосіб життя, після чого обирає між тижневою підпискою з безкоштовним пробним періодом та річним планом без нього. Завданням команд був аналіз воронки користувача та виявлення чинників відтоку (відмови від підписки): порівняння поведінки постійних користувачів та користувачів, що скасували підписку, обґрунтування відібраних чинників, побудова класифікаційних моделей прогнозування відтоку та формулювання рекомендацій щодо вдосконалення продукту. Робота з кейсом передбачала використання продуктових метрик воронки платежів: конверсії в підписку, конверсії в перший платіж та частки відписок.

Структура роботи команд узгоджувалася з етапами стандартної індустріальної методології аналізу даних CRISP-DM [6]: від розуміння бізнес-задачі та даних через їх підготовку й розвідувальний аналіз до моделювання, оцінювання якості моделей та представлення результатів. Принциповою відмінністю від класичної лабораторної роботи була вимога інтерпретувати технічні результати в термінах продуктових рішень: команди мали не лише побудувати модель класифікації, а й пояснити, які зміни в продукті впливають з виявлених закономірностей.

Оцінювання хакатону було трикомпонентним, що також відображає індустріальні практики оцінювання командної роботи: підсумкова оцінка команди та її учасників формувалася з оцінки журі (60 %), взаємного оцінювання членами команди (32 %) та самооцінювання (8 %). Взаємне оцінювання передбачало оцінювання внеску кожного члена команди та його змістовий опис, що стимулювало рівномірний розподіл роботи та відповідальність перед командою.

Результати анкетування студентів. Після завершення семестру було проведено підсумкове анонімізоване анкетування засобами Google Forms; з 34 отриманих відповідей до аналізу включено 33 відповіді студентів дисципліни. Анкета містила закриті питання з 5-бальною шкалою оцінювання окремих компонентів дисципліни та відкриті питання щодо набутих навичок і пропозицій з удосконалення. Розподіл оцінок за ключовими закритими питаннями наведено на рисунку 1.

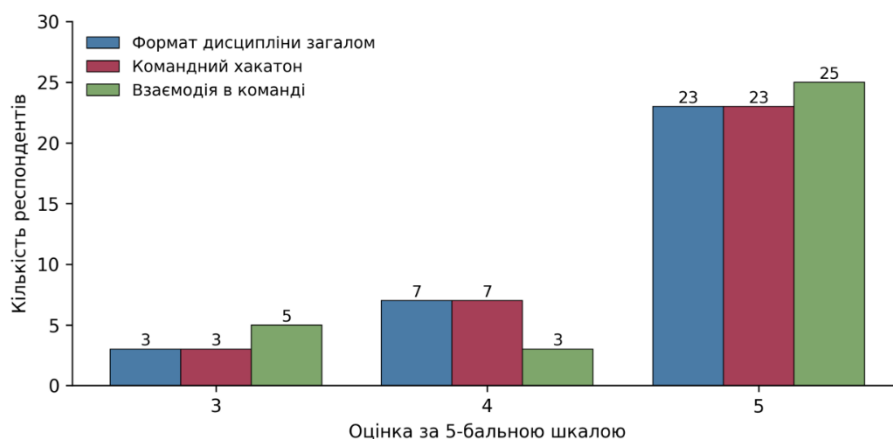


Рис. 1. Розподіл оцінок студентів за результатами підсумкового анкетування

Середня оцінка загального досвіду участі в запропонованому форматі дисципліни (лабораторна робота, онлайн-курс, хакатон) склала 4,61 бала з 5, оцінка досвіду участі в командному хакатоні — також 4,61 бала, комфортності командної взаємодії — 4,61 бала. Оцінок нижче 3 балів не зафіксовано за жодним із питань. На питання про доцільність рекомендувати подібний формат наступним потокам студентів 30 респондентів (91 %) відповіли «так», 2 — «можливо», 1 — «ні».

Аналіз відповідей на відкриті питання дозволяє виокремити кілька повторюваних тем. По-перше, студенти найчастіше відзначали розвиток саме командних та комунікаційних навичок: розподіл ролей і завдань, узгодження спільного рішення, публічну презентацію результатів. Для частини студентів хакатон став першим досвідом очної командної роботи над спільною технічною задачею, що особливо значуще з огляду на тривалий період дистанційного навчання. По-друге, згадувався розвиток аналітичних компетентностей: робота з метриками та їх критичне осмислення, формулювання гіпотез на основі даних, побудова і порівняння класифікаційних моделей, інформативна візуалізація результатів. По-третє,

студенти, які проходили онлайн-курс, відзначали його роль у систематизації уявлень про продуктову аналітику як сферу професійної діяльності та в усвідомленому виборі подальшої кар'єрної траєкторії.

Водночас анкетування виявило й напрями вдосконалення. Частина студентів зауважила, що надання датасету за три дні до події знижує змагальну складову хакатону, наближаючи його до тривалого домашнього завдання. Висловлювалися побажання щодо чіткішої комунікації критеріїв оцінювання, регламентації тривалості виступів, проведення проміжних консультацій під час активної фази, а також щодо вибору дати події з урахуванням близькості екзаменаційної сесії. Окреме зауваження стосувалося структури самого кейсу: частина етапів завдання передбачала послідовне, а не паралельне виконання, через що в окремі моменти робота команди блокувалася на одному учаснику. Це вказує на необхідність проєктувати хакатонні кейси з урахуванням можливості декомпозиції на паралельні підзадачі. Ці спостереження буде враховано в наступних ітераціях дисципліни.

Висновки

Описано та проаналізовано досвід інтеграції індустріальних практик у викладання дисципліни «Основи Data Mining» для студентів четвертого курсу бакалаврату. Запропонована модель поєднує три компоненти: індивідуальну лабораторну роботу з класичних методів Data Mining, онлайн-курс з продуктової аналітики від ІТ-компанії та командний навчальний хакатон на основі реалістичного продуктового кейсу з трикомпонентною системою оцінювання.

Результати підсумкового анкетування засвідчують високий рівень сприйняття підходу студентами: середні оцінки формату дисципліни, хакатону та командної взаємодії склали 4,61 бала з 5, а 91 % респондентів рекомендували б подібний формат наступним потокам. Якісний аналіз відповідей показує, що найбільшу цінність студенти вбачають у розвитку командних, комунікаційних і презентаційних навичок та в набутті досвіду інтерпретації результатів аналізу даних у продуктовому контексті – тобто саме тих компетентностей, які найважче формуються в межах традиційних форм навчання.

Перспективними напрямками розвитку підходу залишаються посилення змагальної складової хакатону, формалізація критеріїв оцінювання та регламенту захистів, запровадження проміжних консультацій під час активної фази, а окремим напрямом подальших досліджень – запровадження

вхідного та вихідного анкетування з єдиною шкалою для кількісного вимірювання динаміки сформованості продуктового мислення протягом семестру.

Отриманий досвід дає підстави розглядати запропонований формат як перспективний напрям організації практичної складової дисципліни: поєднання індивідуальної роботи, індустріального онлайн-курсу та командного хакатону утворює наскрізний навчальний сценарій, у якому знання, здобуті на кожному етапі, безпосередньо застосовуються на наступному, а підсумкова командна подія надає навчання особистісної значущості, що підтверджується як кількісними оцінками, так і якісними відповідями студентів. Водночас слід зазначити обмеження дослідження: коректне порівняння ефективності запропонованого формату з традиційною організацією дисципліни, за якої семестрове оцінювання зводиться до кількох ізольованих індивідуальних лабораторних робіт, потребує аналогічних вимірювань за традиційної схеми — анкетування попередніх потоків або контрольної групи за єдиною методикою. Такі вимірювання не проводилися і становлять завдання подальших досліджень.

З огляду на це доцільним видається перехід описаного підходу від статусу педагогічного експерименту до повноцінної, закріпленої складової навчальної програми дисципліни. Ключовою умовою такого переходу є розширення партнерства з індустрією: залучення продуктивних ІТ-компаній до формування кейсів і надання знеособлених даних, менторської підтримки команд та участі в журі хакатону.

Список використаних джерел

1. Nandi A., Mandernach M. Hackathons as an informal learning platform. Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). New York: ACM, 2016. P. 346–351.
2. Porras J., Knutas A., Ikonen J., Happonen A., Khakurel J., Herala A. Code camps and hackathons in education – literature review. Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS). 2019. P. 7750–7759.
3. Терещенко Я. В. Робоча програма навчальної дисципліни «Основи Data Mining». Київ : КНУ імені Тараса Шевченка, факультет комп'ютерних наук та кібернетики, кафедра теорії та технології програмування, 2025.
4. Kolb D. A. Experiential learning: experience as the source of learning and development. Englewood Cliffs: Prentice-Hall, 1984. 256 p.
5. Аналітика в продуктовому ІТ: онлайн-курс. Genesis. URL: <https://www.genesis-for-univ.com/analytics-course>.
6. Wirth R., Hipp J. CRISP-DM: Towards a standard process model for data mining. Proceedings of the 4th International Conference on the Practical Application of Knowledge Discovery and Data Mining. 2000. P. 29–39.

Шишацький Андрій Вікторович, асистент

ІНТЕГРУВАННЯ ШТУЧНИХ АГЕНТІВ В РОЗРОБКУ В МЕЖАХ КУРСУ «ІНСТРУМЕНТАЛЬНІ СЕРЕДОВИЩА ТА ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ»

Роботу присвячено дослідженню підходів до інтеграції штучних агентів та інструментів генеративного штучного інтелекту в освітній процес підготовки фахівців з інформаційних технологій. На прикладі курсу «Інструментальні середовища та технології програмування» розглядаються методи залучення ШІ для допомоги студентам у проєктуванні баз даних, розробці вебзастосунків на базі ASP.NET Core та контейнеризації. Описано ідеї, які дозволяють не лише оптимізувати написання рутинного коду, а й розвинути у студентів навички критичного аналізу згенерованих рішень та архітектурного мислення.

Ключові слова : штучні агенти, технології програмування, ASP.NET Core, Entity Framework, комп'ютерні науки.

This paper explores approaches to integrating artificial agents and generative AI tools into the educational process for training information technology specialists. Using the "Інструментальні середовища та технології" course as a case study, methods of leveraging AI to assist students with database design, ASP.NET Core web application development, and containerization are examined. The paper outlines strategies that not only streamline the writing of routine code but also foster students' skills in critically analyzing generated solutions and developing architectural thinking.

Keywords: artificial agents, programming technologies, ASP.NET Core, Entity Framework, computer science.

Вступ

Стрімкий розвиток та поширення інструментів на базі генеративного штучного інтелекту докорінно змінюють ландшафт розробки програмного забезпечення. З огляду на ці тенденції, сучасна ІТ-освіта потребує адаптації навчальних програм до нових реалій. Зокрема, підготовка бакалаврів за освітньою програмою «Інформатика» зі спеціальності 122 «Комп'ютерні науки» вимагає оновлених підходів до викладання базових дисциплін. Навчальна дисципліна «Інструментальні середовища та технології програмування» [1] має на меті не лише навчити студентів синтаксису чи роботі з фреймворками, а й сформувати навички проєктування програмних

систем та управління життєвим циклом ПЗ. Метою цієї статті є дослідження та опис ефективних підходів до інтеграції штучних агентів (таких як GitHub Copilot, ChatGPT тощо) у процес вивчення технологій доступу до даних, створення вебзастосунків на платформі ASP.NET Core та контейнеризації застосунків, що дозволить якнайкраще підготувати студентів до реалій сучасного ринку праці.

Основна частина

Процес розробки інформаційних систем у межах курсу розділено на кілька етапів, кожен з яких може бути якісно оптимізований завдяки ІІІ-асистентам. На початкових етапах лабораторних робіт (етапи 1.0 та 2.0) студенти розробляють діаграми прецедентів (Use-case) та проєктують бази даних. Використання генеративного ІІІ на цьому етапі дозволяє студентам швидко валідувати свої моделі та отримувати розгорнуті пояснення щодо нормалізації реляційних баз даних чи відмінностей між підходами Database First та Code First в Entity Framework Core.

Під час реалізації багаторівневої архітектури клієнт-серверних застосунків (робота з ASP.NET MVC, контролерами та представленнями), штучні агенти значно прискорюють написання шаблонного коду. Це дозволяє змістити фокус уваги студентів із запам'ятовування синтаксису (HTML, CSS, JavaScript) на глибоке розуміння архітектури, принципів асинхронної обробки запитів та Dependency Injection. Згідно з дослідженнями впровадження GitHub Copilot у реальних проєктах (дані GitHub Research), використання ІІІ-асистентів кардинально впливає на швидкість розробки та навчання (таблиця 1) [2].

Таблиця 1. Вплив використання ІІІ-асистентів на продуктивність та швидкість виконання завдань

Метрика	Показник (з використанням ІІІ)	Показник (контрольна група без ІІІ)	Результат впливу ІІІ
Час виконання типового завдання	1 год 11 хв	2 год 41 хв	Пришвидшення на 55.8%

Збереження ментальної енергії на рутині	87% розробників підтвердили	-	Значне зниження когнітивного навантаження
Скорочення кривої навчання (Learning Curve)	Швидкість опанування нових технологій зростає на 73%	Базовий час	Суттєво швидша адаптація розробників-початківців

Окрім безпосереднього написання коду, дисципліна охоплює елементи програмної інженерії, методи тестування та безпечного проєктування програмного забезпечення. Штучні агенти виступають ефективними помічниками при генерації Unit-тестів для контролерів (етап 2.4) та створенні конфігураційних файлів Docker для розгортання застосунків. Важливо зазначити, що використання ІІІ не погіршує, а навпаки, за умови правильного ревію, підвищує стабільність розробки (таблиця 2) [3].

Таблиця 2. Вплив ІІІ на якість коду та успішність збірок (на основі корпоративних розгортань, дані Accenture)

Метрика якості	Спостережуваний вплив	Примітка
Успішність збірок (Successful Builds)	Зростання на 84%	Значне зменшення кількості помилок компіляції та синтаксису
Відсоток прийняття коду (Acceptance Rate)	46% пропозицій приймаються без змін	Висока релевантність згенерованих фрагментів
Швидкість ревію коду (Code Review)	Покращення на 15%	Більш стандартизований код легше та швидше перевіряти

Головний педагогічний виклик полягає у запобіганні «сліпому копіюванню». Викладач має трансформувати підхід до оцінювання: замість перевірки обсягу написаного коду, акцент робиться на здатності студента проаналізувати згенерований ШІ код, знайти потенційні вразливості та пояснити логіку роботи автономного рівня ADO.NET чи DbContext. Розподіл навчального часу студента при цьому суттєво змінюється (таблиця 3) [4].

Таблиця 3. Зміна розподілу навчального часу при розробці лабораторного проєкту з використанням ШІ (експертна оцінка)

Вид діяльності в межах проєкту	Традиційний підхід (% часу)	З використанням ШІ (% часу)
Проектування та розробка архітектури	20%	40%
Написання шаблонного (рутинного) коду	50%	15%
Налаштування інфраструктури (Docker, БД)	15%	10%
Тестування, рефакторинг та аналіз	15%	35%

Висновки

Інтегрування штучних агентів у курс «Інструментальні середовища та технології програмування» є необхідним кроком для підготовки конкурентоспроможних фахівців. Використання ШІ дозволяє автоматизувати створення базового коду при роботі з ASP.NET Core, EF Core та клієнтськими технологіями, тим самим вивільняючи час для глибокого вивчення архітектурних патернів, принципів інформаційної безпеки та розгортання розподілених систем. Наведені дані підтверджують, що ШІ значно скорочує криву навчання, пришвидшує виконання рутинних завдань майже на 56% та підвищує загальну успішність збірок. Водночас таке впровадження вимагає від викладачів розробки нових метрик оцінювання, які фокусуються не на

написанні синтаксично правильних рядків, а на розвитку у студентів критичного мислення, навичок проєктування та здатності здійснювати експертне ревію рішень, запропонованих штучним інтелектом.

Список використаних джерел

1. Робоча програма дисципліни Інструментальні середовища та технології програмування.
<https://drive.google.com/file/d/1mX5BSzDQRAUMD4u6giVDggtAM1-T4OVX/view>
1. Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590*. 2023. URL: <https://arxiv.org/abs/2302.06590> (дата звернення: 19.08.2026).
2. How Generative AI is Transforming Software Engineering Management [Електронний ресурс] / Makebot, Accenture. 2025. URL: <https://www.makebot.ai/blog-en/how-generative-ai-is-transforming-software-engineering-management> (дата звернення: 19.08.2026).
3. Generative AI in Computer Science Education: Accelerating Python Learning with ChatGPT / I. McCulloh, P. Rodriguez, S. Kumar та ін. *arXiv preprint arXiv:2505.20329*. 2025. URL: <https://arxiv.org/pdf/2505.20329> (дата звернення: 20.08.2026).
4. Інструментальні середовища та технології програмування. Лабораторний практикум: навчальний посібник / Л.Л. Омелячук, А.О. Свистунов – Київ: 2023. – 160 с.
<http://csc.knu.ua/uk/filer/canonical/1682272182/2242/>

РОЗДІЛ 3. ПРАКТИКО-ОРІЄНТОВАНІ ТЕХНОЛОГІЇ ТА ПРИКЛАДНІ АСПЕКТИ ПІДГОТОВКИ ФАХІВЦІВ ІЗ СИСТЕМНОГО АНАЛІЗУ ТА НАУКИ ПРО ДАНІ

*Лівінська Ганна Володимирівна, к.ф.-м.н, доцент
Розора Ірина Василівна, д. ф.-м.н, професор
Пономарьов Вадим Дмитрович, к.ф.-м.н
Моренець Володимир Ігорович, к.ф.-м.н
Сидоров Микола Володимир-Станіславович, к.ф.-м.н, доцент*

МІЖДИСЦИПЛІНАРНА ІНТЕГРАЦІЯ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ДАНИХ ТА АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ ВЕРИФІКАЦІЇ ЗАДАЧ КЛАСИФІКАЦІЇ

Робота присвячена методології об'єднання візуально-аналітичних підходів та алгоритмічних рішень машинного навчання (Machine Learning). Основна увага приділяється тому, як інтегроване використання візуалізації на кожному етапі розробки моделі класифікації дозволяє виявити приховані закономірності, верифікувати рішення ШІ та мінімізувати ризики, пов'язані з помилковими прогнозами у критично важливих галузях (наприкладі діагностики цукрового діабету). У статті розглядається комплексний підхід до побудови моделі машинного навчання для діагностики діабету. Основна увага приділяється не лише точності алгоритмів, а й методам візуальної верифікації на кожному етапі від очищення даних до оцінки ризиків помилкових прогнозів. Такий підхід дозволяє мінімізувати ефект «чорної скриньки» та підвищити довіру до автоматизованих систем класифікації, зокрема для медичної діагностики.

Ключові слова: візуалізація даних, машинне навчання, класифікація, верифікація моделей МН

The work is devoted to the methodology of combining visual-analytical approaches and algorithmic solutions of machine learning. The main attention is paid to how the integrated use of visualization at each stage of developing a classification model allows to reveal hidden patterns, verify AI solutions and minimize the risks associated with erroneous predictions in critical areas (for example, diabetes diagnostics). The article considers a comprehensive approach to building a machine learning model for diabetes diagnostics. The main attention is

paid not only to the accuracy of algorithms, but also to visual verification methods at each stage from data cleaning to assessing the risks of erroneous predictions. This approach allows to minimize the "black box" effect and increase trust in automated classification systems, in particular for medical diagnostics.

Keywords: data visualization, machine learning, classification, verification of ML models

1. Вступ

Розвиток штучного інтелекту характеризується стрімким ускладненням архітектур алгоритмів. Однак, паралельно із ростом обчислювальної потужності, постає фундаментальна проблема, пов'язана з втратою прозорості прийняття рішень, відома як проблема «чорної скриньки». У критично важливих галузях, таких як медицина, використанні моделей машинного навчання (МН-моделі) без візуального контролю створює високі ризики, оскільки розробник або лікар бачать лише результат (прогноз), не розуміючи логіки, що стоїть за алгоритмом прийняття рішення. Для багатьох випадків, зокрема для медичних застосувань МН-моделей, критично важливо розуміти, на основі яких факторів та співвідношень модель приймає рішення та які чинники мають найбільший вплив.

Концепція інтеграції когнітивних наук, візуальної аналітики та машинного навчання сьогодні постає як критично важливий методологічний стандарт. Необхідність такого поєднання зумовлена кількома факторами:

1. Доповнення обчислювальної потужності: Алгоритми здатні обробляти величезні масиви багатовимірних даних та знаходити математичні закономірності, які неможливо виявити вручну. Проте вони позбавлені контекстного розуміння природи даних.

2. Використання когнітивного потенціалу: Людський мозок володіє унікальною еволюційною здатністю до миттєвого розпізнавання образів, виявлення аномалій та інтерпретації складних візуальних стимулів, що виходять за межі чистої статистики.

3. Верифікація логіки: Візуалізація виступає мостом, який дозволяє експерту переконатися, що модель базується на реальних показниках та зв'язках, а не на випадкових кореляціях чи артефактах збору даних.

4. Інструментарій для оптимізації та коригування: Візуальні методи є незамінними для оцінки якості самої моделі та її тонкого налаштування. Наприклад, використання кривих Precision-Recall або аналіз розподілу ймовірностей дозволяє досліднику свідомо коригувати поріг класифікації. Це дає змогу гнучко адаптувати модель під конкретну медичну задачу, зокрема

мінімізувати пропуски захворювань (збільшення Recall) або уникнути хибних діагнозів (збільшення Precision), що неможливо зробити ефективно, спираючись лише на усереднені метрики на кшталт Accuracy.

При побудові ML-моделей для розв'язанні задач класифікації, зокрема у діагностиці критичних захворювань в медицині, ціна помилки виходить далеко за межі статистичних метрик. Помилки класифікації призводять до таких ризиків:

- Етичні ризики: Хибнопозитивний результат викликає необґрунтований стрес у пацієнта та призначення зайвого лікування, що може мати побічні ефекти.

- Безпеківі ризики: Хибнонегативний результат (пропуск хвороби) може призвести до прогресування захворювання, критичних ускладнень або смерті через відсутність своєчасної терапії.

- Фінансові втрати: Неефективне використання ресурсів медичних закладів на додаткові обстеження та лікування ускладнень, яких можна було запобігти.

Дослідник має зважати на наявність можливих ризиків для конкретної задачі, визначити та оцінити вартість помилки класифікації того чи іншого роду та будувати модель з урахуванням цих деталей, а не покладаючись на стандартні установки. Саме тому візуальна верифікація виступає не просто як допоміжний інструмент, а як необхідний фільтр безпеки та інструмент точного налаштування, що мінімізує ризики помилкових прогнозів у системах підтримки прийняття рішень.

Таким чином, інтеграція методів візуалізації на кожному етапі розробки дозволяє перетворити «чорну скриньку» на прозору систему. Це забезпечує поєднання математичної точності обчислень із фаховою інтуїцією та досвідом людини.

Проблема інтерпретованості моделей (eXplainable AI, XAI) активно досліджується в останнє десятиліття. У фундаментальних роботах [1], [2] було закладено математичні основи локальної інтерпретації прогнозів. В багатьох роботах підкреслено, що самих лише математичних значень недостатньо для прийняття рішення, необхідне інтерактивне середовище, яке дозволяє користувачу «спілкуватися» з даними (див., наприклад, [3], [4]).

У медичній сфері питання верифікації моделей розглядалися в численних працях, присвячених аналізу діабетичних даних, онкологічних даних, тощо. Автори підкреслюють, що наявність пропущених значень та біологічних викидів у таких наборах даних часто призводить до викривлення

результатів, якщо не використовувати методи візуального розвідувального аналізу (див., напр., [4]).

Метою цієї роботи є демонстрація важливості комплексного підходу до розробки та верифікації моделі класифікації на прикладі задачі діагностики цукрового діабету на базі відкритого датасету Pima Indians Diabetes Dataset. Показано, як візуальні методи допомагають не лише зрозуміти модель, а й покращити її якість на кожному етапі. Зауважимо, що в даній роботі ми розглядаємо датасет, який містить значення ознак, виражені кількісними змінними. Для роботи з якісними змінними використовуються дещо інші діагностичні діаграми, проте візуальний супровід при побудові моделі також є надзвичайно важливим.

2. Візуалізація як фундамент підготовки даних

Перш ніж приступати до навчання моделі класифікації, необхідно провести ретельний аудит даних. У цьому розділі ми розглянемо, як візуальний аналіз допомагає виявити приховані вади у структурі даних, що можуть критично вплинути на надійність прогнозу. На відміну від сухої статистики, візуалізація на цьому етапі дає фундаментальні ідеї для подальшої обробки та запобігає навчанню моделі на некоректних даних.

2.1. Датасет 'PimaIndiansDiabetes2'

Для демонстрації методології обрано класичний набір даних Національного інституту діабету, хвороб травлення та нирок (NIDDK). Вибірка включає 768 пацієнток (жінок індіанського походження племені Піма віком від 21 року) і містить 8 діагностичних предикторів (таких як рівень глюкози, ІМТ, артеріальний тиск, кількість вагітностей, тощо) та одну цільову змінну -- наявність діабету (Рис. 1). Наша задача -- передбачити, чи є у жінки з вказаним набором ознак (1-8) діабет (значення цільової змінної pos) чи немає (neg).

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
1	6	148	72	35	NA	33.6	0.627	50	pos
2	1	85	66	29	NA	26.6	0.351	31	neg
3	8	183	64	NA	NA	23.3	0.672	32	pos
4	1	89	66	23	94	28.1	0.167	21	neg
5	0	137	40	35	168	43.1	2.288	33	pos
6	5	116	74	NA	NA	25.6	0.201	30	neg
7	3	78	50	32	88	31.0	0.248	26	pos
8	10	115	NA	NA	NA	35.3	0.134	29	neg
9	2	197	70	45	543	30.5	0.158	53	pos
10	8	125	96	NA	NA	NA	0.232	54	pos
11	4	110	92	NA	NA	37.6	0.191	30	neg
12	10	168	74	NA	NA	38.0	0.537	34	pos
13	10	139	80	NA	NA	27.1	1.441	57	neg
14	1	189	60	23	846	30.1	0.398	59	pos
15	5	166	72	19	175	25.8	0.587	51	pos

Рис.1. Змінні датасету 'PimaIndiansDiabetes2'

Зауважимо, що стартовий етап роботи з даними містить зокрема обробку пропущених значень. Оскільки метою даної статті не є методики роботи з пропущеними даними, отже ми просто видалили неповні рядки. В загальному випадку, звичайно, є ряд методик імпутації, які дозволяють мінімізувати втрати корисної інформації на цьому етапі. Такий підхід гарантує, що модель буде навчатися на біологічно коректних закономірностях, а не на артефактах обробки даних.

2.2. Аналіз топології даних: виявлення класів та зон перекриття

Для візуалізації структури багатовимірних даних та в нашому випадку визначення того, як клінічні показники формують простір станів пацієнтів, варто провести аналіз головних компонент (РСА, див. Рис. 2,3). Цей підхід дозволяє знизити розмірність даних, зберігши при цьому більшу частину загальної дисперсії в межах перших двох компонент.

В нашому випадку перші дві головні компоненти (Dim1 та Dim2) сумарно пояснюють 51.45% загальної варіативності даних (31.99% та 19.46% відповідно). Хоча це значення свідчить про високу складність даних, воно є достатнім для візуалізації основних закономірностей розподілу пацієнтів на площині. Проте в загальному випадку для подальшого покращення моделі варто розглянути аналіз залишку дисперсії (48.5%), який не потрапив на графік, можливо, шляхом додавання третьої компоненти або використання інших методів зниження розмірності.

Графічна візуалізація змінних в площині головних компонент (Рис. 2, Correlation Circle) потрібна для демонстрації того, які саме (біологічні) чинники стоять за геометричним розподілом точок датасету на площині головних компонент (Рис. 3). Діаграма демонструє ортогональність певних клінічних векторів, що свідчить про їх незалежність у межах даної патології. Довжина стрілки вказує на якість змінної на площині. Чим ближче стрілка до кола, тим краще вона пояснюється цими двома компонентами. Якщо стрілки знаходяться близько одна до одної, ці змінні сильно корелюють. Якщо стрілки направлені під кутом 90 градусів, відповідні їм змінні не пов'язані між собою.

Для нашого датасету на Рис. 2 бачимо, що змінні *glucose* (рівень глюкози), *mass* (ІМТ), *triceps* (товщина шкірної складки), *age* (вік) мають сильний позитивний зв'язок із першою головною компонентою (Dim1). Це вказує на те, що Dim1 відображає загальний метаболічний профіль. Змінні *age* (вік) та *pregnant* (кількість вагітностей) сильно корелюють між собою (відповідні їм стрілки знаходяться близько одна до одної) та мають значний внесок у від'ємні значення другої головної компоненти (Dim2). Саме вони спричиняють зміщення групи *pos* (діабет) в праву частину графіка, зображеного на Рис. 3. *Glucose* має найдовшу проєкцію на першу компоненту. Це означає, що вона робить найбільший внесок у розділення людей на групи за наявністю діабету. Показник *age* розташований майже перпендикулярно до вектора *mass*, що вказує на слабку кореляцію між цими параметрами в даній вибірці.

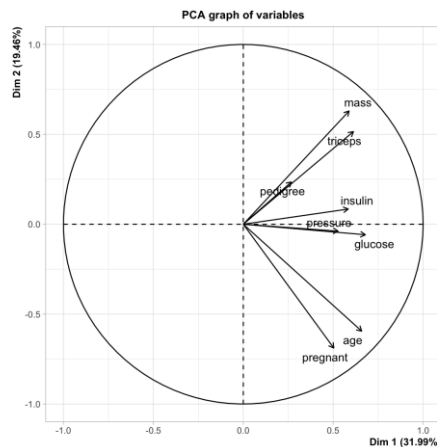


Рис. 2. Діаграма навантажень головних компонент датасету 'PimaIndiansDiabetes2'

Біплот (Рис. 3) – це графік, який відображає проєкцію багатовимірних даних на площину головних компонент. Аналіз біплоту, де точки марковані відповідно до фактичного діагнозу, дозволяє ідентифікувати наступні топологічні особливості:

1. Класова сегрегація: Пацієнти без діабету (клас 0 або neg) переважно концентруються в області нижчих значень першої головної компоненти (Dim1). Вектор глюкози та індексу маси тіла спрямований у протилежний бік, що підтверджує їх роль як домінантних факторів ризику. Попри перекриття, ми бачимо чітке зміщення: група neg тяжіє до лівої частини графіка (негативні значення Dim1); Група pos (діабет) зміщена вправо та трохи вище/нижче по осі Dim2. Еліпс neg більш компактний і витягнутий вздовж горизонтальної осі. Це свідчить про те, що показники здорових людей більш однорідні. Еліпс pos значно ширший. Це означає, що група діабету має велику варіативність показників (високий рівень глюкози, тиску або ІМТ може проявлятися по-різному в різних комбінаціях).

2. Зони високої ентропії (перекриття): Спостерігається значна область перекриття класів у центрі координатної площини. Це свідчить про існування "перехідних станів" – пацієнтів, чий біохімічний профіль уже наближається до патологічного, але ще не досяг клінічного порогу встановлення діагнозу, і, зокрема, про те, що діагноз неможливо встановити лінійно за однією або двома ознаками, що обґрунтовує необхідність використання більш складних методів машинного навчання (наприклад, випадкового лісу або нейронних мереж). Перекриття кластерів є очікуваним для медичних даних, де показники здоров'я (глюкоза, тиск, ІМТ) часто мають неперервний характер і не завжди утворюють ізольовані групи.

3. Атипові спостереження: Виявлено окремі випадки класу 1 (діабет) у зоні норми, що може вказувати на специфічні форми захворювання, не пов'язані з класичними факторами ожиріння чи віку. На графіку видно кілька жовтих трикутників (pos) у верхньому правому куті. Це пацієнти з критично високими показниками, які вибиваються із загальної маси навіть серед хворих.

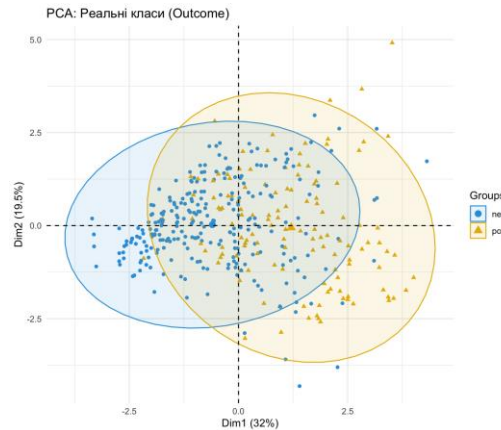


Рис. 3. Біплот для датасету 'PimaIndiansDiabetes2'

Отже, дані мають структуру, що дозволяє відрізнити хворих від здорових (зміщення груп вправо вздовж Dim1), але через велику зону перекриття у центрі графіка, діагностика лише за цими ознаками потребує більш складних методів аналізу, ніж простий візуальний поділ. Наявність розмитих меж та зон перекриття між класами підкреслює нелінійну природу досліджуваного явища. Топологічна складність даних обґрунтовує використання ансамблевих методів машинного навчання, здатних вибудовувати складні гіперплощини розділення там, де класичні лінійні методи дають похибку через перекриття ознак.

2.3. Інтеграція статистичних методів та графічного аналізу. Виявлення прихованих аномалій

Для глибшого розуміння природи даних та валідації якості вибірки необхідно провести розвідувальний аналіз даних (EDA) з використанням методів дескриптивної статистики та візуалізації. Самі по собі статистичні показники (середнє, медіана, дисперсія) можуть бути оманливими без графічного підтвердження.

Ключовими зображеннями на цьому етапі є діаграми розсіювання даних по класах, боксплоти, гістограми, розподіл даних по класах. Їх можна зобразити окремо для більшої деталізації зображення, інколи зручно подивитись на всі діаграми, виведені в одному графічному вікні, одним з таких комбінованих графіків є варіація матричної діаграми розсіювання (Matrix Plot), таке зображення є надзвичайно інформативним, оскільки

поєднує в собі одразу чотири типи аналізу: розподіли, кореляції, попарні залежності та порівняння груп (Рис. 4).

2.3.1. Аналіз розподілів

Використання гістограм та діаграм розсіювання дозволяє оцінити характер розподілу кожної змінної, що важливо зокрема для побудови моделей, які вимагають нормального розподілу даних. Окрім того, при зображенні на одному графіку гістограм для різних класів, можна побачити різницю в розподілі ознак для різних класів, що є ознакою того, що такі змінні будуть ефективними при включенні в модель класифікації.

На діаграмі розподілу цільової змінної часто спостерігається незбалансованість класів, коли кількість спостережень одного з класів суттєво (інколи критично) перевищує кількість спостережень іншого класу. Така структура даних створює загрозу "перекосу" моделі в бік мажоритарного класу. Алгоритм може демонструвати високу загальну точність (Accuracy), проте мати низьку чутливість (Sensitivity) до виявлення реальних випадків, наприклад, рідкісного захворювання. Ця проблема обумовлює необхідність використання спеціальних метрик оцінки (F1-score, AUC-ROC), корекцію значення порогової ймовірності класифікації та, можливо, методів ресемплінгу (SMOTE) на етапі підготовки даних.

2.3.2. Порівняльний аналіз груп та візуалізація викидів

Сирі дані (зокрема медичні) часто містять екстремальні (аномальні) значення, які, якщо їх лишити в датасеті, часто дуже псують результати та точність аналізу. Статистичні середні показники приховують цю проблему, проте графічний аналіз дозволяє відрізнити фізіологічно можливі викиди (наприклад, дуже високий ІМТ) від технічних помилок запису. Зокрема, часто може бути виявлено нульові значення у критичних параметрах (ІМТ, тиск), що з технічної (медичної) точки зору свідчить про пропуски в даних (missing values), замасковані під нулі, а не на реальні показники, і потребує подальшої імпутації (заповнення пропусків).

Боксплот (Box-and-whisker plot) – це простий, але дуже інформативний графік, який дає можливість побачити, якщо зображений для однієї вибірки, симетричність розподілу ознаки, розподіл елементів вибірки по квартилях, наявність викидів (аномальних спостережень). Окрім того, при зображенні боксплотів для різних груп поруч може бути виявлена міжгрупова диференціація, якщо помітна суттєва різниця між характеристиками центру та розсіювання ознак в різних групах. Для формального виявлення

статистичної значущості різниці між групами треба проводити додатково статистичні тести та оцінку розміру ефекту, проте візуальна оцінка дозволяє швидко виявити можливі ознаки, значення яких для різних класів є значущо різними. Це свідчить про високу роздільність класів за відповідною ознакою.

2.3.3. Аналіз кореляцій та статистичних залежностей

Теплові карти (Heatmap) та інші зображення кореляційних матриць є ключовим інструментом для розуміння структури залежності даних. Вони допомагають виявити таку проблему як мультиколінеарність ознак. Наприклад, сильна кореляція між віком та кількістю вагітностей може призвести до перекосу ваг у лінійних моделях та загальної нестійкості (ефект перенавчання) моделей.

За виявлення такої проблеми, особливо у випадку, коли планується побудова регресійних моделей, варто формально перевірки наявності мультиколінеарності, наприклад, через обчислення коефіцієнтів здуття дисперсії (VIF) для кожної незалежної змінної.

З іншого боку, ознаки, які мають незначущий зв'язок з цільовою змінною, можуть бути виключені з моделі. Часто це призводить до спрощення моделі без втрати точності (інколи навіть точність підвищується за рахунок відсутності зайвих змінних, які заважають приймати правильне рішення), підвищення інтерпретованості моделі, підвищенню швидкості роботи алгоритму.

Встановлені статистичні закономірності стають фундаментом для вибору методів попередньої обробки даних (масштабування та обробка нулів) перед етапом моделювання.

2.3.4. Попарні діаграми розсіювання

На попарних діаграмах розсіювання (нижній трикутник діаграми на Рис. 4), де об'єкти, що належать різним класам, зображуються різними кольорами, слід звернути увагу перш за все на перекриття класів. Якщо наявне помітне розділення класів по певній змінній, ця змінна буде корисною в моделі класифікації. Часто наявні змінні, для яких розподіл по класах виглядає рівномірним. Швидше за все, така змінна або не буде корисною для моделі, або навіть буде псувати модель класифікації.

Також на діаграмах розсіювання часто можна побачити інші проблеми. Наприклад, можливе скупчення точок вздовж осей свідчить про типову проблему "нульових значень", які потребують обробки перед навчанням моделі.

Хоча загальна матриця розсіювання дає комплексне уявлення про дані, висока щільність точок часто маскує локальну структуру розподілу цільової змінної. Для глибшого аналізу сепарабельності класів доцільно розглянути проєкції багатовимірного простору на площини найбільш інформативних ознак, наприклад, в нашому випадку, glucose-age та glucose-mass (Рис. 5). Така декомпозиція дозволяє чіткіше візуалізувати межу роздільності класів (decision boundary) та виявити області концентрації позитивних випадків діабету, які в двовимірному просторі формують виразні кластери. Окремий вивід цих графіків підтверджує, що комбінація високого рівня глюкози з віком або індексом маси тіла є ключовим нелінійним предиктором, що значно спрощує подальшу інтерпретацію результатів моделювання.

2.3.5. Розвідувальний аналіз для 'PimaIndiansDiabetes2'

В нашому випадку (Рис. 4) матриця візуалізації дозволяє провести глибокий аналіз структури даних, виділяючи ключові закономірності між фізіологічними показниками та наявністю діабету. Гістограми на діагоналі дають можливість провести аналіз розподілів. Більшість показників (insulin, pedigree, age, pregnant) мають виражений правосторонній скос. Це означає, що у вибірці переважають молоді люди з низькими показниками інсуліну, а екстремально високі значення зустрічаються рідше. Частина ознак мають розподіл, близький до нормального. Помітний зсув гістограм змінних для різних класів glucose, age, mass, triceps, insulin свідчать про вищі в середньому показники в групі з діабетом.

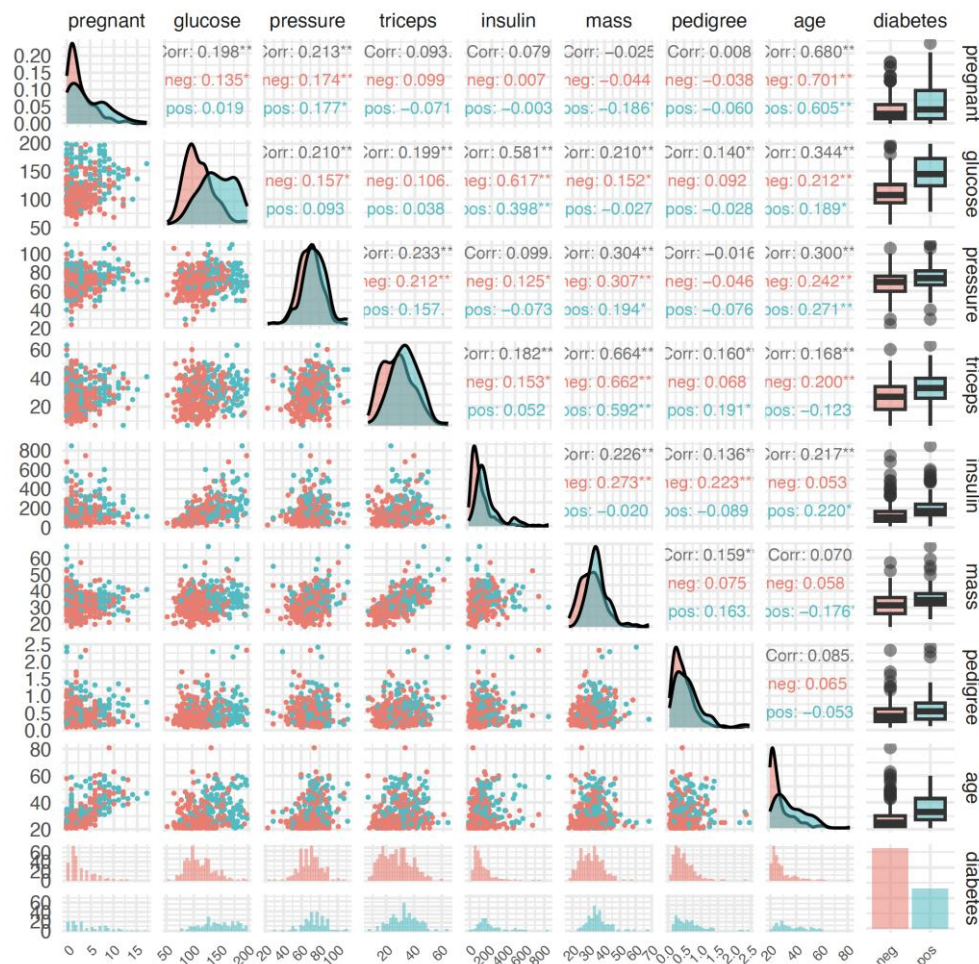


Рис. 4. Діаграма розсіювання для 'PimaIndiansDiabetes2' з розподілом на класи за всіма наявними в датасеті змінними

Розподіл цільової змінної (diabetes) показаний на стовпчиковій діаграмі в нижньому правому кутку. Вона каже про суттєву незбалансованість класів. Кількість пацієнтів без діабету (neg) майже вдвічі перевищує кількість хворих (pos). Це критичний момент для вибору метрик оцінки моделі (пріоритет на Precision/Recall, або інших критичних для конкретної даної задачі метрик, але не Accuracy). Окрім того, для покращення моделі варто в загальному випадку скористатися додатково методиками, які дозволяють збалансувати класи.

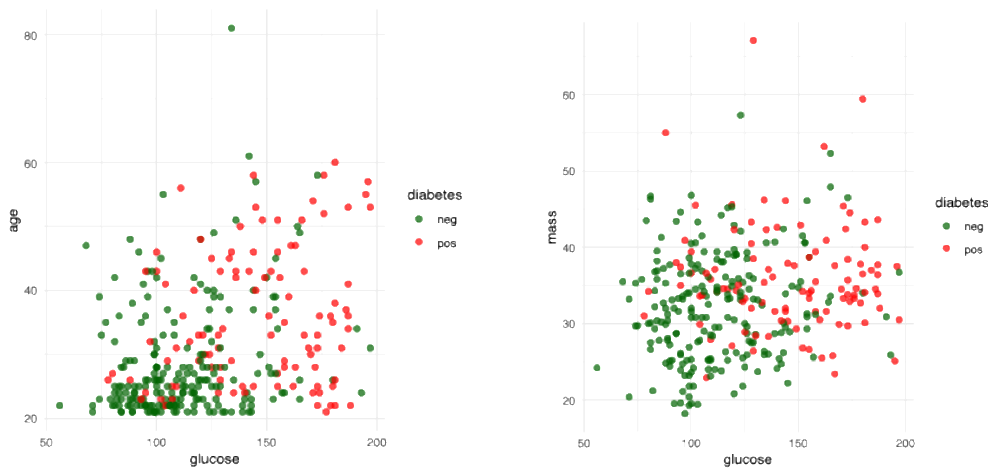


Рис. 5. Діаграма розсіювання з розподілом на класи в площині змінних “глюкоза” - “вік” (ліворуч) та “глюкоза” - “індекс маси тіла” (праворуч)

Аналіз боксплотів (правий стовпчик на Рис. 4) виявив також наявність викидів у багатьох показниках, зафіксовано значення, що виходять за межі 1.5 міжквартильного розмаху. Велика кількість точок за межами "вусів" у змінних *insulin* та *pregnant* вказує на наявність специфічних клінічних випадків або аномалій у зборі даних.

Боксплоти, побудовані в розрізі класів (здоровий/хворий), демонструють помітну різницю в медіанних значеннях, що свідчить про сепарацію класів за певними ознаками. Для змінної *glucose* найбільш виражена різниця між медіанами. У групи *pos* медіана значно вища, що підтверджує статус глюкози як головного предиктора. ІМТ та вік також показують помітний зсув медіан у бік вищих значень для групи з діабетом, що свідчить про їх значущий внесок у формування ризику.

Числові значення коефіцієнтів кореляції Пірсона (варіант за замовчуванням, верхній трикутник діаграми) виявляють важливі зв'язки в даних. Маємо свідчення мультиколінеарності ознак. Найсильніший зв'язок спостерігається між ознаками *age* та *pregnant* (0.68, природна позитивна кореляція), а також між *mass* та *triceps* (0.664, висока кореляція вказує на те, що обидва показники характеризують ступінь ожиріння). Це логічно з біологічної точки зору. Сильно значуще значення коефіцієнту кореляції між глюкозою та інсуліном (0.58) теж свідчить про очікуваний фізіологічний зв'язок, що підтверджує релевантність даних медичній теорії. Водночас слабка кореляція деяких ознак (наприклад, *pressure* – кров'яний тиск)

підштовхує до ідеї перевірки певних моделей, таких як дискримінантний аналіз, логістична регресія, на надмірність (overfitting).

Аналізуючи діаграми розсіювання, видно, що на багатьох графіках (наприклад, age - pressure) спостерігається сильне змішування червоних (neg) та блакитних (pos) точок. Це свідчить про те, що дані не є лінійно роздільними в базовому просторі ознак по жодній з окремо взятих змінних, проте класи все ж мають зсув по ряду змінних.

Отже, можемо зробити висновок попереднього аналізу датасету 'PimaIndiansDiabetes2'. Дані мають складну структуру з вираженим дисбалансом класів та нелінійними залежностями. Найбільшу роздільну здатність мають показники глюкози, віку та маси тіла, що підтверджується як візуальним аналізом розподілів, так і результатами кореляцій та PCA. Наявність викидів вимагає обов'язкового етапу стандартизації та імпутації даних.

Тепер маємо структурований фундамент для побудови надійного діагностичного алгоритму. Для багатьох моделей класифікаторів, таких як наївний баєсів класифікатор, випадковий ліс, всі ознаки доцільно залишити в моделі для подальшого тренування. Проте для побудови моделей на базі регресії (логістична регресія, дискримінантний аналіз) варто вилучити певні змінні для уникнення ефекту перенавчання та забезпечення стійкості моделі.

3. Побудова моделі логістичної регресії

Для прогнозування наявності діабету в якості базової моделі було обрано модель логістичної регресії. Це статистичний метод, що використовується для передбачення ймовірності події шляхом припасування даних до логістичної кривої. Модель логістичної регресії, з одного боку, дає непогані результати класифікації, а з іншого, на ній можна продемонструвати деякі додаткові діаграми, які свідчать про роздільність класів, зокрема залежно від вибору порогового рівня ймовірності класифікації. Власне, основним завданням алгоритму класифікації є знаходження оптимальної межі прийняття рішень (decision boundary), яка найкращим чином розмежовує об'єкти різних класів у просторі ознак, і для моделі логістичної регресії цю межу можна зобразити.

3.1. Математична модель

В основі моделі лежить лінійна комбінація ознак, яка доповнюється логістичним перетворенням.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_m \quad (1)$$

де β_i – коефіцієнти моделі, а x_i – значення ознак (наприклад, глюкоза, вік).

Модель повинна прогнозувати значення ймовірності приналежності об'єкта, який має вказаний набір значень ознак, до цільового класу (в нашому випадку – наявність діабету). Щоб отримати значення ймовірності в межах від 0 до 1, використовується функція сигмоїди:

$$P = \sigma(z) = \frac{1}{1+e^{-z}} \quad (2)$$

Ця функція дозволяє інтерпретувати результат P як ймовірність $P(y = 1|x)$, де y – клас об'єкта, $y \in \{0,1\}$, $x = (x_1, \dots, x_m)$ – вектор ознак.

Класифікація відбувається шляхом порівняння ймовірності з порогом (за замовчуванням це 0.5). Математично межа розділення класів в моделі логістичної регресії – це гіперплощина, для якої $\sigma(z) = 0.5$, а отже $z = 0$. Зокрема в одновимірному випадку (коли модель будується лише на основі однієї змінної, наприклад, рівня глюкози) ми можемо зобразити безпосередньо графік сигмоїди (2) (Рис. 6), у двовимірному просторі ознак це пряма лінія, яку можна побачити на графіку (Рис. 8):

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

Проте типовою є ситуація, коли стандартний поріг ймовірності 0.5 не дає задовільних результатів класифікації, і зміна цього значення дозволяє суттєво покращити якість моделі. Візуалізація роздільності класів дозволяє попередньо оцінити якість класифікації «за замовчуванням» та можливості покращення моделі шляхом зміни порогового рівня ймовірності.

Для побудови роздільних меж, оскільки ми не можемо зобразити модель в просторі високої розмірності, варто спочатку виділити найбільш ефективні ознаки, які для моделі логістичної регресії мають найвищу ефективність. Корисним зображенням для цього є діаграма важливості ознак.

3.2. Діаграма важливості ознак

Діаграма важливості ознак будує та демонструє коефіцієнти логістичної регресії β_i^* при кожній з нормалізованих змінних моделі, які є зручними для математичних розрахунків та статистичних тестів, і показують напрямок зміни результуючої змінної залежно від зміни значення ознаки, або ж відношення шансів (OR, Odds Ratio), більш традиційне при аналізі ризиків

в біології, медицині, тощо. Відношення шансів має просте трактування. В нашому випадку значення $OR=3.25$ для змінної *glucose* означає, що шанси на виникнення діабету зростають у 3.26 рази при зростанні рівня глюкози на одне стандартне відхилення.

Ми опускаємо математичні деталі, проте зауважимо, що при класифікації за допомогою логістичної регресії саме аналіз коефіцієнтів при нормалізованих змінних дозволяє зрозуміти, чому модель приймає те чи інше рішення.

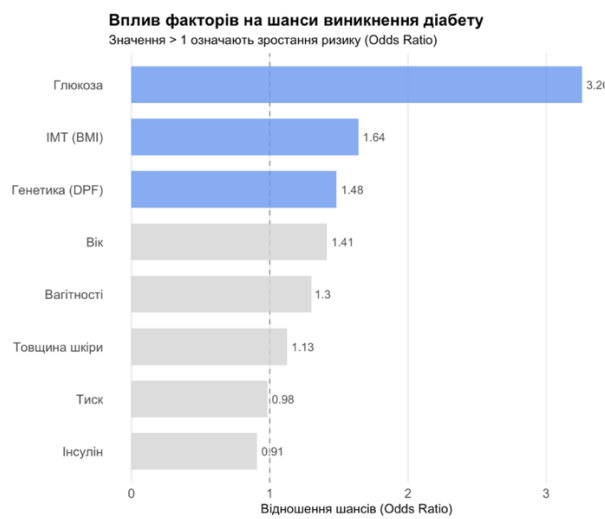


Рис. 6. Діаграма важливості ознак для класифікації наявності діабету.
Відношення шансів для кожної з ознак (стандартизованих)

На Рис. 6 зображена діаграма важливості ознак для класифікації наявності діабету на базі даних датасету 'PimaIndiansDiabetes2', де демонструється значення відношення шансів для кожної з ознак (стандартизованих). Синім позначено значущі на рівні 0.05 ознаки. Згідно цього графіку глюкоза, маса та спадковість є визначальними факторами. Насправді, на рівні 0.1 значущою змінною є також вік. Всю детальну інформацію можна отримати при побудові математичної моделі та аналізу відповідного статистичного звіту.

Діаграма важливості ознак дозволяє зробити наступні висновки:

1. Глюкоза є найвпливовішою ознакою. Якщо у пацієнта високий рівень цукру, модель майже автоматично підвищує ймовірність діабету. Odds Ratio близько 3.26 означає дуже суттєвий вплив. Це пояснює, чому на

попередніх графіках ми бачили найбільш чіткий поділ саме за віссю глюкози. Навіть невеликі (стандартизовані) коливання цього показника мають вдвічі більший вплив на ризик, ніж зайва вага або генетична схильність.

2. Індекс маси тіла – другий за важливістю фактор. Він має "вагу" 1.64. Це підтверджує медичну теорію про те, що ожиріння є ключовим тригером для діабету 2-го типу.

3. Генетична схильність також є важливим фактором, коефіцієнт 1.48.

4. Вік та Кількість вагітностей. Ці фактори мають помірний вплив ($OR = 1.41, 1.3$). Вони додають фоновий ризик. Ми бачили, що кількість вагітностей у цьому наборі даних корелює з віком, тому їх ефекти частково перекриваються.

5. Слабкі фактори (Тиск, Інсулін): Зверніть увагу, що pressure (кров'яний тиск) знаходиться біля одиниці (0.98). Це означає, що в межах цього дослідження тиск сам по собі не є хорошим індикатором діабету. Це пояснює, чому модель може помилятися на пацієнтах з високим тиском, але нормальною глюкозою.

Визначивши найбільш впливові змінні, можемо візуалізувати межі роздільності класів по них в різних проєкціях.

3.3. Візуальний аналіз роздільності класів

Оскільки в багатовимірному просторі важко візуалізувати роботу моделі, ми звертаємося до аналізу функції сигмоїди для окремих змінних, які володіють найбільшою розділюючою здатністю, та аналізу 2D-проєкцій для двох змінних. На таких діаграмах чіткіше простежується логіка роботи моделі. Зокрема ми можемо побачити наскільки ефективною є лінійна роздільність моделі, оскільки логістична регресія проводить пряму лінію (у 2D), з метою мінімізувати помилки класифікації. Візуалізація дозволяє побачити також перехідні зони, де ймовірність віднесення до позитивного класу змінюється від низької до високої.

Використання саме цих проєкцій підтверджує, що модель спирається на ключові біологічні маркери, а графічне відображення межі розділення дозволяє оцінити якість класифікації не лише через метрики (Accuracy, Precision), а й візуально, аналізуючи кількість "проникнень" одного класу в зону іншого.

3.3.1. Візуалізація роздільності за однією ознакою

Найвпливовішою на наявність діабету, судячи з попередньої візуалізації датасету, є змінна glucose. Для демонстрації процесу класифікації

за допомогою сигмоїдної кривої зобразимо її графік. Для цього ми використаємо модель (2) лише з однією ознакою – glucose. Модель оцінює ймовірність належності до цільового класу 1 (наявність діабету) через функцію сигмоїди. Точка перетину кривої з лінією $y=0.5$ є стандартним критичним порогом класифікації для однієї змінної.

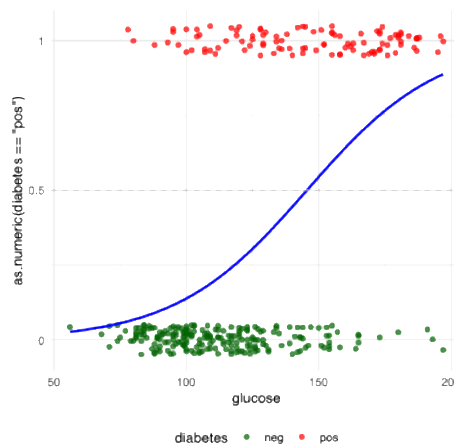


Рис. 7. Імовірність наявності діабету залежно від рівня глюкози (синя крива)

Отже, на Рис. 7 бачимо зображення моделі простої логістичної регресії, яка на основі лише однієї змінної, рівня глюкози, обчислює оцінку ймовірності наявності діабету. Як бачимо на рисунку, при низьких значеннях глюкози ймовірність переважно ближча до 0 (зелені точки neg), а при високих – до 1 (червоні точки pos). Варто звернути увагу, що у випадку встановлення стандартного порогу ймовірності для прийняття рішення щодо розділу на класи 0.5 (пунктирна пряма) ми б мали для тих жінок, які мають рівень глюкози вищий за приблизно 145, визначити наявність діабету, а тих, у кого цей показник менший, визнати здоровими. При цьому бачимо, що більшість жінок без діабету будуть правильно класифіковані як здорові, проте досить багато жінок з діабетом помилково потраплять в категорію neg, що, власне, недобре, інколи навіть критично. Тобто проста модель класифікує наявність діабету погано. Теоретично, якщо встановити пороговий рівень глюкози приблизно 80, то майже всі жінки з діабетом потраплять в позитивний клас. При цьому поріг ймовірності суттєво знизиться і більшість здорових жінок помилково потраплять в позитивний клас, що робить модель класифікації не дуже корисною.

Отже, незважаючи на те, що змінна glucose є дуже значущою при виявленні діабету, одна вона не зможе забезпечити ефективне розділення класів. Додаючи інші ознаки, ми можемо також побачити розділювальні межі у проєкції двох ознак для різних значеннях порогового рівня ймовірності.

3.3.2. Візуалізація межі розділення для двох ознак

Якщо на загальній діаграмі розсіювання структура класів виглядає занадто зашумленою, варто подивитися окремі діаграми для найбільш значущих пар ознак, як-от glucose-age та glucose-mass (Рис. 5).

Для моделі з двома предикторами ми вже не побудуємо зрозумілий графік модельної кривої, подібно до того, як ми зобразили графік логістичної кривої для моделі з одним предиктором glucose (Рис. 7), проте ми можемо побудувати інший графік, який демонструє те, як модель розділяє класи залежно від встановленого порогового рівня ймовірності приналежності до цільового класу – так звані розподільчі межі (decision boundaries).

Коли ми додаємо другу змінну (наприклад, mass), межа розділення стає лінійною функцією у просторі ознак. Математично це спирається на поняття логарифму шансів (Log-Odds), або логіту (logit).

Неважко побачити, що якщо взяти логарифм в рівнянні (2), де лінійна комбінація (1) будується лише по двох змінних, ми легко можемо отримати рівняння

$$\ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 \cdot \text{glucose} + \beta_2 \cdot \text{mass} \quad (3)$$

Тут P – це ймовірність приналежності до цільового класу, яка моделюється рівнянням (2), $P/(1 - P)$ – шанси (Odds), $\ln\left(\frac{P}{1-P}\right)$ – логарифм шансів або логіт.

Отже, по суті, хоча ймовірність P змінюється нелінійно, межа прийняття рішення залишається прямою лінією, яку можна зобразити, якщо зафіксувати значення ймовірності P .

Аналіз 2D-проєкцій (наприклад, glucose проти mass) дозволяє побачити цю пряму. На графіку ми також спостерігаємо зони впевненості, де точки одного класу знаходяться далеко від межі, та область невизначеності – точки біля самої лінії, де ймовірність наближається до рівня порогу класифікації (наприклад, 0.5), що вказує на ризик помилкової класифікації через перекриття ознак.

Використання такого підходу дозволяє зрозуміти фізичний зміст коефіцієнтів, побачити, наскільки сильно кожна ознака "відштовхує" межу розділення в той чи інший бік. Важливо також, що ми можемо оцінити, наскільки ефективний прогноз буде забезпечуватись з використанням різних значень порогової ймовірності класифікації.

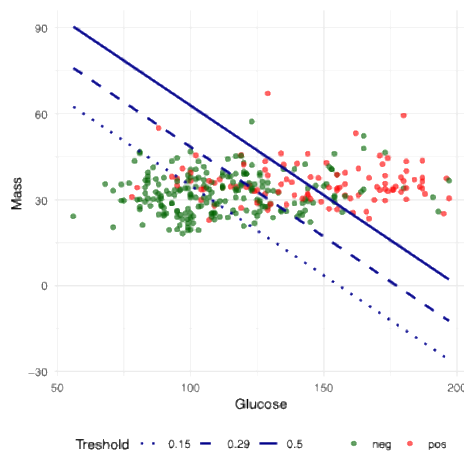


Рис. 8. Розподільчі межі для моделі логістичної регресії з двома предикторами glucose+mass, побудовані для порогових ймовірностей класифікації 0.15, 0.29 та 0.5

Для датасету 'PimaIndiansDiabetes2' такий графік наведено на Рис. 8. Зниження порогу класифікації зсуває лінію, яка є розподільчою межею для встановленого рівня ймовірності, вниз. Це дозволяє моделі бути більш чутливою (класифікувати більше випадків як pos), що критично важливо в медицині для мінімізації пропущених діагнозів (хибнонегативних результатів), навіть ціною збільшення кількості хибнопозитивних. Проте, як і у випадку з одним предиктором, модель не дуже добре розділяє класи. Для того, щоб діабет було виявлено майже у всіх, у кого він наявний, треба встановити дуже низький пороговий рівень ймовірності класифікації, наприклад, 0.15. І навіть за такого порогу будуть наявні хибнонегативні результати.

Важливо, що ми бачимо, що стандартний поріг класифікації на рівні 0.5 даватиме неприпустимо високий для медичної діагностики рівень хибнонегативних прогнозів, і для побудови більш ефективної моделі цей поріг необхідно буде суттєво знизити. Визначити оптимальний з певних міркувань поріг класифікації допомагають діаграми залежності таких

показників якості моделі як Sensitivity та Specificity від порогового рівня ймовірності та аналіз ROC-кривих після того, як будуть побудовані моделі на різних наборах предикторів, та оцінені метрики їх точності.

4. Метрики точності моделей логістичної регресії з різним набором вхідних ознак

Визначення метрик класифікації не є метою даної роботи, наголосимо лише на трьох з них, які є показовими та можуть бути критично важливими при прийнятті рішення щодо вибору моделі:

- загальна точність (Accuracy), яка показує загальну частку правильних прогнозів, найпростіша і найпоширеніша метрика, проте може бути оманливою на незбалансованих датасетах;

- повнота (Recall), вона ж – чутливість (Sensitivity) або рівень істинно позитивних прогнозів (True Positive Rate, TPR);

- специфічність (Specificity) або рівень істинно негативних прогнозів (True Negative Rate, TNR), та протилежна величина – рівень хибнопозитивних прогнозів (False Positive Rate, FPR), $FPR = 1 - Specificity$.

Вибір правильної метрики для оцінки моделі є критично важливим і залежить від конкретної задачі та наслідків різних типів помилок. Наприклад, в задачах діагностики певних хвороб критичною є наявність хибнонегативних результатів, коли особу з хворобою класифікують як здорову. В подібних випадках варто будувати модель з метою мінімізувати ймовірність хибнонегативних помилок. При цьому доведеться пожертвувати втратою точності класифікації здорових осіб: рівень хибнопозитивних класифікацій при цьому зростає, проте такі помилки часто є значно менш критичними.

Для аналізу та порівняння ми побудували різні моделі логістичної регресії для класифікації діабету: по всьому набору предикторів (Модель 1), з двома ознаками (рівень глюкози та ІМТ, Модель 2) та чотирма (рівень глюкози, ІМТ, вік та спадковість, Модель 3). Для порівняння ми додали модель наївного Бассового класифікатора, побудованого по повному набору змінних (Модель 4). Базові метрики точності для цих моделей, отримані на тестовій частині датасету, наведено в Таблиці 1. Окрім вказаних вище метрик точності ми додали для моделей логістичної регресії (без деталізації, має бути мінімально можливим) також значення інформаційного критерію Акаїке – важливий показник якості самої побудованої моделі.

Таблиця 1. Порівняння метрик моделей на тестових даних: логістичної регресії за усіма наявними змінними (Модель 1), за змінними glucose+mass (Модель 2), glucose+mass+age+pedigree (Модель 3), модель наївного Баєсового класифікатора по повному набору змінних (Модель 4)

Метрика	Модель 1	Модель 2	Модель 3	Модель 4
Accuracy	0.8077	0.7821	0.8077	0.8333
Sensitivity (Recall)	0.4231	0.4615	0.4615	0.6538
Specifity	1.000	0.9423	0.9808	0.9231
AIC	306.05	310.44	300.83	

В даному випадку наївний баєсів класифікатор показує найкращі показники для ключових метрик. Він демонструє найвищу загальну точність (0.8333), тобто він робить більше правильних прогнозів в цілому. Попри те, що всі моделі логістичної регресії мають вищу специфічність, тобто трохи краще ідентифікують випадки відсутності діабету, наївний Баєсів класифікатор має значно вищу чутливість (0.6538 проти 0.4615). І в даному випадку це ключовий момент. Вища чутливість означає, що він значно краще розпізнає наявність діабету, роблячи менше помилкових негативних прогнозів. Це робить модель більш надійною для таких критичних завдань, як медична діагностика.

Тут ми не наводимо для порівняння точність моделей наївного баєсового класифікатора (НБК) з меншою кількістю предикторів, зауважимо лише, що для них результати гірші. Насправді це спостереження чудово ілюструє ключову відмінність між моделями логістичної регресії та наївного баєсового класифікатора. Головна причина полягає в тому, що НБК робить дуже сильне, в абсолютній більшості випадків несправедливе, але часто корисне припущення про незалежність предикторів. У ньому, навіть якщо змінна (triceps, pressure, insulin) не є ідеальним предиктором, вона все одно вносить свій маленький, але, як виявилось, позитивний внесок у фінальний прогноз. Кожна така змінна, навіть якщо вона здається "шумовою", додає ще один незалежний шматочок інформації, що в сукупності підвищує загальну якість моделі. Проте це має місце не для всіх датасетів.

Логістична регресія, навпаки, враховує всі предиктори разом, оцінюючи їхній сукупний вплив на результат. Якщо між предикторами є сильна залежність (наприклад, як між glucose та insulin), модель може "заплутатися" і надати меншої ваги одному з них, або ж його включення не принесе суттєвого покращення.

Порівнюючи метрики моделей логістичної регресії (Моделі 1-3) бачимо, що найбільша загальна точність досягається для повної моделі та моделі з чотирма предикторами. Проте повна модель, маючи найвищу специфічність, програє іншим в ключовому моменті: вона гірше виявляє діабет. І ми знову підкреслимо ключову проблему у виборі моделі: не існує однієї найкращої метрики. Вибір моделі завжди залежить від конкретної задачі та ціни помилок. В даному випадку з моделей логістичної регресії варто обрати одну зі спрощених моделей, спираючись на критичне для нашої задачі значення чутливості, і, окрім того, зважаючи на мінімальне значення AIC, ми оберемо і будемо намагатися покращити модель з чотирма предикторами: `glucose + mass + age + pedigree`.

Зауважимо, що не варто одразу відкидати модель логістичної регресії як таку, яка програє в точності моделі наївного баєсового класифікатора. Спочатку треба проаналізувати інші графіки та можливості підвищити критичні метрики точності обох моделей.

Очевидно, всі побудовані моделі далеко не бездоганні, значення чутливості все ж надто низьке. Проте їх можна покращити за рахунок зміни порогу класифікації, що ми зробимо, користуючись зображенням ROC-кривої та пов'язаних з нею графіків.

5. Оцінка ROC-кривої та оптимізація порогу класифікації

За замовчуванням, більшість класифікаторів, що видають ймовірність, використовують поріг 0.5. Якщо ймовірність належності до позитивного класу більша за 0.5, об'єкт класифікується як позитивний. Це працює добре, коли класи приблизно збалансовані, а вартість помилок (хибно-позитивних і хибно-негативних) однакова. Однак, у реальних задачах зазвичай класи не є збалансованими, один клас зустрічається набагато частіше за інший. Окрім того, в різних задачах ціна помилок дуже відрізняється. Вартість хибно-негативного прогнозу може бути значно вищою, ніж вартість хибно-позитивного і навпаки. Наприклад, у медичній діагностиці пропустити хворобу (хибно-негативний) набагато гірше, ніж поставити помилковий діагноз (хибно-позитивний), який потім буде спростовано додатковими обстеженнями.

Зміна стандартного порогу дозволяє змінити баланс між Sensitivity та Specificity відповідно до вимог конкретної задачі. Знижуючи поріг (наприклад, з 0.5 до 0.35), ми робимо модель більш чутливою до позитивного класу, збільшуючи Sensitivity, але водночас можемо збільшити кількість хибно-позитивних результатів (знижити Specificity). Існує декілька

поширених методів для визначення оптимального порогу. Одним з найбільш зручних та поширених є аналіз ROC-кривої (Receiver Operating Characteristic) та значення площі під нею, AUC (Area Under the ROC Curve), як допоміжної метрики для порівняння загальної здатності моделей розділяти класи.

У машинному навчанні ROC-крива – це графік, що показує співвідношення між True Positive Rate (чутливістю, Sensitivity) та False Positive Rate ($1 - \text{Specificity}$) для всіх можливих значень порогу класифікації (ймовірність приналежності об'єкту до цільового класу, яка використовується моделлю як межа) від 1 до 0. Вона допомагає оцінити здатність моделі розрізняти класи, незалежно від конкретного порогу.

Ідеальна точка на кривій знаходиться у лівому верхньому куті, де Sensitivity і Specificity є максимальними, що відповідає 100% чутливості та 0% хибнопозитивних спрацювань ($\text{TPR}=1, \text{FPR}=0$). Маючи зображення ROC-кривої, можна знайти поріг, який відповідає цій точці на кривій або іншій точці, яка відповідає оптимальним значенням TPR та FPR для конкретної задачі. Модель, яка класифікує випадковим чином, матиме ROC-криву, що проходить майже по діагоналі від (0,0) до (1,1).

Ще одним показником загальної якості моделі класифікатора є AUC – це площа під ROC-кривою. Значення AUC варіюється від 0 до 1. При цьому:

- $\text{AUC} = 1$ свідчить про ідеальну модель;
- $\text{AUC} = 0.5$ означає, що модель класифікує випадковим чином;
- $\text{AUC} < 0.5$ означає, що модель працює гірше за випадкову.

Отже, загалом високе значення AUC вказує на те, що модель добре розрізняє класи. AUC є особливо корисною метрикою для незбалансованих датасетів, оскільки вона не залежить від розподілу класів. На Рис. 9 можна побачити вигляд ROC-кривої для невдалої моделі класифікатора, де крива йде вздовж діагоналі, і значення AUC буде близьким до 0.5. Це свідчить про те, що рішення приймається абсолютно випадково, на основі такої інформації немає сенсу робити прогнози.

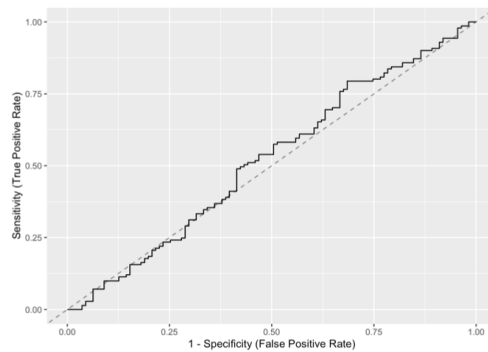


Рис. 9. Приклад ROC-кривої для невдалої моделі класифікації прогнозу динаміки курсу валют за даними кількох останніх днів

Оскільки загальна точність не завжди є найкращою метрикою, особливо для незбалансованих даних, ми можемо використати ROC-криву та AUC, як допоміжну метрику, для більш глибокого аналізу. Маючи графік ROC-кривої та значення AUC ми можемо порівняти загальну якість різних моделей (здатність до правильної класифікації), наприклад, логістичної регресії та НБК. Також, змінюючи поріг класифікації, ми можемо налаштовувати модель на краще виявлення того чи іншого класу. Графік ROC-кривої дозволяє побачити цю можливість наочно.

5.1. Оцінка ROC-кривої та оптимізація порогу класифікації для задачі виявлення діабету

В прикладі з класифікацією діабету нашою головною метою є зменшення ймовірності визнання хворої людини здоровою, тобто максимізація чутливості. Аналіз за допомогою ROC-кривих дозволить нам не лише порівняти загальну ефективність моделей, а й знайти ідеальний компроміс між чутливістю та специфічністю для кожної з них.

Ми будемо аналізувати дві моделі, які виявилися найкращими:

- 1) модель логістичної регресії, побудованої по чотирьох змінних glucose + mass + age + pedigree,
- 2) модель НБК, побудованого на всіх вхідних змінних.

Візуалізуємо ROC-криві для обох моделей на одному графіку (Рис. 10). Для того, щоб розуміти можливості моделі, на ROC-кривій помітимо точку,

яка відповідає порогу 0.5, який за замовчуванням використовується при класифікації об'єктів моделлю.

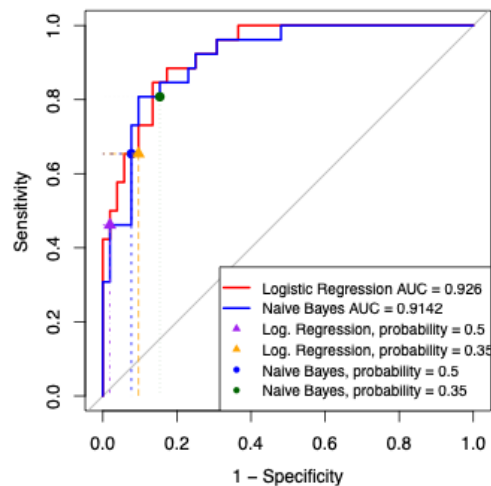


Рис. 10. ROC-криві для моделей класифікації наївної баєсівської та логістичної регресії з чотирма предикторами з різними пороговими ймовірностями

Треба звернути увагу на цьому зображенні на те, що обидві криві мають помітний вигин, отже, обидві моделі доволі непогані, і можуть забезпечити приблизно однакову точність класифікації при корекції порогу. І жодна з моделей не виглядає суттєво кращою за іншу. Також видно неозброєним оком, що стандартний поріг 0.5 дає далеко не найкращий результат. Зменшивши поріг, ми можемо суттєво підняти рівень чутливості для обох моделей. Отже, зміщення порогу в бік, наприклад, 0.35, є вдалою спробою підвищити чутливість (тобто, виявляти більше позитивних випадків) за рахунок специфічності (модель частіше помилково ідентифікуватиме негативні випадки як позитивні). Як бачимо, в обох моделях чутливість суттєво зросла (точки посунулись вище), при невеликих втратах в значеннях специфічності (точки посунулись праворуч), що і було нашою метою. Проте, очевидно, ми можемо ще опустити поріг класифікації для покращення результатів.

Графік на Рис. 10 демонструє, що зміна порогу для кожної з моделей може суттєво вплинути на ефективність класифікації. Для визначення оптимального порогу є різні методи, які спираються на різні критерії

(наприклад, визначення найближчої точки до верхнього лівого кута ROC-кривої або максимізації певної метрики, як-от F1-score). Знову таки, поняття оптимального порогу залежить від конкретної задачі. Для того, щоб побачити динаміку метрик якості, побудуємо окремо графіки залежності чутливості, специфічності та середньої точності від значення порогової ймовірності та спробуємо побачити прийнятне для нашої задачі її значення (Рис. 11).

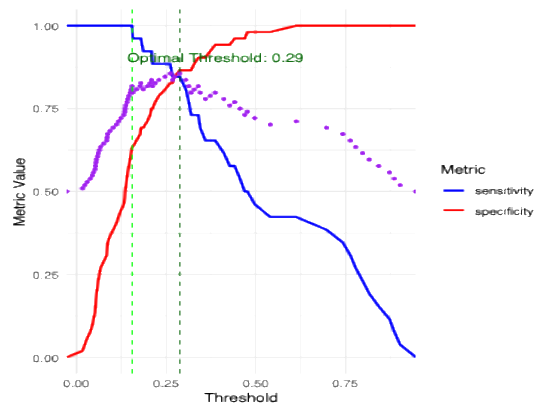


Рис. 11. Залежність базових метрик (чутливості, специфічності та збалансованої точності) від порогової ймовірності класифікації для моделі класифікації логістичної регресії з чотирма предикторами. Пунктирними лініями позначені порогові ймовірності на рівні 0.155 (світло-зелена) та 0.29 (темно-зелена).

Бачимо, що стовідсоткового виявлення діабету можна досягнути, якщо встановити пороговий рівень ймовірності класифікації 0.155 (світло-зелена пунктирна лінія). При цьому рівень специфічності буде приблизно 0.63. Насправді, зважаючи на те, що ця похибка є суттєво менш критичною, це цілком придатний показник. Якщо йти шляхом максимізації збалансованої точності, то варто обрати рівень порогової ймовірності 0.29.

Метрики точності цієї моделі (логістична регресія, 4 предиктори, пороговий рівень ймовірності класифікації 0.29) зображені на Рис. 12. Бачимо, що в результаті зміни порогової ймовірності класифікації ми отримали не лише доволі високий рівень чутливості, але й збільшення загальної точності класифікації. При цьому втрата в значенні специфічності несуттєва. Знову таки, якщо стоїть задача мінімізувати ймовірність хибнонегативного результату, можна, користуючись останніми діаграмами, ще опустити рівень ймовірності класифікації для підвищення чутливості.

```

> print(conf_matrix_logreg_new_thresh)
Confusion Matrix and Statistics

          Reference
Prediction neg pos
neg      45   4
pos       7  22

      Accuracy : 0.859
      95% CI   : (0.7617, 0.9274)
No Information Rate : 0.6667
P-Value [Acc > NIR] : 0.0001012

      Kappa : 0.6916

McNemar's Test P-Value : 0.5464936

      Sensitivity : 0.8462
      Specificity : 0.8654
      Pos Pred Value : 0.7586
      Neg Pred Value : 0.9184
      Prevalence : 0.3333
      Detection Rate : 0.2821
      Detection Prevalence : 0.3718
      Balanced Accuracy : 0.8558

'Positive' Class : pos

```

Рис. 12. Матриця похибок та інші метрики для моделі логістичної регресії виявлення діабету за чотирма змінними з порогом класифікації 0.29

Подивимось на аналогічні зображення для найвного баєсового класифікатора на Рис. 13. Бачимо, що модель доволі суттєво відрізняється від моделі логістичної регресії за динамікою метрик залежно від порогової ймовірності. Зокрема 100% випадків діабету на тестовій вибірці модель виявляє при пороговому рівні 0.022, при цьому модель буде давати хибнопозитивний прогноз майже в половині випадків. Це значно гірша ситуація, ніж в попередній моделі.

Максимальне значення збалансованої точності визначено на рівні порогу класифікації 0.38. Якщо встановити цей поріг в модель, матимемо результати для метрик, наведені на Рис. 14. Бачимо високу загальну точність, але рівень чутливості дещо нижчий, ніж в моделі логістичної регресії.

На наш погляд, знову таки, маючи на меті підвищення чутливості без великих втрат в специфічності, можемо для баєсової моделі ще трохи зменшити пороговий рівень класифікації. На Рис. 13 видно, що якщо зсунути його до рівня 0.305, ми дещо виграємо в чутливості і небагато втратимо в специфічності. Якщо встановити цей поріг в моделі найвного баєсового

класифікатора, ми дещо втратимо в загальній точності, проте досягнемо рівня чутливості як в моделі логістичної регресії з порогом 0.29.

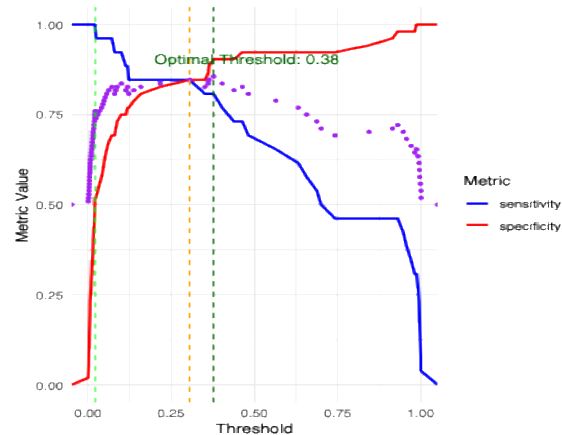


Рис. 13. Залежність базових метрик (чутливості, специфічності та збалансованої точності) від порогової ймовірності класифікації для наївної баєсової моделі класифікації. Пунктирними лініями позначені порогові ймовірності 0.022 (світло-зелена), 0.305 (помаранчева) та 0.38 (темно-зелена)

```
Confusion Matrix and Statistics

      Reference
Prediction neg pos
neg      47    5
pos       5   21

      Accuracy : 0.8718
      95% CI : (0.7768, 0.9368)
      No Information Rate : 0.6667
      P-Value [Acc > NIR] : 3.132e-05

      Kappa : 0.7115

McNemar's Test P-Value : 1

      Sensitivity : 0.8077
      Specificity : 0.9038
      Pos Pred Value : 0.8077
      Neg Pred Value : 0.9038
      Prevalence : 0.3333
      Detection Rate : 0.2692
      Detection Prevalence : 0.3333
      Balanced Accuracy : 0.8558

      'Positive' Class : pos
```

Рис. 14. Матриця похибок та інші метрики для наївної баєсової моделі виявлення діабету з пороговим рівнем класифікації 0.38.

Отже, для нашого датасету завдяки зсуву порогового рівня класифікації вдалось дуже суттєво покращити якість обох моделей, що було б важче зробити без відповідних діаграм. При цьому модель логістичної регресії при однаково високій чутливості виглядає дещо кращою за НБК за рахунок вищої специфічності (0.865 проти 0.846). Проте в таких випадках при порівнянні моделей обов'язково треба дати відповідь на питання: чи не є це випадковістю?

6. Статистична перевірка ідентичності моделей

При виборі оптимальної моделі треба усвідомлювати, що отримані метрики точності (зокрема Accuracy, AUC, Sensitivity) є точковими оцінками, обчисленими для конкретної реалізації випадкової вибірки та специфічного її розбиття на тренувальну та тестову частини. Оскільки ми оперуємо вибірковими даними, оцінка будь-якої метрики є випадковою величиною, значення якої залежить від варіативності вхідних даних. Отже, перевага однієї моделі над іншою може бути наслідком випадкового збігу, а не реальної різниці в узагальнювальній здатності. Якщо розбити вибірку на тренувальну та тестову частини іншим способом, метрики можуть змінитися, і найкраща модель серед кількох альтернативних може втратити лідерську позицію. Очевидно, якщо взяти інший набір даних, то також результати точності можуть бути зовсім іншими.

Для прийняття обґрунтованого рішення щодо вибору фінальної архітектури класифікатора необхідно провести формальний статистичний аналіз із використанням наступного інструментарію:

1. Аналіз довірчих інтервалів (CI) для Accuracy (виводяться в стандартному звіті метрик моделі, див. Рис. 12, 14). Ми розглядаємо не лише точкову оцінку, а й інтервал, у якому з заданою надійністю (зазвичай 0.95) знаходиться істинне значення параметру. Якщо довірчі інтервали двох моделей мають значну зону перекриття, ми не маємо статистичних підстав стверджувати про перевагу однієї з них, навіть якщо її середня точність вища. Це свідчить про високу дисперсію помилки моделі.

2. Критерій ДеЛонга (DeLong's Test) для порівняння AUC – загальної здатності моделей розділяти класи. Оскільки AUC є інтегральною метрикою якості розділення класів, ми використовуємо критерій ДеЛонга для перевірки нульової гіпотези $H_0: AUC_1 = AUC_2$. Цей тест враховує кореляцію між кривими (оскільки вони побудовані на одній вибірці) і дозволяє визначити,

чи є різниця в роздільній здатності моделей статистично значущою ($p - value < 0.05$).

3. Критерій Мак-Немара (McNemar's Test). Цей непараметричний тест є критично важливим при зміні порогів класифікації. Він аналізує узгодженість помилок моделей, фокусуючись на неузгоджених парах (випадках, де одна модель помилилася, а інша – ні). Нульова гіпотеза стверджує, що ймовірності помилок обох моделей однакові. Це дозволяє нам статистично підтвердити, що зсув порогу дійсно приводить до значущої зміни структури помилок (зменшення, наприклад, ймовірності похибки II роду за рахунок похибки I роду).

Не наводячи деталей, зауважимо, що статистичні тести для задачі виявлення діабету показують загальну ідентичність роботи цих двох обраних моделей, тобто формально немає математичної підстави вважати одну модель кращою за іншу на цій вибірці. При скоригованих порогах моделей ЛР та НБК критерій Мак-Немара дає p -значення, рівне 0.01333, тобто на рівні значущості 0.05 ми маємо відхилити нульову гіпотезу про рівність моделей. Проте воно не є переконливим і до висновку треба ставитися з обережністю.

У випадку, коли моделі статистично рівні, варто обрати ту, яка простіша в інтерпретації (зазвичай це логістична регресія) або ту, яка має менше "дорогих" помилок (False Negatives) в конкретній задачі.

7. Висновки.

У роботі обґрунтовано концептуальну значущість міждисциплінарної інтеграції методів візуалізації на всіх етапах життєвого циклу побудови класифікаційних моделей. Проведене дослідження дозволяє зробити наступні узагальнюючі висновки:

1. Роль візуальної аналітики на етапі препроцесингу: Застосування комплексних матричних діаграм (Matrix Plots) та аналізу топології даних у просторі головних компонент (PCA) дозволяє не лише виявити приховані закономірності та колінеарні зв'язки (зокрема, між віком пацієнта та репродуктивним анамнезом), а й ідентифікувати критичні аномалії в даних. Це підтверджує, що візуалізація є першим і необхідним етапом верифікації якості вибірки. Без візуального контролю математичні алгоритми ризикують оперувати артефактами даних (маскованими пропусками чи нетиповими викидами), що нівелює подальшу точність моделювання.

2. Геометрична інтерпретація статистичних рішень: Встановлено, що перехід від багатовимірних матриць до аналізу проєкцій на головні площини та візуалізації меж прийняття рішень (Decision Boundaries) дозволяє

дестилювати "чорну скриньку" алгоритмів машинного навчання. Візуальна оцінка зон перекриття класів дає досліднику інтуїтивне розуміння фізичного змісту моделі, яке неможливо отримати виключно через інтегральні метрики точності.

3. Оптимізація структури моделі: Шляхом порівняльного аналізу конфігурацій логістичної регресії встановлено, що нарощування кількості предикторів не завжди призводить до покращення прогностної сили. Використання інформаційного критерію Акаїке (AIC) спільно з аналізом факторних навантажень PCA дозволило обґрунтувати вибір оптимальної чотирьохфакторної моделі, яка забезпечує найкращу узагальнювальну здатність.

4. Візуальне обґрунтування вибору порогів: Продемонстровано, що візуалізація логіт-трансформацій та ймовірнісних кривих є фундаментальним підґрунтям для стратегічної зміни порогів класифікації. Це дозволяє усвідомлено керувати балансом між помилками першого та другого роду, що є критично важливим у прикладних задачах, де ціна пропуску цільової події є надзвичайно високою.

5. Методологічна цілісність: Запропонований підхід доводить, що статистична верифікація (критерії ДеЛонга, Мак-Немара) та графічна аналітика є взаємодоповнюючими компонентами. Візуалізація формує гіпотезу та стратегію моделювання, а формальні критерії забезпечують математичну строгість отриманих результатів.

Отже, інтеграція методів візуалізації на кожному етапі машинного навчання перетворює процес побудови моделі з «чорної скриньки» на прозорий, верифікований інструмент підтримки прийняття рішень. Такий підхід мінімізує ризики формального застосування алгоритмів і дозволяє створювати моделі, що мають не лише високі математичні метрики, а й реальну діагностичну цінність.

Список використаних джерел

1. Ribeiro, M. T., Singh, S., & Guestrin, C. "Why should i trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135-1144: <https://arxiv.org/abs/1602.04938>
2. Lundberg, S. M., & Lee, S. I. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, №30, 2017: <https://arxiv.org/abs/1705.07874>

3. Shneiderman, Ben. The Big Picture for Big Data: Visualization. Science (New York). 2014. 343. 730. 10.1126/science.343.6172.730-a.
4. Hanna Livinska, Daria Skrypnyk, José Luis GALÁN-GARCÍA: Comparative Study of Machine Learning Algorithms for Diabetes Detection Using Binary Data, *"Bulletin of Taras Shevchenko National University of Kyiv. Physics and Mathematics"*, №1(78)/2024 -- Ст. 119-127.
5. Blake, C.L. & Merz, C.J. UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Irvine, Department of Information and Computer Science. Formerly available from <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

*Лівінська Ганна Володимирівна, к.ф.-м.н, доцент
Розора Ірина Василівна, д. ф.-м.н, професор
Шарапов Михайло Михайлович, к.ф.-м.н, доцент,
Македон Тетяна Валентинівна*

СИСТЕМНА МЕТОДОЛОГІЯ СТАТИСТИЧНОГО АНАЛІЗУ СКЛАДНИХ СТОХАСТИЧНИХ СИСТЕМ ТА ВЕРИФІКАЦІЇ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ НА ОСНОВІ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ ТА МЕТОДІВ РЕСЕМПЛІНГУ

У статті запропоновано системну методологію верифікації алгоритмів машинного навчання, що функціонують в умовах складних стохастичних об'єктів. Обґрунтовано, що класичні асимптотичні статистичні критерії часто виявляються неадекватними через порушення фундаментальних припущень про нормальність розподілу та однорідність дисперсій у реальних системах. На основі результатів імітаційного моделювання методом Монте-Карло проведено порівняльний аналіз потужності статистичних методів, що дозволило встановити переваги ресемплінг-верифікації для систем із «важкими хвостами» розподілів та обмеженим обсягом вибірки. Особливу увагу приділено концептуальним ризикам прийняття рішень, пов'язаним із низькою статистичною потужністю та прихованими похибками II роду, які можуть призводити до хибної верифікації неадекватних моделей. Показано, що інтеграція методів ресемплінгу в цикл системного аналізу дозволяє забезпечити стійкість прогнозних рішень та мінімізувати стохастичну невизначеність. Запропонована методологія може бути рекомендована для використання в інтелектуальних системах підтримки прийняття рішень, медичній діагностиці та при верифікації складних стохастичних моделей, де надійність висновків є пріоритетною.

Ключові слова: системний аналіз, стохастичні системи, верифікація моделей, методи ресемплінгу, асимптотичні критерії

The article proposes a systemic methodology for the verification of machine learning algorithms operating within complex stochastic environments, based on simulation modeling and resampling methods. It is demonstrated that classical asymptotic statistical criteria often prove inadequate due to the violation of fundamental assumptions regarding normality and homoscedasticity in real-world

systems. Based on Monte Carlo simulation results, a comparative analysis of statistical power was conducted, establishing the advantages of resampling-based verification for systems characterized by "heavy-tailed" distributions, high heteroscedasticity, and limited sample sizes. Particular attention is paid to the conceptual risks of decision-making associated with low statistical power and latent Type II errors, which can lead to the false verification of inadequate models. The proposed methodology is recommended for use in intelligent decision support systems and medical diagnostics, where the reliability of conclusions is paramount.

Keywords: system analysis, stochastic systems, verification methodology, resampling methods, asymptotic criteria

1. Вступ

Процеси прийняття рішень у складних динамічних системах дедалі частіше базуються на прогностичних моделях машинного навчання, які змушені функціонувати в умовах високого рівня стохастичної невизначеності. Проте, попри високі показники формальної точності, проблема об'єктивної оцінки адекватності таких моделей у реальних умовах функціонування залишається відкритою. Особливої гостроти це питання набуває при аналізі складних стохастичних систем, де дані характеризуються високим рівнем невизначеності, асиметрією та нестационарністю. Традиційний підхід до оцінки якості таких моделей зазвичай базується на обчисленні агрегованих метрик, таких як Ассурасу або F1-score, на фіксованих тестових вибірках. Однак, з позицій системного аналізу, подібна оцінка є суто статичною і не враховує внутрішню стохастичну динаміку об'єкта. Це створює серйозний методологічний ризик ігнорування прихованих похибок, які виникають внаслідок невідповідності структури емпіричних даних теоретичним припущенням класичної статистики.

Ця невідповідність часто стає критичною, оскільки більшість стандартних статистичних критеріїв, що використовуються для верифікації гіпотез про параметри систем, за своєю природою є асимптотичними. Формально вони гарантують коректність висновків лише за умови прямування обсягу вибірки до нескінченності та дотримання нормального закону розподілу. Проте у реальних задачах системного аналізу ми постійно стикаємося з недотриманням цих фундаментальних припущень. Типовими є ситуації роботи з малими вибірками, де обсяг даних є недостатнім для справедливості граничних теорем. Крім того, складні системи часто демонструють ефекти «важких хвостів» розподілів (наприклад, розподіл Коші), що робить класичні оцінки середнього та дисперсії нестійкими, або ж

виявляють гетероскедастичність – мінливість стохастичного шуму, що призводить до суттєвого зміщення оцінок адекватності.

Зазначені деформації даних безпосередньо впливають на статистичну потужність критеріїв, яка визначає здатність моделі виявляти реальні закономірності в умовах стохастичного шуму. В ігноруванні саме цього аспекту криється небезпека похибок II роду: прийняття моделі як адекватної, коли насправді її прогнозна сила є ілюзорною. У таких умовах виникає нагальна потреба у переході від асимптотичних методів до альтернативних стратегій верифікації. Одним із найбільш перспективних інструментів тут виступають методи ресемплінгу та перестановочні тести, які дозволяють оцінювати параметри системи безпосередньо на наявних даних, мінімізуючи ризики, пов'язані з невідомою природою розподілу.

Питання статистичної верифікації та стійкості моделей машинного навчання знаходяться у центрі уваги багатьох сучасних досліджень. Фундаментальні аспекти обмеженості асимптотичних методів при роботі з нестандартними розподілами детально розглядалися у класичних працях з непараметричної статистики ([1], [2]), де підкреслювалася перевага методів, вільних від припущень про форму розподілу. Розвиток обчислювальних потужностей дав поштовх до активного впровадження методів ресемплінгу, зокрема бутстреп-аналізу та перестановочних тестів ([3], [4]), які дозволяють отримувати надійні оцінки варіативності параметрів навіть у складних стохастичних середовищах. У контексті системного аналізу сучасні автори все частіше наголошують на необхідності переходу від оцінки точності на одиничних вибірках до аналізу статистичної стабільності прогнозів ([5], [6]), що є критичним для систем прийняття рішень у медицині, фінансах та управлінні критичною інфраструктурою.

Попри значну кількість публікацій, присвячених технічним аспектам алгоритмів ресемплінгу, спостерігається певний дефіцит робіт, що інтегрують ці методи у цілісну системну методологію верифікації моделей машинного навчання з акцентом на аналіз стохастичної природи об'єктів. Зокрема, недостатньо висвітленим залишається питання порівняльної потужності таких методів відносно класичних критеріїв у специфічних умовах, таких як наявність «важких хвостів» або екстремально малий обсяг спостережень.

Метою даної статті є обґрунтування системної методології статистичної верифікації моделей машинного навчання, що базується на використанні методів ресемплінгу для оцінки їхньої адекватності. На відміну від асимптотичних методів, ресемплінг дозволяє оцінити реальну

варіативність моделі та потужність критеріїв безпосередньо на наявних даних, що забезпечує глибше розуміння стійкості складних стохастичних систем. Основна увага приділяється порівняльному аналізу статистичної потужності асимптотичних та перестановочних критеріїв у складних стохастичних умовах. Дослідження спрямоване на мінімізацію ризиків виникнення прихованих похибок II роду, що дозволяє забезпечити високу надійність системного аналізу та стійкість результатів класифікації у середовищах із високим рівнем невизначеності.

2. Теоретичні засади системної методології верифікації на основі ресемплінгу

Запропонована методологія розглядає верифікацію моделі машинного навчання не як разову перевірку точності, а як ітераційний процес оцінки статистичної стійкості в умовах стохастичної невизначеності. В основу підходу покладено принцип мінімізації ризику неадекватності, що реалізується через три рівні аналізу.

Перший рівень: Оцінка статистичної потужності та величини ефекту. Традиційно в машинному навчанні основна увага приділяється p -value або помилкам I роду (хибнопозитивні результати). Проте для складних систем часто критичне значення має потужність – здатність моделі (або критерію верифікації) виявляти реальні закономірності там, де вони існують. Методологія передбачає використання методів ресемплінгу для побудови емпіричних розподілів статистики, що дозволяє оцінити потужність без апіорних припущень про параметри генеральної сукупності. Важливим аспектом на цьому рівні є контроль величини ефекту, оскільки суто статистична значущість ($p - value < 0.05$) на великих даних може маскувати практично нікчемні розбіжності. Системний підхід вимагає оцінки стандартизованого розміру ефекту (наприклад, d Коена), що дає змогу відокремити формальну математичну значущість від реальної практичної цінності знайдених модельних розбіжностей.

Другий рівень: Тестування стійкості до порушення асимптотики. Системна верифікація вимагає перевірки чутливості моделі до аномалій даних. Основним інструментом тут виступає перестановочний критерій (permutation test). Його математична перевага полягає в тому, що він спирається на точний розподіл для конкретної реалізації даних, а не на граничні асимптотичні наближення. Це дозволяє уникнути зміщення оцінок у випадках гетероскедастичності або наявності «важких хвостів» (розподілів з нескінченною дисперсією, таких як розподіл Коші).

Третій рівень: Оцінка адекватності через ресемплінг-метрики. Замість точкової оцінки Accuracy, методологія пропонує використовувати інтервальне оцінювання параметрів через процедуру Bootstrap. Це дає змогу визначити довірчі інтервали для кожної метрики якості. У межах системного аналізу критично важливо будувати надійні інтервали не лише для точності моделі, а й безпосередньо для величини ефекту. На відміну від класичних формул, ресемплінг-оцінка довірчих інтервалів відображає реальну точність вимірювань за наявності стохастичного шуму. Якщо довірчий інтервал є занадто широким, модель вважається статистично крихкою, навіть за умови високого середнього значення точності.

Така багаторівнева верифікація дозволяє трансформувати висновок про якість моделі з двійкового «точна/неточна» у системну оцінку «адекватна/неадекватна» з урахуванням ризиків II роду.

3. Методологія аналізу потужності критеріїв

Перед переходом до аналізу результатів моделювання необхідно формалізувати об'єкт статистичної верифікації. У межах даної методології ми розглядаємо класичну задачу порівняння двох станів складної системи (або двох незалежних вибірок даних), що генеруються стохастичним об'єктом.

Основним інструментом верифікації є перевірка нульової гіпотези про однорідність (рівність) центрів розподілу (математичних сподівань у класичному сенсі або параметрів зсуву для непараметричних моделей) у двох групах: $H_0: \mu_1 = \mu_2$. Альтернативна гіпотеза, відповідно, стверджує наявність системних відмінностей: $H_1: \mu_1 \neq \mu_2$.

Для об'єктивної оцінки чутливості критеріїв вводиться поняття розміру ефекту (Effect Size), який у нашому дослідженні представлений параметром зсуву (Shift). Розмір ефекту в загальному статистичному розумінні відображає величину реальної системної розбіжності між станами об'єкта. В імітаційному моделюванні це дозволяє нам побудувати криві потужності не як статичні значення, а як динамічну функцію:

$$Power = f(Shift)$$

Такий підхід дозволяє визначити поріг чутливості кожного методу: наскільки великим має бути реальний зсув у стохастичній системі, щоб обрана методика верифікації змогла його ідентифікувати з високою ймовірністю $(1 - \beta)$. Саме аналіз потужності відносно розміру ефекту дає змогу виявити приховану неадекватність класичних методів у складних умовах.

В цьому контексті порівнюється ефективність критеріїв: асимптотичний t-критерій базується на припущенні, що при справедливості H_0 статистика має розподіл Стюдента, тоді як перестановочний тест буде точний розподіл статистики на основі ресемплінгу наявних даних. Порівняння проводиться за критерієм статистичної потужності – ймовірності відхилення (коректного) гіпотези H_0 , коли насправді вірна H_1 . Чим швидше крива потужності прямує до одиниці при збільшенні реальної різниці між μ_1 та μ_2 , тим вищою є здатність методу верифікувати адекватність моделі та уникати прихованих похибок II роду.

Для забезпечення об'єктивності порівняльного аналізу до розгляду було включено три типи статистичних інструментів, що представляють різні підходи до верифікації:

1. t-критерій Стюдента (еталон асимптотичного параметричного підходу);
2. Критерій Манна-Вітні (класичний непараметричний метод, що базується на рангових оцінках);
3. Перестановочний тест (метод ресемплінгу, що будує емпіричний розподіл).

Було проведено порівняльне дослідження адекватності цих критеріїв у специфічних стохастичних умовах, що зазвичай призводять до виникнення прихованих похибок. Методологія дослідження базувалася на імітації трьох критичних сценаріїв, що найчастіше зустрічаються в аналізі складних об'єктів: наявність «важких хвостів» у розподілах, малий обсяг вибірки та гетероскедастичність (неоднорідність дисперсій). Основним інструментом оцінки було обрано криві потужності, які дозволяють візуалізувати ймовірність правильного відхилення нульової гіпотези залежно від величини реального ефекту. Розрахунки проводилися в середовищі R.

3.1. Потужність критеріїв за умов наявності аномальних спостережень

Перший етап верифікації стосувався здатності методів ідентифікувати ефекти в системах із високою ймовірністю виникнення аномальних спостережень. Для генерації даних було обрано розподіл Коші, який є класичним прикладом розподілу з «важкими хвостами», для якого не визначені ні математичне сподівання, ні дисперсія. У контексті системного аналізу такі умови відповідають об'єктам із частими стрибкоподібними змінами стану або високим рівнем імпульсних завад.

Як свідчать результати моделювання (див. Рис. 1), у ситуації відхилення від нормальності на користь розподілу Коші спостерігається суттєва деградація t-критерію Стьюдента. Його крива потужності залишається практично горизонтальною навіть при значному збільшенні розміру ефекту (*Shift*, вісь абсцис), що свідчить про нездатність методу виявляти системні відмінності за даних умов. Навіть при максимальному розмірі ефекту (*Shift* = 2.0) його потужність ледь сягає 0.2. Отже, у 80% випадків системний зсув у подібних даних залишається непоміченим. Це пояснюється тим, що поодинокі великі викиди, характерні для розподілу Коші, надмірно роздувають значення вибіркової дисперсії, маскуючи реальний зсув центрів розподілу. Як наслідок, модель, верифікована лише асимптотичними методами в таких умовах, матиме приховану похибку II роду та ігноруватиме значущі зміни в об'єкті, вважаючи їх шумом.

Натомість ранговий критерій Манна-Вітні та перестановочний тест продемонстрували високу ефективність. Оскільки критерій Манна-Вітні базується на рангах, він ігнорує абсолютні значення викидів, зберігаючи стабільність. Потужність цього критерію демонструє стабільне зростання, досягаючи рівня ~0.9 при значенні ефекту *Shift* = 2.0. Це підтверджує його стійкість до відхилень від нормальності.

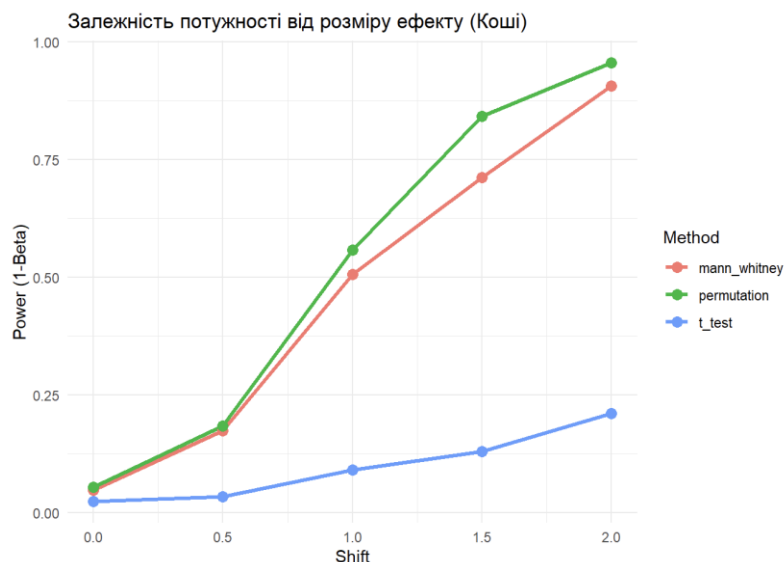


Рис.1. Криві потужності трьох критеріїв (асимптотичного t-критерію Стьюдента, рангового критерію Манна-Вітні та перестановочного критерію) залежно від розміру ефекту

Найвищу потужність продемонстрував саме перестановочний тест. Він дає найвищу чутливість до розміру ефекту на всьому діапазоні, найшвидше наближаючись до одиниці. Перестановочний тест виявився найкращим інструментом верифікації, оскільки будує точний розподіл статистики на основі конкретної реалізації даних, що дозволяє йому адаптуватися до будь-якої форми стохастичного шуму.

Отже, бачимо, що використання асимптотичних критеріїв для верифікації моделей у системах із «важкими хвостами» веде до системного ігнорування значущих факторів, тоді як методи ресемплінгу дозволяють забезпечити адекватність висновків навіть за відсутності теоретичних моментів розподілу. Саме перестановочний критерій забезпечує найменший поріг виявлення змін, мінімізуючи ризик пропустити критичний зсув у стані об'єкта.

3.2. Потужність критеріїв за умов обмеженої вибірки

Іншим важливим аспектом адекватності є робота в умовах дефіциту даних. При обсягах вибірок менше 25-30 одиниць, що є типовим при тестуванні медичних протоколів або пілотних стадій розробки інтелектуальних систем, асимптотичні наближення стають ненадійними, і відповідні методи втрачають контроль над помилкою першого роду, або ж їхня потужність стає недостатньою для верифікації складних гіпотез. Тому другий етап апробації методології був присвячений дослідженню залежності потужності критеріїв від обсягу вибірки.

Цей етап моделювання мав на меті дослідити динаміку зміни статистичної потужності критеріїв залежно від обсягу даних (n) та оцінити швидкість їхньої асимптотичної збіжності до одиниці за умов фіксованої альтернативи ($Shift = 1$). Для отримання об'єктивної системної оцінки було зіставлено дві радикально відмінні стохастичні ситуації: ідеальну теоретичну модель (нормальний розподіл) та модель із критичним рівнем зашумлення (розподіл Коші з «важкими хвостами»). В усіх експериментах фіксувався стабільний ефект системного зсуву. Результати імітаційного експерименту, а саме отримані криві залежності статистичної потужності асимптотичного, рангового та перестановочного критеріїв від обсягу вибірки в діапазоні від 10 до 50 спостережень, представлено на Рис. 2 та Рис.3.

Як демонструє графік на Рис. 2, в умовах суворого виконання припущення про нормальність параметричний t-критерій Стюдента (синя лінія) очікувано виступає як рівномірно найпотужніший критерій,

демонструючи найвищу статистичну потужність на всьому інтервалі обсягу вибірок. Його крива першою перетинає загальноприйнятий у системному аналізі поріг надійності верифікації (≈ 0.81) вже при $n = 20$.

Ранговий критерій Манна-Уїтні (червона лінія) та перестановочний критерій (зелена лінія) повторюють загальну траєкторію, проте мають певне відставання за потужністю на малих вибірках ($n < 30$). Це відображає класичне теоретичне положення про те, що непараметричні методи та процедури ресемплінгу мають дещо меншу асимптотичну відносну ефективність (близько 95%) порівняно з оптимальним параметричним критерієм, якщо припущення про нормальність суворо виконується.

Ситуація кардинально змінюється, коли система починає генерувати аномалії у вигляді важких хвостів (Рис. 3). За цих умов спостерігається «статистична сліпота» критерію Стюдента, його потужність для всіх значень розміру вибірки знаходиться на критично низькому рівні ($\approx 0.05 - 0.08$) і не реагує на збільшення обсягу вибірки. Поодинокі екстремальні викиди повністю руйнують вибірккову дисперсію, роблячи класичний параметричний аналіз абсолютно неадекватним.

Натомість перестановочний критерій (зелена лінія) демонструє максимальну потужність на всьому інтервалі, випереджаючи навіть ранговий критерій Манна-Уїтні. Уже при $n = 50$ ресемплінг забезпечує цільову потужність верифікації на рівні 0.86.

Результати моделювання демонструють ціну універсальності методів ресемплінгу. За ідеальних умов істинної нормальності даних використання перестановочного тесту призводить до незначної втрати потужності на малих вибірках (для досягнення аналогічної з t-критерієм потужності ресемплінгу потрібно в середньому на 3–5 спостережень більше). Проте за наявності реальних системних аномалій, де класичний критерій Стюдента стає ненадійним, ресемплінг виявляється найкращим інструментом, здатним забезпечити адекватну верифікацію.

Оскільки в реальних задачах дослідник майже ніколи не має апіорної гарантії нормальності, при верифікації моделей машинного навчання на обмежених датасетах використання лише асимптотичних критеріїв є ризикованим. Системний підхід вимагає свідомого компромісу між потенційною високою потужністю параметричних тестів та гарантованою надійністю ресемплінгу, і часто – саме вибору на користь методів ресемплінгу як стратегії мінімізації ризиків (помилки II роду).

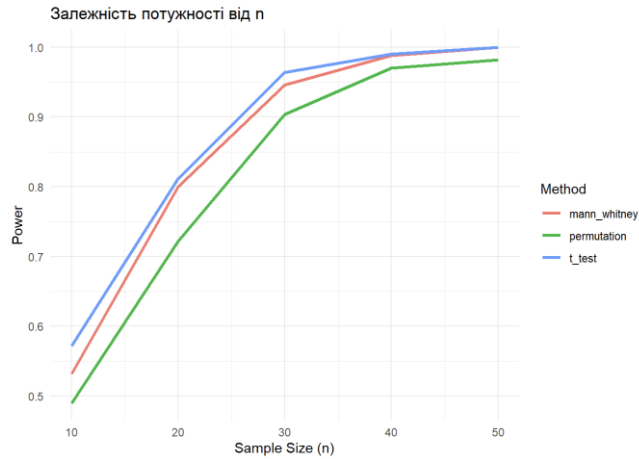


Рис. 2. Криві потужності асимптотичного t-критерію Стюдента, рангового критерію Манна-Вітні та перестановочного критерію залежно від розміру вибірки за умов нормального розподілу даних та фіксованого ефекту $Shift = 1$. Графік ілюструє класичну теоретичну перевагу параметричного t-критерію як оптимального для нормальних даних, та демонструє межі відставання за потужністю для методів ресемплінгу в ідеалізованих умовах.

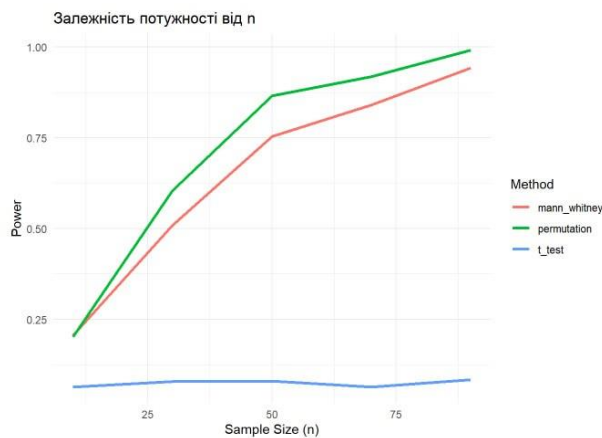


Рис. 3. Криві потужності асимптотичного t-критерію, рангового критерію Манна-Вітні та перестановочного критерію залежно від розміру вибірки за умов важких хвостів (розподіл Коші) для фіксованого ефекту $Shift = 1$.

Графік демонструє неефективність та інваріантність t-критерію до збільшення обсягу даних через вплив екстремальних викидів, тоді як перестановочний критерій забезпечує найвищу швидкість зростання потужності верифікації в цих умовах

3.3. Потужність критеріїв за умов гетероскедастичності та неоднорідності форми розподілів

Третій сценарій стосується найбільш наближеної до реальності ситуації, коли стохастичний об'єкт генерує дані з різною мінливістю та різними законами розподілу в межах однієї системи. Такий підхід комбінує два складні чинники: гетероскедастичність та неоднорідність розподілів. Ефект гетероскедастичності – наявність різної дисперсії у групах порівняння – є прямим індикатором нестабільності стохастичної системи. Це критична точка для верифікації, оскільки більшість класичних критеріїв вимагають гомоскедастичності (однорідності дисперсій) для коректної роботи.

Для цього сценарію встановлено розмір вибірки $n = 25$ та закладено фундаментальну розбіжність: порівнювані вибірки генерувалися за різними законами розподілу, хоча їхні центри зміщення були штучно вирівняні (встановлено $Shift = 0$). Таким чином ми припускаємо, що математичні сподівання груп рівні, проте характер їхньої стохастичної поведінки та різниця в дисперсіях (яка зростає пропорційно квадрату параметра $Scale$) суттєво відрізняються.

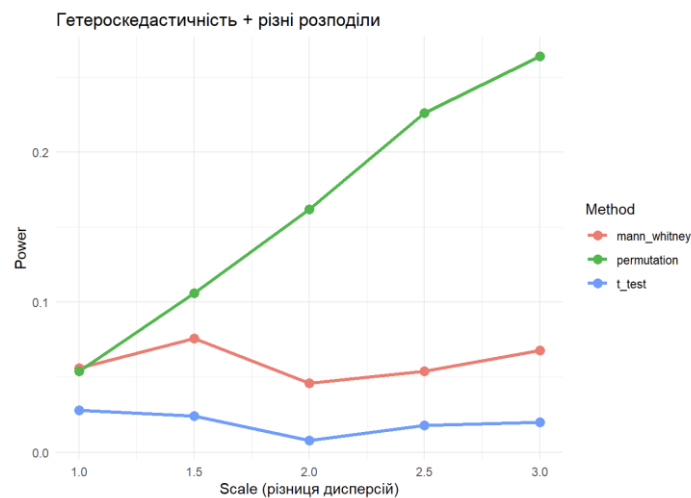


Рис. 4. Криві потужності трьох критеріїв (асимптотичного t-критерію Стюдента, рангового критерію Манна-Вітні та перестановочного критерію) за умов відсутнього ефекту $Shift = 0$ та зростаючого відношення стандартних відхилень: $Scale = \sigma_1/\sigma_2$. За умов відсутності ефекту по осі ординат відкладається ймовірність похибки I роду

За таких припущень ми фактично тестуємо критерії на номінальний рівень значущості. У такому сценарії значення по осі ординат на Рис. 4 відображають не потужність у класичному розумінні, а ймовірність помилки I роду – ризик того, що методика верифікації зафіксує «фантомні» відмінності, спричинені лише різницею в дисперсіях чи формі розподілів. Зростання лінії перестановочного тесту свідчить про те, що при збільшенні гетероскедастичності критерій стає надчутливим: він ідентифікує різницю у структурі даних як статистично значущу, навіть за умови рівності середніх.

Отже, перестановочний критерій у цьому специфічному сценарії виявляється чутливим до самої архітектури розсіювання даних. Оскільки він базується на припущенні про повну ідентичність розподілів (гіпотеза обмінності), будь-яка системна неоднорідність призводить до відхилення нульової гіпотези. Натомість асимптотичні критерії продемонстрували формальну стабільність, фокусуючись виключно на агрегованих показниках (середніх або рангах). У контексті системного аналізу така «стабільність» може бути небезпечною, оскільки вона маскує зміну самого характеру функціонування об'єкта.

Зауважимо, що в цьому контексті зростання ординати для перестановочного тесту слід трактувати не як методичну помилку, а як здатність методу виявляти системну неоднорідність. Ресемплінг діє як чутливий індикатор, що сигналізує про неідентичність порівнюваних станів системи за їхньою внутрішньою структурою. Для системного аналітика це критичний сигнал про те, що модель машинного навчання стикається з якісно іншим типом даних, що вимагає перегляду стратегії верифікації.

4. Практична верифікація діагностичної моделі

Для валідації запропонованого підходу було використано результатів клінічних досліджень пацієнтів із підозрою на ішемічну хворобу серця. Для аналізу використано відкритий набір даних Cleveland Heart Disease Database, що є частиною репозиторію машинного навчання UCI (UCI Machine Learning Repository). Даний датасет є загальнодоступним (open-access) і широко використовується у науковій спільноті для тестування алгоритмів класифікації та верифікації прогностичних моделей ([7]).

Аналізувався вплив рівня холестерину (параметр chol) на наявність серцевої патології. Як демонструє візуалізація щільності розподілу (Рис. 5), дані мають складну стохастичну структуру: помітну асиметрію, наявність

екстремальних викидів у правій частині спектра (значення > 400 мг/дл) та суттєве перекриття областей розподілу для здорових і хворих пацієнтів.

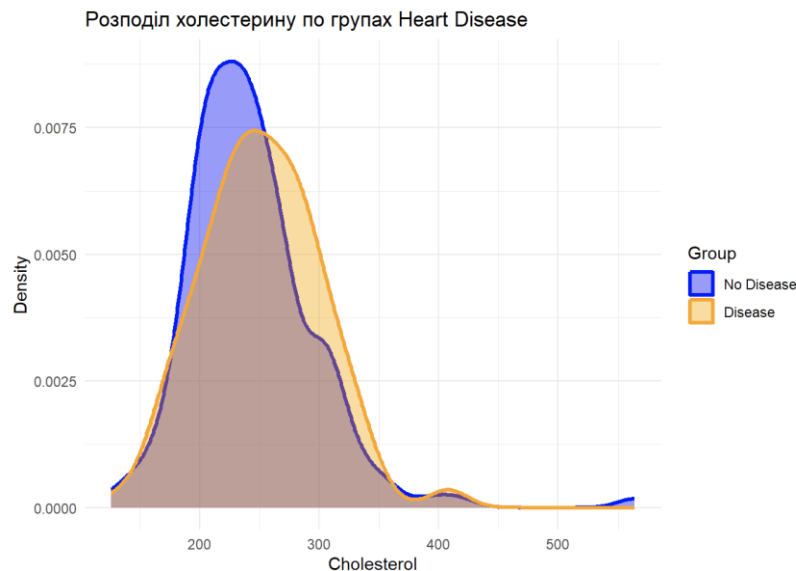


Рис. 5. Емпіричні щільності розподілу рівня холестерину (параметр chol) для груп пацієнтів із серцевою патологією та без неї (на основі датасету Cleveland Heart Disease)

Верифікація відмінностей між групами пацієнтів (з діагностованою хворобою та без неї) за допомогою різних статистичних підходів дала наступні результати:

- Асимптотичний t-критерій: Отримане значення $p - value = 0.165$ є статистично незначущим за стандартного порогу значущості $\alpha = 0.05$, отже є підстави прийняти нульову гіпотезу про відсутність системних відмінностей, що у медичному контексті означало б пропуск важливого діагностичного маркера (похибка II роду). Це пояснюється тим, що поодинокі викиди (пацієнти з рівнем холестерину > 400 мг/дл) надмірно збільшують стандартну похибку середнього, розмиваючи реальний ефект.

- Ранговий непараметричний підхід (Wilcoxon test): Значення $p - value = 0.0467$ знаходиться на межі прийняття рішення. Хоча метод виявився стійкішим за t-критерій завдяки роботі з рангами, він демонструє

низький запас міцності верифікації, що в системному аналізі вважається ознакою статистичної крихкості висновку.

- Перестановочний тест: Використання ресемплінг-процедури в межах нашої методології дало значення $p - value = 0.007$ вказує на високу статистичну ймовірність розбіжностей.

Отже, перестановочний тест виявився єдиним інструментом, який зміг чітко відокремити системний сигнал від стохастичного шуму в умовах реальних медичних даних. Якщо асимптотичний тест веде до помилки II роду, а ранговий метод дає непевну оцінку, то ресемплінг забезпечує впевнену верифікацію адекватності моделі. Це наочно ілюструє перевагу системної верифікації над стандартними перевірками.

5. Висновки

У роботі розроблено та обґрунтовано системну методологію статистичної верифікації моделей машинного навчання, що базується на аналізі потужності критеріїв та використанні методів ресемплінгу. Проведений порівняльний аналіз дозволяє сформулювати наступні висновки.

1. Імітаційне моделювання підтвердило, що класичні параметричні критерії (t-тест Стьюдента) демонструють критичну втрату потужності в умовах відхилення від нормальності (зокрема, з «важкими хвостами» типу Коші) та при малих обсягах вибірок. Це створює ризик системних помилок II роду, коли значущі зміни у стані об'єкта ігноруються через невідповідність даних теоретичним припущенням.

2. Перестановочний тест продемонстрував найвищу стійкість та чутливість до розміру ефекту в усіх критичних сценаріях. На відміну від рангових методів, ресемплінг дозволяє працювати безпосередньо з метриками системи, забезпечуючи при цьому точний контроль рівня значущості навіть у нестационарних умовах.

3. Встановлено, що в умовах сильної гетероскедастичності та змішаних розподілів методи ресемплінгу діють як чутливий індикатор структурних змін. На відміну від асимптотичних критеріїв, які фокусуються лише на агрегованих середніх, запропонована методологія ідентифікує розбіжності у самій архітектурі стохастичного шуму, що є критичним для верифікації моделей машинного навчання.

4. Апробація на реальних даних з репозиторію UCI (Cleveland Heart Disease) наочно продемонструвала перевагу розробленого підходу. У ситуації, де асимптотичний підхід не виявив значущих розбіжностей, а ранговий тест дав непевну оцінку на межі значущості, використання

перестановочного тесту дозволило впевнено ідентифікувати системний сигнал.

5. Інтеграція статистичної верифікації в життєвий цикл ML-моделей трансформує процес оцінки з суто обчислювальної задачі у фундаментальну процедуру контролю адекватності складної стохастичної системи.

Запропонована методологія може бути рекомендована для використання в інтелектуальних системах підтримки прийняття рішень, медичній діагностиці та при верифікації складних стохастичних моделей, де надійність висновків є пріоритетною над обчислювальною складністю.

Список використаних джерел

1. Higgins, J. J. An introduction to modern nonparametric statistics. Pacific Grove, CA: Brooks/Cole, 2004.
2. Wasserman, L. All of Nonparametric Statistics. – Springer-Verlag New York, 2006.
3. Efron, B. and Tibshirani, R.J. (1994) An Introduction to the Bootstrap. Chapman and Hall/CRC, UK.
4. Good, P. Permutation, Parametric, and Bootstrap Tests of Hypotheses. 3-rd ed., Springer, 2005.
5. Steyerberg, E. W., et al. Clinical Prediction Models: A Practical Approach to Development, Validation, and Updating. Springer; 2019.
6. Kuhn, M., & Johnson, K. Applied Predictive Modeling. Springer; 2013.
7. Heart Disease Data Set [Електронний ресурс] / A. Janosi, W. Steinbrunn, M. Pfisterer, R. Detrano // UCI Machine Learning Repository. – 1988. – Режим доступу: <https://archive.ics.uci.edu/dataset/45/heart+disease>

*Шарапов Михайло Михайлович, к.ф.-м.н, доцент,
Розора Ірина Василівна, д.ф.-м.н, доцент,
Лівінська Ганна Володимирівна, к.ф.-м.н, доцент,
Пономарьов Вадим Дмитрович, к.ф.-м.н, доцент,
Моренець Володимир Ігорович, к.ф.-м.н, асистент,
Макушенко Ігор Анатолійович, к.ф.-м.н, асистент*

ФОРМУВАННЯ ІМОВІРНІСНО-СТАТИСТИЧНОГО МИСЛЕННЯ ЯК БАЗИСУ ПІДГОТОВКИ ФАХІВЦІВ З ІТ В МЕЖАХ КУРСУ «ТЕОРІЯ ЙМОВІРНОСТЕЙ ТА МАТЕМАТИЧНА СТАТИСТИКА»

Роботу присвячено методиці формування імовірно-статистичного мислення у майбутніх ІТ-фахівців у межах курсу «Теорія ймовірностей та математична статистика». Описано комплексний підхід кафедри Прикладної статистики, що поєднує фундаментальну математику з програмуванням. Розглянуто інтерактивний формат занять «Доповідь», спрямований на розвиток аналітичних і комунікативних навичок. Акцентовано увагу на інтеграції сучасних мов програмування у навчальні веб-ресурси та використанні авторських прикладних проєктів («Індекс», «Steganeisodos»), які розвивають критичне мислення, культуру верифікації результатів і розуміння ролі статистики в кібербезпеці.

Ключові слова: імовірно-статистичне мислення, синергія математики та програмування, верифікація результатів, компетентнісний підхід.

The work is dedicated to the methodology of forming probabilistic-statistical thinking in future IT specialists within the course "Probability Theory and Mathematical Statistics". The comprehensive approach of the Department of Applied Statistics, which combines fundamental mathematics with programming, is described. The interactive class format "Report", aimed at developing analytical and communication skills, is considered. Emphasis is placed on the integration of modern programming languages into educational web resources and the use of the author's applied projects ("Index", "Steganeisodos"), which develop critical thinking, a culture of results verification, and an understanding of the role of statistics in cybersecurity.

Keywords: probabilistic-statistical thinking, synergy of mathematics and programming, results verification, competence-based approach.

Вступ

Сучасний етап розвитку інформаційних технологій вимагає від фахівців не лише навичок написання програмного коду, але й глибокого розуміння математичних моделей, що лежать в основі алгоритмів штучного інтелекту, аналізу даних (Data Science), кібербезпеки та комп'ютерного моделювання. Таке поєднання фундаментальної науки та ІТ повністю відповідає вектору Стратегічного плану розвитку Київського національного університету імені Тараса Шевченка, де наголошується на необхідності розбудови комплексної системи міждисциплінарних програм, які формують цілісний світогляд здобувачів та забезпечують міцний зв'язок дослідницької роботи з практичною спрямованістю [1, с. 14]. У цьому контексті дисципліна «Теорія ймовірностей та математична статистика» перестає бути суто теоретичним математичним інструментом і перетворюється на фундаментальну базу для формування специфічного імовірнісно-статистичного мислення ІТ-спеціаліста.

Концепцію такого комплексного підходу до навчання успішно реалізує кафедра Прикладної статистики факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка. Здобуті студентами знання, вміння та навички під час вивчення базового курсу на другому році навчання стають надійним підґрунтям для опанування більш спеціалізованих дисциплін на старших курсах, таких як «Криптографія», «Ймовірнісні основи імітаційного моделювання», «Математична демографія» тощо. Компетентнісний підхід, впроваджений кафедрою, передбачає органічне поєднання класичної математичної теорії з сучасними методами програмування, розвитком навичок самостійної роботи та критичного аналізу отриманих результатів.

Теоретичний базис курсу та формування математичної логіки розв'язування задач

Навчальна програма курсу розрахована на два семестри та охоплює широкий спектр тем, необхідних для формування цілісного розуміння стохастичних процесів. У першому семестрі студенти проходять шлях від базових понять комбінаторики та аксіоматики теорії ймовірностей до складних концепцій, таких як багатовимірні генератриси, характеристичні функції та ланцюги Маркова. Другий семестр присвячений вивченню закону великих чисел, центральної граничної теореми, випадкових векторів та, власне, математичній статистиці: від точкового та інтервального оцінювання параметрів до перевірки параметричних і непараметричних гіпотез.

Однак, як свідчить педагогічний досвід, засвоєння потужного теоретичного апарату є лише першим етапом навчання. Найбільшим когнітивним бар'єром для студентів ІТ-спеціальностей часто стає етап формалізації – перехід від текстового формулювання практичної проблеми до її математичної моделі.

Для вирішення цієї проблеми на навчальному сайті доцента кафедри Шарапова М. М. створено спеціальний розділ «Логіка розв'язування задач» [2]. Цей ресурс спрямований на розвиток алгоритмічного підходу до аналізу умови задачі. Студентів навчають розкладати складний текст на елементарні складові, ідентифікувати ключові події та випадкові величини, правильно обирати ймовірнісні схеми (наприклад, розпізнавати схему незалежних випробувань Бернуллі чи марківський процес). Такий методичний прийом дозволяє трансформувати абстрактні математичні формули у зрозумілий для майбутнього програміста алгоритм дій. Здатність побудувати коректну математичну модель є критично важливою навичкою, оскільки помилка на етапі формалізації призводить до написання програмного коду, який ідеально компілюється, але вирішує хибну задачу.

Крім того, наявність у відкритому доступі детально розв'язаних прикладів [3] дозволяє студентам самостійно аналізувати логіку застосування теорем та методів. Це формує у них звичку до структурованого мислення, яка є невіддільною складовою професійної діяльності розробника програмного забезпечення чи аналітика даних.

Практичний вимір курсу: формат «Доповідь» як інструмент розвитку професійних та комунікативних компетенцій

Традиційний підхід до проведення практичних занять, що зводиться до механічного розв'язування задач біля дошки, не повною мірою відповідає потребам підготовки сучасних ІТ-фахівців. Зважаючи на це, в межах курсу було впроваджено спеціальний тип завдань для студентів – «Доповідь».

Кожна доповідь являє собою глибоко опрацьований та належним чином оформлений розв'язок 2-3 задач. Однак головна мета цього формату полягає не просто у демонстрації правильної відповіді, а у формуванні комплексу навичок (hard та soft skills). Студент має продумати чіткий план презентації, за яким у жорстко лімітований час (5-7 хвилин) він зможе максимально логічно і доступно пояснити хід розв'язку так, щоб його зрозуміли всі слухачі в аудиторії. При цьому доповідач вчиться робити акцент на найважливіших алгоритмічних та концептуальних аспектах, свідомо уникаючи рутинних технічних розрахунків, які перевантажують сприйняття.

Особлива увага під час доповіді приділяється верифікації отриманих даних: студент зобов'язаний пояснити, які саме види перевірок результату він здійснив, та обґрунтувати вибір джерел інформації. Формат передбачає активну інтерактивну взаємодію – доповідач має бути готовим відповідати на фахові запитання як викладача, так і своїх одногрупників. Для перевірки засвоєння матеріалу аудиторією, студент заздалегідь готує кілька контрольних запитань до слухачів. Такий підхід стимулює уважність групи та розвиває у доповідача навички технічного менторства, що є вкрай важливим для майбутньої роботи в IT-командах (Teamwork).

Важливою складовою цього процесу є можливість отримати попередню консультацію. Після підготовки чернетки розв'язку студент може звернутися до викладача для перевірки матеріалу на відсутність помилок і отримання методичних порад щодо оптимізації самої доповіді. Це формує культуру відповідального ставлення до публічної презентації результатів своєї роботи.

Синергія математики та програмування: навчальне середовище та спеціалізовані задачі

Важливим кроком у формуванні імовірісно-статистичного мислення є безпосередня інтеграція математичних методів у площину програмування. Для забезпечення цього процесу кафедрою Прикладної статистики було розроблено та видано спеціалізований збірник задач з теорії ймовірностей та математичної статистики [4]. Його унікальністю є наявність окремого розділу, присвяченого завданням для програмістів. Усі задачі з програмування у цьому розділі органічно пов'язані з імовірнісними моделями та статистичним аналізом, що дозволяє студентам на практиці побачити цінність математичного апарату для написання ефективного коду.

Крім того, активна самостійна робота студентів підтримується розгалуженою інфраструктурою навчального сайту доцента Шарапова М. М. [5]. На ресурсі представлена значна база прикладів розв'язаних задач, проте студенти мають можливість не лише вивчати готовий матеріал, а й самостійно створювати контент. У межах системи заохочення (отримання додаткових балів) студенти розробляють власні задачі для публікації на сайті. Ці завдання часто включають елементи програмування, спрямовані на створення скриптів для комп'ютерної перевірки (симуляції) отриманих теоретичних результатів. Фактично, на цьому етапі студенти здобувають базовий досвід роботи у парадигмі Data Science: вони вчаться генерувати вибірки, обробляти масиви даних та

інтерпретувати результати симуляцій, що формує стійкі навички підготовки даних (Data Preparation) для їх подальшого аналізу. Такий підхід є особливо корисним у теорії ймовірностей, де метод Монте-Карло чи імітаційне моделювання дозволяють наочно підтвердити правильність аналітичних розрахунків.

Варто відзначити і технологічну гнучкість навчального ресурсу. При оформленні онлайн-прикладів як викладач, так і студенти мають змогу використовувати сучасні методи інтеграції програмного коду. Завдяки цьому код на більшості сучасних мов програмування (зокрема Python, який є стандартом де-факто у сфері Data Science), а не лише на JavaScript, може додаватися безпосередньо в HTML-код сторінки. Приклади такого сучасного впровадження скриптів та їх виконання у браузері представлені у відповідному розділі сайту. Додатково функціонує онлайн-лабораторія, яка містить корисні скрипти за напрямками «Теорія ймовірностей», «Статистика», «Моделювання випадкових величин», «Шифрування» тощо. Це створює повноцінне інтерактивне середовище, де студент одночасно виступає і як математик-аналітик, і як software-інженер.

Прикладні веб-проекти як середовище для формування критичного мислення та відповідальності

Важливим етапом закріплення здобутих знань, умінь та навичок є залучення студентів до роботи з реальними прикладними проектами, що дозволяє наочно продемонструвати перетворення абстрактних математичних концептів на дієві IT-рішення.

Яскравим прикладом такої інтеграції є проєкт «Steganeisodos» [6], започаткований доцентом кафедри Шараповим М. М. Цей ресурс переслідує подвійну мету. З одного боку, це потужний навчальний інструмент, який демонструє застосування методів математичної статистики для створення додаткового захисту веб-контенту. Робота проєкту базується на стратегії NTR (ідентифікація за природним ритмом друку користувача), де статистичний аналіз часових інтервалів між натисканнями клавіш стає основою для інноваційних систем автентифікації. З іншого боку, ознайомлення з цим ресурсом стимулює розвиток практичних навичок студентів як майбутніх програмістів та фахівців з кібербезпеки, показуючи їм нестандартні вектори застосування статистичного апарату у реальних умовах.

Не менш значущим для формування професійної етики та інженерного підходу є онлайн-проєкт «Індекс» [7]. Його фундаментальна філософія полягає не просто у наданні зручної платформи для виконання онлайн-

завдань. Найголовніша суть «Індексу» – у вихованні у студентів здатності та стійкої звички багаторазово перевіряти отримані результати своїх розрахунків та досліджень перед наданням остаточної відповіді.

У сучасній ІТ-індустрії (особливо у сферах Data Science, фінансових технологій чи проєктування високонавантажених систем) ціна помилки у математичній моделі може бути критичною. Тому робота з проєктом «Індекс» ефективно бореться з проблемою «сліпого вгадування», привчаючи майбутніх фахівців до жорсткого самоконтролю, використання методів крос-валідації та усвідомлення особистої відповідальності за фінальний результат своєї роботи.

Висновки

Таким чином, викладання дисципліни «Теорія ймовірностей та математична статистика» на кафедрі Прикладної статистики виходить далеко за межі класичного академічного підходу. Завдяки цілісній навчальній екосистемі, що поєднує ґрунтовну математичну теорію, інноваційний комунікативний формат практичних занять («Доповідь»), використання сучасних вебтехнологій для інтеграції програмного коду та залучення до авторських проєктів («Steganeisodos», «Індекс»), у студентів формується глибоке імовірісно-статистичне мислення.

Студенти отримують не лише сукупність формул і теорем, а реальні знання, вміння та навички, необхідні для розв'язання нетипових завдань. Зрештою, такий багатогранний базис підготовки дозволяє випускати на ринок праці не просто рядових «кодерів», які механічно виконують технічні завдання, а висококваліфікованих інженерів-аналітиків, здатних критично мислити, проєктувати складні інформаційні системи та ефективно працювати у сучасних ІТ-командах.

Список використаних джерел

1. Стратегічний план розвитку Київського національного університету імені Тараса Шевченка на 2025–2032 роки / Київський національний університет імені Тараса Шевченка. Київ, 2024. URL: <https://knu.ua/pdfs/official/Development-strategic-plan-2025-2032.pdf> (дата звернення: 28.04.2026).
2. Шарапов М. М. Логіка розв'язування задач. Теорія ймовірностей та математична статистика: навчальний сайт. URL: <https://teorver.pp.ua/ukr/ukr.php?templ=how2> (дата звернення: 28.04.2026).

3. Шарапов М. М. Приклади розв'язування задач. Теорія ймовірностей та математична статистика: навчальний сайт. URL: https://teorver.pp.ua/ukr/ukr.php?templ=tv_examples (дата звернення: 28.04.2026).
4. Шарапов М.М., Розора І.В., Лівінська Г.В., Пономарьов В.Д. Збірник задач з теорії ймовірностей та математичної статистики / Київський національний університет імені Тараса Шевченка, Кафедра прикладної статистики. Київ, 2023. URL: https://teorver.pp.ua/ukr/met/main_2023.pdf (дата звернення: 28.04.2026).
5. Шарапов М. М. Теорія ймовірностей та математична статистика: навчальний сайт. URL: <https://teorver.pp.ua/ukr/ukr.php?templ=tv> (дата звернення: 28.04.2026).
6. Шарапов М. М. Steganeisodos: [вебпроект]. URL: <https://mmsharapov.github.io/NTR/> (дата звернення: 28.04.2026).
7. Шарапов М. М. Індекс: онлайн-програма перевірки практичних знань. URL: <https://indexator.pp.ua/> (дата звернення: 28.04.2026).

*Моренець Володимир Ігорович, к.ф.-м.н, асистент
Пономарьов Вадим Дмитрович, к.ф.-м.н, доцент
Шарапов Михайло Михайлович, к.ф.-м.н, доцент*

МЕТОДИ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ ПРИ АВТОМАТИЗОВАНОМУ ЗБОРІ ІНФОРМАЦІЇ З ВІДКРИТИХ ДЖЕРЕЛ

Роботу присвячено методичним підходам до викладання технологій автоматизованого збору інформації з відкритих вебджерел та методів інтелектуального аналізу даних у процесі підготовки фахівців з аналізу даних. Розглянуто особливості використання вебскрейпінгу, API та засобів попередньої обробки відкритих даних у навчальному процесі. Запропоновано pipeline-орієнтований підхід до організації навчання, що інтегрує етапи збору, очищення, аналізу та візуалізації даних у єдиний практично орієнтований процес. Описано структуру навчального курсу, принципи організації практичних проєктів та підходи до оцінювання результатів навчання. Показано, що використання реальних відкритих даних сприяє формуванню у студентів навичок роботи з гетерогенними даними та підвищує ефективність підготовки у сфері аналізу даних.

Ключові слова: відкриті дані, вебскрейпінг, інтелектуальний аналіз даних, API, pipeline, обробка даних.

The work is devoted to methodological approaches to teaching technologies for automated information collection from open web sources and methods of intelligent data analysis in the training of data analysis specialists. The paper considers the use of web scraping, APIs, and open data preprocessing tools within the educational process. A pipeline-oriented approach to course organization is proposed, integrating data collection, cleaning, analysis, and visualization into a unified practice-oriented workflow. The structure of the educational course, principles of practical project organization, and approaches to learning outcome assessment are described. It is shown that the use of real open datasets contributes to the development of students' skills in working with heterogeneous data and improves the effectiveness of training in the field of data analysis.

Keywords: open data, web scraping, intelligent data analysis, API, pipeline, data processing.

Вступ

Розвиток цифрового суспільства характеризується стрімким зростанням обсягів відкритих даних та активним використанням вебресурсів як джерел інформації для аналітичних досліджень, прогнозування та підтримки прийняття рішень. Значна частина таких даних представлена у вигляді вебсторінок, API-сервісів, відкритих державних порталів та інших гетерогенних джерел, що потребують автоматизованих підходів до збору, обробки та аналізу інформації. У зв'язку з цим особливого значення набувають технології вебскрейпінгу та методи інтелектуального аналізу даних.

Автоматизований збір інформації з відкритих вебджерел активно використовується у сфері соціально-економічного аналізу, маркетингових досліджень, моніторингу інформаційних ресурсів та інших прикладних задачах аналізу даних [1–3]. Водночас робота з відкритими вебданими характеризується гетерогенністю форматів, нестабільністю структури вебресурсів, наявністю пропусків та дублювань. Це вимагає від фахівців не лише знань мов програмування та статистичних методів, але й навичок автоматизованого збору, очищення, інтеграції та аналізу даних.

У підготовці студентів спеціальностей, пов'язаних з аналізом даних та прикладною статистикою, важливим є поєднання теоретичних знань із практично орієнтованими підходами до роботи з реальними відкритими даними. Традиційні навчальні курси часто зосереджуються окремо на програмуванні або статистичному аналізі, тоді як сучасні задачі Data Science потребують комплексного підходу, що інтегрує автоматизований збір даних, їх попередню обробку, аналіз та візуалізацію результатів.

Одним із перспективних підходів до організації навчального процесу є pipeline-орієнтована модель навчання, у межах якої студенти послідовно проходять усі етапи роботи з даними: від пошуку відкритих джерел та вебскрейпінгу до очищення, аналізу та представлення результатів. Такий підхід дозволяє формувати системне розуміння процесів роботи з даними та забезпечує практичну спрямованість навчання.

Метою роботи є розробка методичних підходів до викладання технологій автоматизованого збору інформації з відкритих вебджерел та методів інтелектуального аналізу даних у процесі підготовки фахівців з аналізу даних.

Особливості автоматизованого збору інформації з відкритих вебджерел

Автоматизований збір інформації з відкритих вебджерел є важливою складовою сучасних процесів аналізу даних та побудови інформаційно-аналітичних систем. Зростання кількості відкритих вебресурсів, електронних платформ та державних порталів відкритих даних створює значні можливості для отримання актуальної інформації у різних сферах діяльності. Водночас великі обсяги відкритих даних та постійне оновлення інформації роблять ручний збір даних малоефективним, що зумовлює необхідність використання автоматизованих підходів.

Основними джерелами відкритих даних є вебсайти, API-сервіси, відкриті державні реєстри, електронні каталоги та інформаційні портали. На відміну від традиційних структурованих баз даних, вебджерела часто мають неоднорідну структуру та різні формати представлення інформації. Дані можуть бути представлені у вигляді HTML-документів, JSON- або XML-файлів, текстових блоків, таблиць та мультимедійного контенту. Така гетерогенність ускладнює процес автоматизованого збору та подальшої аналітичної обробки інформації.

Одним із найбільш поширених підходів до автоматизованого збору інформації є вебскрейпінг, що передбачає отримання даних безпосередньо зі структури вебсторінок. Для реалізації вебскрейпінгу використовуються технології аналізу HTML-документів, DOM-структури, CSS-селекторів та XPath-запитів. У сучасних умовах важливу роль також відіграє використання API, які забезпечують доступ до структурованих даних у стандартизованих форматах та значно спрощують процес інтеграції інформації з різних джерел.

Процес автоматизованого збору інформації включає кілька основних етапів: пошук джерел даних, отримання інформації, попередню обробку, очищення, інтеграцію та підготовку даних до подальшого аналізу. Узагальнену pipeline-модель роботи з відкритими вебданими наведено на рис. 1.



Рис.1. Pipeline-модель роботи з відкритими вебданими

Особливістю відкритих вебданих є наявність пропущених значень, дублювань, різних форматів запису та нестабільності структури вебресурсів. Це потребує застосування методів попередньої обробки та очищення даних перед виконанням аналітичних задач. Крім того, важливими аспектами роботи з відкритими вебджерелами є дотримання етичних та правових принципів використання даних, врахування обмежень доступу до вебресурсів та коректне використання API-сервісів.

Таким чином, автоматизований збір інформації з відкритих вебджерел є комплексним процесом, що поєднує використання вебтехнологій, методів обробки даних та аналітичних підходів. Формування у студентів практичних навичок роботи з відкритими вебданими є важливою складовою сучасної підготовки фахівців у сфері аналізу даних та Data Science.

Методи інтелектуального аналізу даних у навчальному процесі

Важливою складовою підготовки фахівців у сфері аналізу даних є використання методів інтелектуального аналізу даних у процесі роботи з відкритими вебджерелами. На відміну від традиційних навчальних прикладів

із невеликими структурованими наборами даних, відкриті вебдані характеризуються значним обсягом, неоднорідністю структури та наявністю шуму, що потребує застосування сучасних методів обробки та аналізу інформації.

Одним із перших етапів роботи з даними є їх попередня обробка. У процесі навчання студенти виконують очищення наборів даних, усунення дублікатів, нормалізацію форматів запису, обробку пропущених значень та інтеграцію інформації з різних джерел. Такий підхід дозволяє сформувати практичні навички підготовки даних до подальшого аналітичного опрацювання та демонструє важливість якості даних для отримання коректних результатів аналізу.

Після етапу попередньої обробки застосовуються методи інтелектуального аналізу даних, що дозволяють виявляти закономірності у великих масивах інформації. У межах навчального процесу студенти опановують методи кореляційного аналізу, кластеризації, виявлення аномалій, аналізу часових рядів та побудови візуалізацій. Використання таких методів дає можливість аналізувати структуру даних, визначати взаємозв'язки між показниками та формувати аналітичні висновки на основі реальних наборів відкритих даних.

Одним із важливих елементів навчального процесу є оцінювання результатів виконання практичних проєктів. Для комплексного оцінювання рівня виконання студентських робіт може використовуватись інтегральний показник якості проєкту:

$$Q(S) = w_1 Q_{code} + w_2 Q_{data} + w_3 Q_{analysis} + w_4 Q_{report} , \quad (1)$$

де Q_{code} - якість програмної реалізації, Q_{data} - якість підготовки та обробки даних, $Q_{analysis}$ - рівень аналітичної обробки інформації, Q_{report} - якість представлення результатів дослідження, а w_i - вагові коефіцієнти відповідних критеріїв оцінювання.

Структуру критеріїв оцінювання практичних проєктів наведено у таблиці 1.

Таблиця 1. Критерії оцінювання практичних проєктів студентів

Критерій	Зміст оцінювання
Якість коду	Коректність роботи програмного рішення та використання інструментів автоматизованого збору даних
Підготовка даних	Очищення, нормалізація та інтеграція наборів даних
Аналіз даних	Використання методів інтелектуального аналізу та інтерпретація результатів
Представлення результатів	Якість візуалізацій, аналітичного звіту та презентації результатів

Використання методів інтелектуального аналізу даних у навчальному процесі дозволяє поєднати автоматизований збір інформації з практичним аналізом реальних відкритих даних. Такий підхід сприяє розвитку у студентів навичок роботи з великими масивами даних, формує системне розуміння процесів аналітичної обробки інформації та забезпечує підготовку до практичної діяльності у сфері Data Science та аналізу даних.

Практичні аспекти використання відкритих вебданих у навчальному процесі

Використання відкритих вебданих у навчальному процесі дозволяє наблизити навчальні завдання до реальних практичних задач сучасної аналітики даних. На відміну від штучно сформованих навчальних наборів даних, відкриті вебджерела характеризуються динамічністю, неоднорідністю структури та необхідністю постійного оновлення інформації. Це створює умови для формування у студентів практичних навичок роботи з реальними даними та сучасними інструментами автоматизованого аналізу інформації.

У межах навчального процесу студенти працюють із відкритими API, державними порталами даних, статистичними ресурсами та інформаційними вебплатформами. Практичні завдання охоплюють аналіз структури вебресурсів, автоматизований збір інформації, підготовку наборів даних та проведення аналітичного опрацювання результатів. Особлива увага

приділяється роботі з даними, що мають неоднорідний формат представлення та потребують попереднього очищення й нормалізації.

Практична реалізація завдань передбачає використання сучасних засобів аналізу даних та мов програмування для автоматизації процесів збору й обробки інформації. У процесі виконання проєктів студенти реалізують сценарії автоматизованого оновлення даних, інтеграції інформації з кількох джерел та побудови аналітичних візуалізацій. Такий підхід дозволяє формувати навички побудови інформаційно-аналітичних систем та роботи з великими масивами відкритих даних.

Однією з переваг використання відкритих вебджерел у навчальному процесі є можливість адаптації практичних завдань до сучасних інформаційних ресурсів та актуальних наборів даних. Це забезпечує постійне оновлення змісту навчальних проєктів і підвищує практичну цінність отриманих результатів. Крім того, робота з відкритими даними сприяє розвитку у студентів навичок самостійного пошуку інформації, аналізу структури даних та інтерпретації результатів дослідження.

Таким чином, використання відкритих вебданих у навчальному процесі забезпечує поєднання теоретичної підготовки з практичною реалізацією задач автоматизованого збору та інтелектуального аналізу інформації. Це дозволяє підвищити ефективність підготовки фахівців у сфері аналізу даних та забезпечує формування сучасних практичних компетентностей у галузі Data Science.

Висновки

У роботі досліджено особливості автоматизованого збору інформації з відкритих вебджерел та використання методів інтелектуального аналізу даних у навчальному процесі. Описано основні підходи до організації роботи з відкритими вебданими, використання технологій вебскрейпінгу, API та засобів попередньої обробки інформації.

Проаналізовано можливості використання pipeline-орієнтованого підходу у процесі підготовки фахівців з аналізу даних. Показано, що поєднання автоматизованого збору інформації, обробки даних та аналітичного аналізу дозволяє формувати у студентів комплексні практичні навички роботи з великими масивами відкритих даних.

Описано особливості використання методів інтелектуального аналізу даних у навчальному процесі, зокрема методів попередньої обробки даних, кореляційного аналізу, кластеризації та візуалізації результатів. Визначено,

що використання реальних відкритих вебджерел сприяє підвищенню практичної підготовки студентів та забезпечує наближення навчальних завдань до сучасних задач у сфері Data Science та інформаційної аналітики.

Отримані результати підтверджують доцільність використання практично орієнтованих підходів до викладання технологій автоматизованого збору та аналізу відкритих даних. Подальші дослідження можуть бути спрямовані на розширення використання сучасних методів машинного навчання та вдосконалення практичної підготовки студентів у сфері аналізу даних.

Список використаних джерел

1. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web. 2-ге вид. Sebastopol : O'Reilly Media, 2018. 366 с.
2. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. 2-ге вид. Waltham : Morgan Kaufmann, 2011. 703 с.
3. Пасічник В., Шаховська Н. Інтелектуальний аналіз даних. Львів : Видавництво Львівської політехніки, 2018. 432 с.
4. Python Language Reference. *Python.org. Documentation*. URL: <https://www.python.org/> (дата звернення: 27.03.2026).

ВІДОМОСТІ ПРО АВТОРІВ

ГАЛАВАЙ Олександр Миколайович

магістр комп'ютерних наук, асистент катедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

HALAVAI Oleksandr

Master of Computer Science, Assistant Professor of the Department of Theory and Technology of Programming,

Taras Shevchenko National University of Kyiv.

e-mail: halavai@knu.ua

ORCID ID 0009-0000-7393-1971

КРИВОЛАП Андрій Володимирович

кандидат фізико-математичних наук, доцент кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

KRYVOLAP Andrii Volodymyrovych

Ph.D. in Physical and Mathematical Sciences, Associate Professor of the Department of Theory and Technology of Programming,

Taras Shevchenko National University of Kyiv.

e-mail: andriikryvolap@knu.ua

ORCID ID 0009-0002-0493-9587

ЛІВІНСЬКА Ганна Володимирівна

кандидат фізико-математичних наук, доцент, доцент кафедри прикладної статистики,

Київський національний університет імені Тараса Шевченка

LIVINSKA Hanna Volodymyrivna

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Applied Statistics,

Taras Shevchenko National University of Kyiv.

e-mail: hanna.livinska@knu.ua

ORCID ID 0000-0001-9676-7932

МАКЕДОН Тетяна Валентинівна

студентка 1-го курсу магістратури факультету комп'ютерних наук та кібернетики,

Київський національний університет імені Тараса Шевченка

MAKEDON Tetiana Valentynivna

1st year master's student at the Faculty of Computer Science and Cybernetics,

Taras Shevchenko National University of Kyiv.

e-mail: taniamakedon03@gmail.com

МАКУШЕНКО Ігор Анатолійович

кандидат фізико-математичних наук, асистент кафедри прикладної статистики,

Київський національний університет імені Тараса Шевченка

MAKUSHENKO Igor Anatolijovych

Ph.D. in Physical and Mathematical Sciences, Assistant Professor of the Department of Applied Statistics,

Taras Shevchenko National University of Kyiv.

e-mail: 1978iamak@gmail.com

ORCID ID [0000-0002-4313-4936](https://orcid.org/0000-0002-4313-4936)

МОРЕНЕЦЬ Володимир Ігорович

кандидат фізико-математичних наук, асистент кафедри прикладної статистики,

Київський національний університет імені Тараса Шевченка

MORENETS Volodymyr Igorovych

Ph.D. in Physical and Mathematical Sciences, Assistant Professor of the Department of Applied Statistics,

Taras Shevchenko National University of Kyiv.

e-mail: vmorenets@knu.ua

ORCID ID [0009-0006-0899-4400](https://orcid.org/0009-0006-0899-4400)

НІКІТЧЕНКО Микола Степанович

доктор фізико-математичних наук, професор, професор кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

NIKITCHENKO Mykola Stepanovych

Doctor of Physical and Mathematical Sciences, Professor,

Professor of the Department of Theory and Technology of Programming,

Taras Shevchenko National University of Kyiv.

e-mail: nikitchenko_ms@knu.ua

ORCID ID [0000-0002-4078-1062](https://orcid.org/0000-0002-4078-1062)

ОМЕЛЬЧУК Людмила Леонідівна

кандидат фізико-математичних наук, доцент, доцент кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

OMELCHUK Liudmyla Leonidivna

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Theory and Technology of Programming, Taras Shevchenko National University of Kyiv.

e-mail: l.omelchuk@knu.ua

ORCID ID 0000-0002-2287-1304

ПАНЧЕНКО Тарас Володимирович

кандидат фізико-математичних наук, доцент, завідувач кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

PANCHENKO Taras Volodymyrovych

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Head of the Department of Theory and Technology of Programming, Taras Shevchenko National University of Kyiv.

e-mail: taras.panchenko@knu.ua

ORCID ID 0000-0003-0412-1945

ПОНОМАРЬОВ Вадим Дмитрович

кандидат фізико-математичних наук, асистент кафедри прикладної статистики, Київський національний університет імені Тараса Шевченка

PONOMAROV Vadym Dmytrovych

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Applied Statistics, Taras Shevchenko National University of Kyiv.

e-mail: vponomarov@gmail.com

ORCID ID 0000-0002-7813-3298

РОЗОРА Ірина Василівна

доктор фізико-математичних наук, професор, завідувач кафедри прикладної статистики,

Київський національний університет імені Тараса Шевченка

ROZORA Irina Vasylivna,

Doctor of Sciences in Physical and Mathematical Sciences, Professor, Head of Department of Applied Statistics,

Taras Shevchenko National University of Kyiv.
e-mail: irozora@knu.ua
ORCID ID 0000-0002-8733-7559

РУСІНА Наталія Геннадіївна

кандидат педагогічних наук, доцент, доцент кафедри теорії та технології програмування,
Київський національний університет імені Тараса Шевченка

RUSINA Nataliia Hennadiivna

Ph.D. in Pedagogical Sciences, Associate Professor of the Department of Theory and Technology of Programming,
Taras Shevchenko National University of Kyiv.
e-mail: rusina@knu.ua
ORCID ID 0000-0002-5595-9548

СВИСТУНОВ Антон Олександрович

асистент кафедри теорії та технології програмування,
факультет комп'ютерних наук та кібернетики,
Київський національний університет імені Тараса Шевченка

SVYSTUNOV Anton Oleksandrovych

Assistant Proffesor, Department of Theory and Technology of Programming,
Faculty of Computer Science and Cybernetics,
Taras Shevchenko National University of Kyiv
e-mail: antonsvystunov@knu.ua
ORCID ID 0009-0005-4536-5083

СИДОРОВ Микола Володимир-Станіславович

кандидат фізико-математичних наук, доцент, завідувач кафедри методології та методів соціологічних досліджень,

Київський національний університет імені Тараса Шевченка

SYDOROV Mykola Volodymyr-Stanislavovych

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Applied Statistics,
Taras Shevchenko National University of Kyiv.
e-mail: myksyd@knu.ua
ORCID ID 0000-0002-5333-8393

ТКАЧЕНКО Олексій Миколайович

кандидат технічних наук, доцент,
доцент кафедри теорії та технології програмування,
Київський національний університет імені Тараса Шевченка

TKACHENKO Oleksii Mykolaiovych

Ph.D. in Computer Sciences, Associate Professor,
Associate Professor of the Department of Theory and Technology of Programming,
Taras Shevchenko National University of Kyiv.

e-mail: otkachenko@knu.ua

ORCID ID 0000-0002-9514-516X

ШАРАПОВ Михайло Михайлович

кандидат фізико-математичних наук, доцент, доцент кафедри прикладної
статистики,

Київський національний університет імені Тараса Шевченка

SHARAPOV Mykhailo Mykhailovych

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate
Professor of the Department of Applied Statistics,
Taras Shevchenko National University of Kyiv.

e-mail: mmsharapov@knu.ua

ORCID ID 0000-0002-8047-995X

ШКІЛЬНЯК Степан Степанович

доктор фізико-математичних наук, професор,
професор кафедри теорії та технології програмування,
Київський національний університет імені Тараса Шевченка

SHKILNIAK Stepan Stepanovych

Doctor of Physical and Mathematical Sciences, Professor,
Professor of the Department of Theory and Technology of Programming,
Taras Shevchenko National University of Kyiv

e-mail: ss.sh@knu.ua

ORCID ID: 0000-0003-8624-5778

ШИШАЦЬКА Олена Володимирівна

кандидат фізико-математичних наук, доцент, доцент кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

SHYSHATSKA Olena Volodymyrivna

Ph.D. in Physical and Mathematical Sciences, Associate Professor, Associate Professor of the Department of Theory and Technology of Programming, Taras Shevchenko National University of Kyiv.

e-mail: shyshatska@knu.ua

ORCID ID 0000-0002-4078-1062304

ШИШАЦЬКИЙ Андрій Вікторович

асистент кафедри теорії та технології програмування,

Київський національний університет імені Тараса Шевченка

SHYSHATSKII Andrii Victorovich

Assistant Proffesor, Department of Theory and Technology of Programming, Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv

e-mail: ashyshatskyi@knu.ua

ORCID ID 0009-0000-5038-1110