МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ДОСЛІДЖЕННЯ ТА ОПТИМІЗАЦІЯ AUTOML-СИСТЕМ ДЛЯ ПОБУДОВИ ПАЙПЛАЙНІВ МАШИННОГО НАВЧАННЯ»

Виконав: студент 2 курсу

рівень: магістр

галузь знань 12 «Інформаційні технології»

спеціальність 122 «Комп’ютерні науки»

Дерев’ягін Владислав Сергійович

Керівник: к.пед.н., доцент

Лобода Юлія Геннадіївна

Рецензент: к.т.н., доцент

Гура Володимир Ігорович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **інформаційних технологій**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **122 Комп'ютерні науки**
Назва освітньої програми: **Комп'ютерні науки**

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент _____ Н.І. Логінова
«____ »_____ 20__ року

ЗАВДАННЯ
на кваліфікаційну (магістерську) роботу
здобувачу Дерев'ягіну Владиславу Сергійовичу

1. Тема роботи: «Дослідження та оптимізація AutoML-систем для побудови пайплайнів машинного навчання»

Керівник роботи: к.пед.н, доцент Лобода Юлія Геннадіївна
затверджені наказом НУ ОЮА від «*10*» *жовтня 2025* року № *3182-06*

2. Строк подання здобувачем роботи «*25*» *листопада 2025 рік*

3. Дата видачі завдання «*02*» *червня 2025 рік*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Вибір теми, вивчення літературних джерел та складання плану роботи	09.09.2025	
2.	Підготовка першого розділу роботи та подання його керівнику	01.10.2025	
3.	Підготовка другого розділу роботи та подання його керівнику	14.10.2025	
4.	Підготовка третього розділу роботи та подання його керівнику	07.11.2025	
5.	Підготовка вступу та висновків	10.11.2025	
6.	Доопрацювання роботи з урахуванням зауважень керівника	24.11.2025	
7.	Подання електронного варіанту роботи для перевірки на плагіат	25.11.2025	
8.	Допуск роботи до захисту завідувачем кафедри	01.12.2025	
9.	Захист роботи	03.12.2025	

Здобувач _____ Дерев'ягін В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Лобода Ю.Г.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Дослідження та оптимізація AutoML-систем для побудови пайплайнів машинного навчання» на здобуття другого (магістерського) рівня вищої освіти за спеціальністю 122 Комп'ютерні науки, освітньою програмою: Комп'ютерні науки, містить 18 рисунків, 84 літературних джерела за переліком посилань. Робота виконана на 83 сторінках загального тексту і 74 сторінках основного тексту.

Метою дослідження – дослідження та оптимізація AutoML-систем для підвищення ефективності побудови пайплайнів машинного навчання.

Об'єктом дослідження – процес автоматизованої побудови пайплайнів машинного навчання.

Предметом дослідження – методи оптимізації гіперпараметрів у AutoML-системах та механізми прискорення їх виконання.

У кваліфікаційній роботі розглядаються підходи до побудови та оптимізації AutoML-систем, спрямованих на підвищення ефективності створення моделей за умов великих обсягів даних. Проаналізовано існуючі концепції AutoML та методи оптимізації, включаючи базові підходи та байєсову оптимізацію. Представлено розроблення модульного AutoML-пайплайна та запропоновано адаптивний метод оптимізації гіперпараметрів, який дозволяє скоротити час побудови моделей без істотної втрати точності. Проведено експериментальне дослідження на репрезентативних наборах даних, що включало оцінювання якості моделей, аналіз швидкодії та застосування статистичних тестів для перевірки значущості результатів. Отримані висновки можуть бути використані для вдосконалення AutoML-рішень та застосування їх у практичних задачах машинного навчання.

AUTOML, АВТОМАТИЗОВАНЕ МАШИННЕ НАВЧАННЯ, ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ, МОДУЛЬНИЙ ПАЙПЛАЙН, АДАПТИВНИЙ МЕТОД, СТАТИСТИЧНИЙ АНАЛІЗ, ЕФЕКТИВНІСТЬ МОДЕЛЕЙ

ABSTRACT

The qualification work entitled “Research and optimization of AutoML systems for machine learning pipeline construction”, submitted for the second (master’s) level of higher education in specialty 122 Computer Science, educational program Computer Science, contains 18 figures and 84 bibliographic sources. The total volume of the work is 83 pages, including 74 pages of main text.

The purpose of the study is to research and optimize AutoML systems in order to increase the efficiency of building machine learning pipelines.

The object of the study is the process of automated construction of machine learning pipelines.

The subject of the study is the methods of hyperparameter optimization in AutoML systems and the mechanisms for accelerating their execution.

The qualification work examines approaches to the construction and optimization of AutoML systems aimed at improving the efficiency of creating models under conditions of large data volumes. Existing AutoML concepts and optimization methods, including basic approaches and Bayesian optimization, are analyzed. The development of a modular AutoML pipeline is presented, and an adaptive method of hyperparameter optimization is proposed, which makes it possible to reduce the time required for model construction without a significant loss of accuracy. An experimental study is performed on representative datasets, which includes model quality assessment, performance analysis, and the application of statistical tests to verify the significance of the results. The obtained findings can be used to improve AutoML solutions and to apply them in practical machine learning tasks.

AUTOML, AUTOMATED MACHINE LEARNING, HYPERPARAMETER OPTIMIZATION, MODULAR PIPELINE, ADAPTIVE METHOD, STATISTICAL ANALYSIS, MODEL EFFICIENCY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ML – Machine Learning

AutoML – Automated Machine Learning

CASH – Combined Algorithm Selection and Hyperparameter Optimization

RFE – Recursive Feature Elimination

SVM – Support Vector Machine

HPO – Hyperparameter Optimization

ROC-AUC– ROC Curve Area Under

RMSE – – Root Mean Squared Error

TPE – Tree-structured Parzen Estimator

SMBO – Sequential Model-Based Optimization

GP – Gaussian Process

BOHB – Bayesian Optimization Hyperband

TPOT – Tree-based Pipeline Optimization Tool

GLM – General Language Model

NAS – Neural Architecture Search

SMAC – Sequential Model-based Algorithm Configuration

DARTS – Darts is an open-source Python library

PCA – Principal Component Analysis

FLAML – Fast and Lightweight Auto ML

AWS – Amazon Web Services

AP – Average Precision

MAE – Mean Absolute Error

MSE - Mean Squared Error

UCB – Upper Confidence Bound

PI – Probability of Improvement

UML – Unified Modeling Language

ВСТУП

Актуальність теми дослідження зумовлена зростаючою складністю моделей машинного навчання та необхідністю їх швидкого та ефективного налаштування в умовах великих обсягів даних. Традиційні підходи до вибору моделей і гіперпараметрів потребують значних обчислювальних ресурсів, часу та експертного досвіду, що ускладнює широке застосування машинного навчання у практичних задачах. AutoML-системи дозволяють автоматизувати процес побудови пайплайнів машинного навчання, проте їх ефективність значною мірою залежить від методів оптимізації. У багатьох випадках сучасні підходи забезпечують високу якість моделей, але є надто повільними або недостатньо гнучкими, що створює потребу в удосконаленні механізмів оптимізації гіперпараметрів і скороченні часу пошуку оптимальних конфігурацій.

Метою дослідження є дослідження та оптимізація AutoML-систем для підвищення ефективності побудови пайплайнів машинного навчання.

Об’єктом дослідження є процес автоматизованої побудови пайплайнів машинного навчання.

Предметом дослідження – методи оптимізації гіперпараметрів у AutoML-системах та механізми прискорення їх виконання.

Для досягнення поставленої мети в кваліфікаційній роботі необхідно вирішити наступні **завдання**:

- дослідити предметну область;
- проаналізувати сучасні концепції та архітектури AutoML-систем;
- визначити критерії ефективності AutoML-рішень;
- розробити модульний AutoML-пайплайн;
- реалізувати оптимізаційні методи;
- провести експериментальне дослідження ефективності оптимізаторів на репрезентативних benchmark-датасетах;
- виконати статистичний аналіз отриманих результатів;

– сформулювати рекомендації щодо вибору оптимізаційних стратегій в AutoML-системах.

В кваліфікаційній роботі використовувалися загальнонаукові та спеціальні методи наукового дослідження таки, як системний аналіз, синтез, порівняння, статистичний аналіз, методи машинного навчання та методи оптимізації гіперпараметрів.

Наукова новизна дослідження полягає у розробленні та перевірці адаптивного методу оптимізації гіперпараметрів, який забезпечує скорочення часу побудови моделей без помітної втрати їх точності. Запропонований підхід уточнює підходи до підвищення ефективності автоматизованих систем машинного навчання.

Теоретична значущість проведеного дослідження полягає у поглибленні наукових уявлень про процеси оптимізації гіперпараметрів у AutoML-системах, а також у формуванні системної моделі побудови пайплайнів машинного навчання з урахуванням критеріїв якості, швидкодії та стабільності. Результати роботи уточнюють підходи до оцінювання AutoML-процесів та можуть бути використані для подальших досліджень у галузі автоматизованого машинного навчання.

Практичне значення одержаних результатів полягає у створенні робочого AutoML-пайплайна модульної архітектури та реалізації адаптивного оптимізатора, здатного скорочувати час підбору гіперпараметрів. Розроблений інструментарій може бути використаний у прикладних задачах Data Science, а також як основа для подальшого розширення функціональності та інтеграції у більш масштабні ML-платформи.

Результати кваліфікаційного дослідження **апробовані** на VI Міжнародної науково-практичної конференції «Кібербезпека в сучасному світі: актуальні виклики», (м. Одеса, 28 листопада 2025 р.) [19].

Структура кваліфікаційної роботи відповідає меті та завданням дослідження та складається зі вступу, основної частини з трьох розділів, висновків, списку використаних джерел та додатків. Робота викладена на 83 сторінках, містить 18 рисунків, 12 таблиць. Список використаних джерел включає 84 джерела.

1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗОВАНОГО МАШИННОГО НАВЧАННЯ

1.1 Концепція та принципи AutoML

Автоматизоване машинне навчання (Automated Machine Learning, AutoML) підхід, який дає змогу автоматизувати процес побудови моделей машинного навчання, зменшуючи кількість ручних налаштувань, що традиційно виконуються аналітиками [1]. Традиційний цикл розроблення моделей машинного навчання є трудомістким та вимагає глибокої експертизи на кожному етапі. Дослідження свідчать, що фахівці витрачають до 70–80 % часу на підготовку даних та підбір налаштувань моделі [2-4], що суттєво знижує ефективність аналітичного процесу. Основна ідея AutoML полягає у спрощенні процесу моделювання, підвищенні його швидкості та доступності для користувачів, які не є експертами у сфері Data Science.

Стандартний процес створення ML-моделі складається з кількох основних етапів:

– збір та аналіз даних – дослідження структури набору даних, виявлення пропусків, аномалій, особливостей розподілів [5];

– препроцесинг – очищення, нормалізація, кодування категоріальних ознак;

– інженерія ознак – створення додаткових та трансформація змінних, які можуть покращити якість моделі [6];

– відбір ознак – зменшення розмірності та видалення нерелевантних атрибутів;

– вибір алгоритму навчання – визначення моделі, яка найкраще відповідає типу задачі;

– налаштування гіперпараметрів – підбір параметрів, що впливають на процес навчання;

– навчання моделі – побудова залежності між ознаками та цільовою змінною;

– оцінювання результатів – перевірка точності на тестових даних [7].

Такий процес є ітеративним, у разі низької якості моделі аналітик повертається до попередніх етапів, змінює спосіб обробки даних, інженерію ознак або налаштування алгоритму. Саме повторюваність і трудомісткість цього процесу стали причиною появи AutoML [8].

AutoML автоматизує рутинні та технічно складні етапи побудови моделі. Замість того, щоб вручну обирати методи препроцесингу, алгоритми та їхні параметри, користувач задає лише вхідні дані та мету, а система автоматично аналізує структуру даних, підбирає методи обробки ознак, обирає кілька можливих моделей, оптимізує їх гіперпараметри та формує найкращий пайплайн.

На концептуальному рівні пайплайн AutoML подають як послідовність операцій, що виконуються одна за одною. У найпростішому випадку AutoML автоматизує лише оптимізацію моделі, тоді як у більш просунутих системах автоматизується весь процес від роботи з сирими даними до отримання готової моделі [9].

Формально задача машинного навчання може бути подана як оптимізація функції втрат [10]:

$$\theta^* = \arg\min_{\theta \in \Theta} L\big(f_\theta(D)\big), \qquad (1.1)$$

де f_θ – модель,

θ – вектор її параметрів,

Θ – простір параметрів,

L – функція втрат (cross-entropy, MSE, hinge loss для SVM тощо),

D – набір даних.

AutoML-пайплайн відрізняється від класичного ML-процесу тим, що трансформації даних, генерація ознак, підбір моделей та гіперпараметрів виконуються автоматично в межах єдиного оптимізаційного циклу.

Розвиток AutoML відбувався поступово, проходячи кілька чітко визначених етапів (рисунок 1.1):

– ручне моделювання (до 2010 р.) – аналітик самостійно виконував всі кроки;

– часткова автоматизація (2010–2015 рр.) – поява окремих інструментів оптимізації параметрів;

– AutoML 1.0 (2015–2020 рр.) – комплексні системи, які автоматизують декілька етапів моделювання;

– AutoML 2.0 (з 2020 р.) – інтеграція глибинного навчання, мета-навчання, структурного пошуку моделей.

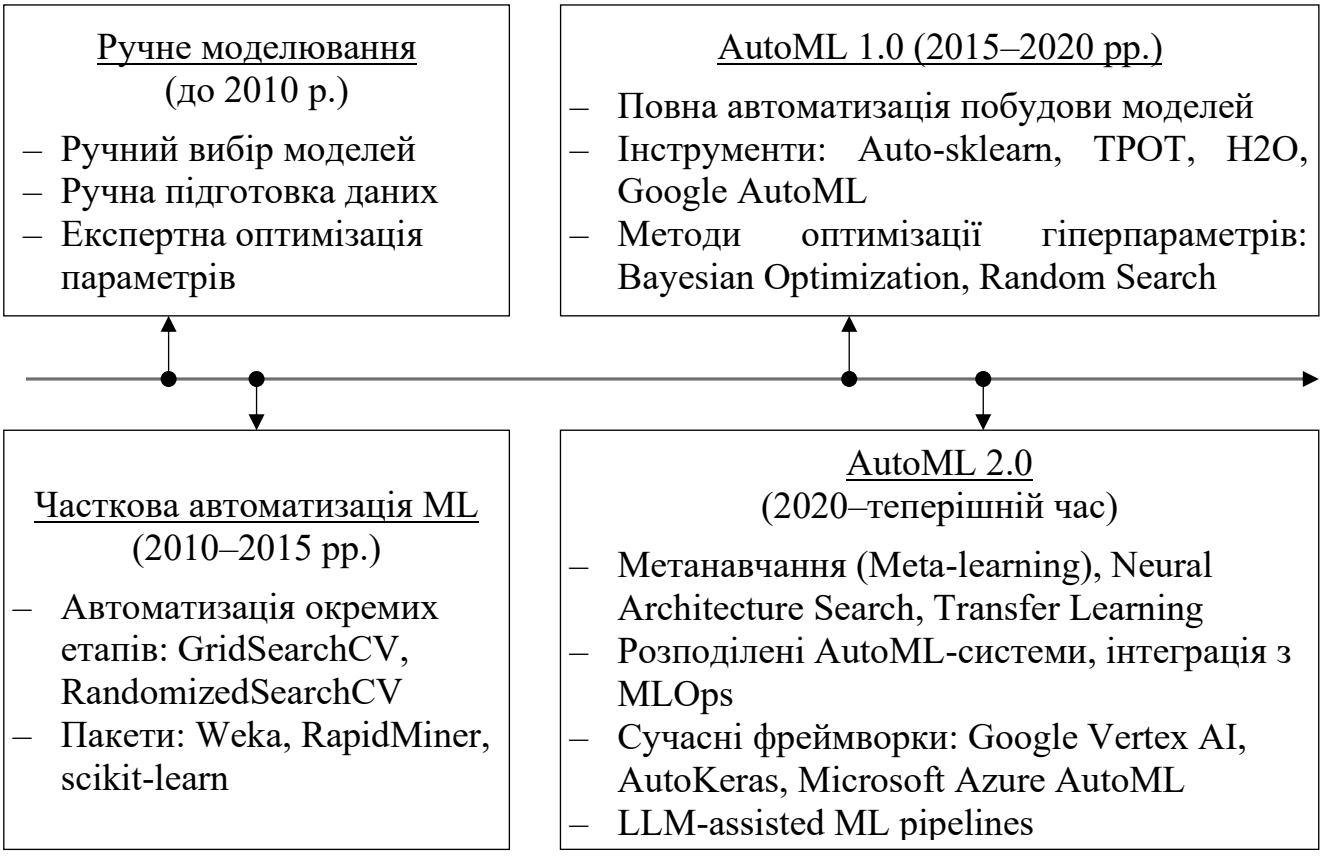

Рисунок 1.1 – Еволюція підходів до побудови AutoML

Сучасні системи AutoML здатні автоматично визначати оптимальні архітектури моделей та адаптувати стратегії обробки даних. Фундаментальним принципом AutoML є комплексна автоматизація життєвого циклу моделі від сирих даних до готового рішення. Система повинна вміти визначати типи змінних, обирати методи обробки, генерувати ознаки, обирати алгоритм та оптимізувати його гіперпараметри [11].

Ефективне використання обчислювальних ресурсів є ще одним важливим принципом AutoML. Повний перебір всіх можливих комбінацій параметрів є неприйнятним для задач реального масштабу, тому сучасні системи застосовують методи прискорення, зокрема байєсову оптимізацію, раннє зупинення та мета-навчання [12].

У сучасній літературі виділяють три рівні автоматизації ML-систем:

– базовий рівень – автоматизація окремих компонентів (генерація ознак, відбір ознак, оптимізація параметрів окремої моделі);

– проміжний рівень – автоматизація комбінованих задач, зокрема одночасного вибору алгоритму та оптимізації його гіперпараметрів (Combined Algorithm Selection, CASH) [13-15];

– повна автоматизація – системи End-to-end AutoML, які приймають сирі дані та повертають навчений пайплайн без проміжного втручання. У таких системах інтегруються всі компоненти, і додатково може використовуватися мета-навчання для пришвидшення оптимізації [16, 17].

Попри значний потенціал, AutoML має низку обмежень, які необхідно враховувати при практичному застосуванні:

– великі обчислювальні витрати при роботі з великими просторами параметрів;

– складність інтерпретації автоматично сформованих моделей;

– можливі помилки під час автоматичного оброблення нетипових даних;

– недостатнє використання експертних знань у специфічних предметних галузях.

Таким чином, AutoML є перспективним напрямом, що істотно спрощує та прискорює процес побудови моделей машинного навчання. Водночас він не може повністю замінити роль експерта, особливо у випадках, що потребують глибокого контекстного розуміння аналізу або врахування доменної специфіки.

1.2 Архітектура та основні компоненти AutoML-систем

AutoML-системи поєднують набір програмних модулів, які забезпечують виконання ключових етапів моделювання від аналізу даних до оптимізації параметрів моделей [18]. На відміну від класичного підходу, де аналітик самостійно визначає послідовність операцій та налаштувань, AutoML автоматично формує та оцінює різні комбінації пайплайніву межах єдиного оптимізаційного процесу.

Архітектура більшості сучасних AutoML-систем узгоджується зі структурою, наведеною на рисунку 1.2, і включає такі основні компоненти, які послідовно трансформують вхідні дані у валідовану модель.

1. Препроцесинг даних (Data Preprocessing).

На цьому етапі система автоматично застосовує методи первинної обробки даних: імпутація або видалення пропусків, виявлення та обробку аномалій (IQR, Isolation Forest), кодування категоріальних змінних (One-hot, Target encoding) та масштабування числових ознак (Standard, MinMax) [19-22].

Ключовою особливістю AutoML є автоматичний вибір методів (блок "Автовибір методів" на рисунку 1.2) залежно від властивостей даних: типу змінних, розподілів, наявності пропусків та викидів.

2. Інженерія та відбір ознак (Feature Engineering & Selection).

Цей компонент відповідає за формування оптимального простору ознак та включає дві підсистеми: підсистема генерації ознак (Polynomial features, Deep Feature Synthesis) [23, 24]; підсистема відбору ознак: фільтраційні методи відбору (Correlation, Mutual Information), wrapper-методи - ітеративний відбір через оцінку моделі (RFE, Forward selection) та embedded-підходи - відбір інтегрований у процес навчання (L1 regularization, Tree importance) [25-28].

Блок «Автовідбір ознак» забезпечує автоматичний добір найбільш інформативних ознак, що дає змогу зменшити розмірність та покращити якість моделі.

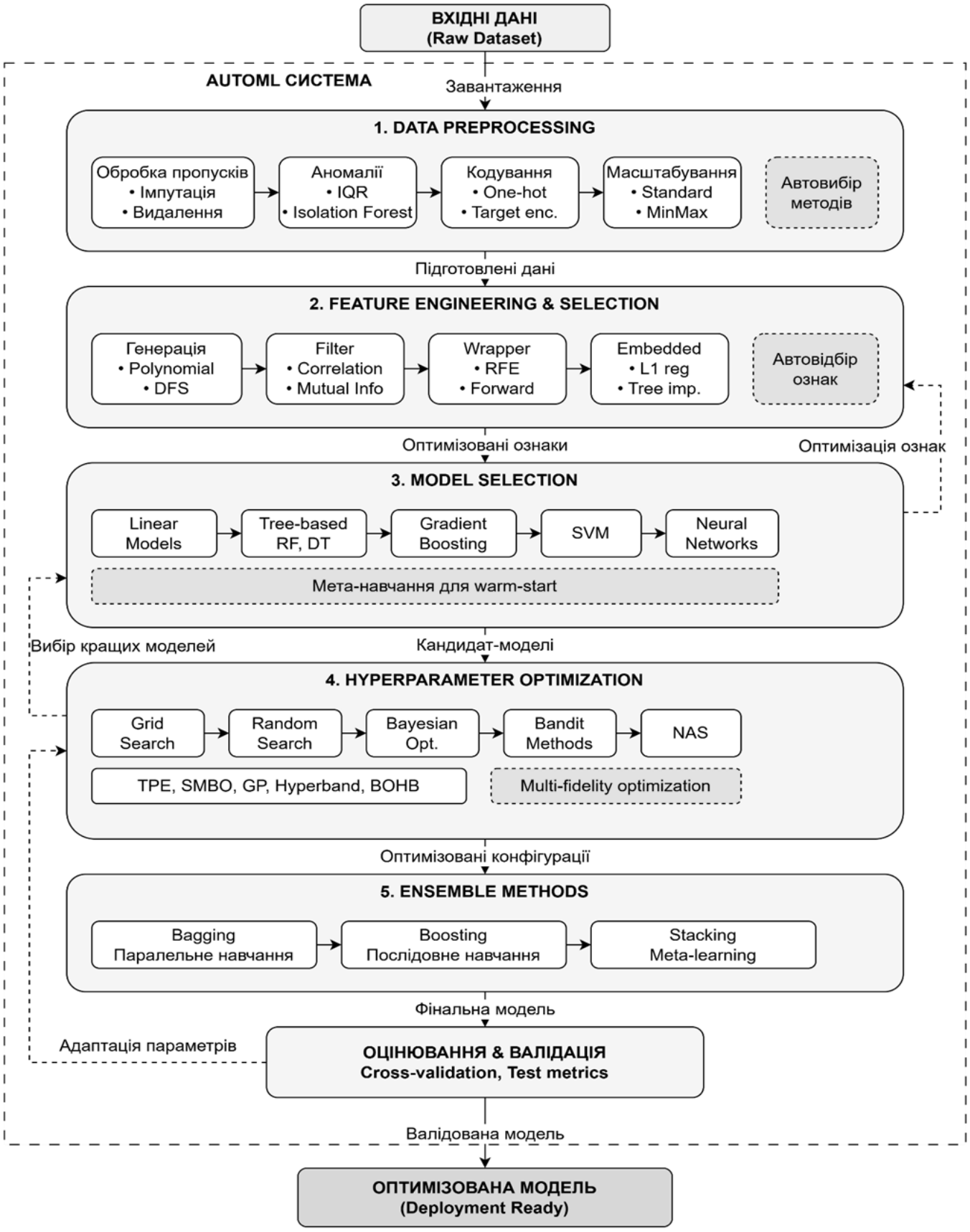

Рисунок 1.2 – Архітектура AutoML-системи

3. Вибір моделі (Model Selection).

Система автоматично тестує різні типи алгоритмів: лінійні моделі, деревоподібні методи (Random Forest, Decision Trees), градієнтний бустинг, SVM та нейронні мережі. Для прискорення пошуку застосовується мета-навчання (warm-start), що дозволяє ініціалізувати оптимізацію на основі знань з попередніх задач зі схожими характеристиками даних.

4. Оптимізація гіперпараметрів (Hyperparameter Optimization, HPO).

У цьому модулі реалізуються методи пошуку найкращих гіперпараметрів (Grid Search [29], Random Search [30], Bayesian Optimization [31], Bandit-based approaches [32], Neural Architecture Search) [33-35].

Сучасні системи підтримують багатокритеріальну оптимізацію (multi-fidelity optimization), що дозволяє балансувати між точністю моделі та обчислювальними витратами.

5. Ансамблювання моделей (Ensemble Methods).

Система автоматично комбінує кілька моделей для підвищення якості: bagging (паралельне навчання), boosting (послідовне навчання з корекцією помилок) та stacking (мета-навчання над передбаченнями базових моделей) [36-37].

6. Оцінювання та валідація.

Оцінювання моделей виконується за допомогою крос-валідації та відповідних метрик: класифікація: accuracy, precision, recall, F1-score, ROC-AUC, регресія: RMSE, MAE, R^2, MAPE. Результатом є валідована модель, яка готова до розгортання у виробничому середовищі (Deployment Ready.

На рисунку 1.2 наведено узагальнену архітектуру AutoML-процесу. Вхідні дані (Raw Dataset) надходять до модуля препроцесингу (1), де виконується автоматичний вибір методів обробки. Підготовлені дані передаються до модуля інженерії ознак (2), який генерує та відбирає оптимальні характеристики з можливістю зворотного зв'язку для оптимізації. Далі формується набір кандидат-моделей у модулі вибору моделі (3). Оптимізатор гіперпараметрів (4) застосовує просунуті методи пошуку (TPE, SMBO, GP,

Hyperband, BOHB). Отримані конфігурації можуть бути додатково покращені через ансамблювання (5). Фінальний етап оцінювання та валідація, результатом якого є оптимізована модель, готова до розгортання (Deployment Ready).

Автоматизація побудови моделей передбачає роботу з великою кількістю можливих комбінацій операцій. Для точного описання процесу автоматизації використовуються наступні поняття.

Оператор (pipeline operator) – окрема операція, яка виконується над даними (масштабування, кодування, відбір ознак, вибір моделі). Формально оператор o є функцією:

$$\text{o: D} \rightarrow \text{D'} \qquad (1.2)$$

де D – вхідне представлення даних, D' – трансформоване представлення.

Приклади операторів: StandardScaler, OneHotEncoder, RandomForestClassifier.

Пайплайн (pipeline) – впорядкована послідовність операторів, що утворює завершений процес обробки та моделювання даних.

$$P = (o_1, o_2, \dots, o_k), \qquad (1.3)$$

де o_i – оператор, який належить до множини допустимих операцій.

Виконання пайплайна є композицією операторів:

$$P(D) = o_{\text{k}}\left(\text{o}_{\text{k}}{}^{-1}\left(\dots \text{o}^2\left(\text{o}^1(\text{D})\right)\right)\right) \qquad (1.4)$$

Простір пошуку (search space) – множина всіх припустимих пайплайнів, які AutoML може згенерувати та оцінити:

$$S = \{P = (o_1, o_2, \dots, o_k) | o_{\text{i}} \in O_{\text{i}}, i = 1..k\} \qquad (1.5)$$

де O_{i} – множина доступних операторів для і-го етапу.

Ширина простору пошуку визначається як:

$$|S| = \prod_{\text{i}=1}^{\text{k}} |O_{\text{i}}| \qquad (1.6)$$

Формула демонструє комбінаторну природу задачі AutoML, навіть при помірній кількості операторів на кожному етапі загальна кількість можливих пайплайнів зростає експоненційно. Як видно з рисунка 1.2, на кожному з шести етапів (препроцесинг,

інженерія ознак, вибір моделі, HPO, ансамблювання, оцінювання) має численні альтернативи, що формує великий простір можливих конфігурацій.

У складніших системах, таких як Tree-based Pipeline Optimization Tool (TPOT), пайплайн представляється у вигляді графа операцій (operator graph)

$$G = (V, E), \qquad (1.7)$$

де вузли V – оператори, а ребра E – потоки даних між ними [32].

Автоматизований процес побудови моделі AutoML-системи може бути представлена як алгоритм з восьми послідовних кроків:

1. Аналіз структури даних і визначення типу задачі.
2. Автоматичне застосування відповідних методів препроцесингу.
3. Генерація та відбір ознак із можливістю зворотного зв’язку.
4. Тестування різних типів моделей із використанням мета-навчання.
5. Оптимізація гіперпараметрів через сучасні методи пошуку.
6. Формування ансамблів для підвищення робастності.
7. Оцінювання якості моделі через крос-валідацію.
8. Повернення фінального пайплайна та надання користувачу валідованої моделі, готової до розгортання.

Така архітектура забезпечує гнучкість і адаптивність AutoML-систем, дозволяючи автоматично будувати високоякісні моделі для широкого спектра задач машинного навчання з мінімальним втручанням користувача.

1.3 Огляд сучасних AutoML-платформ та підходів до оптимізації

Сучасні AutoML-системи реалізують різні підходи до автоматизації побудови моделей машинного навчання, проте всі вони мають спільну мету скорочення часу створення ефективних моделей та зменшення кількості ручних налаштувань [38]. Розглянемо найбільш відомі AutoML-платформ, що відрізняються архітектурою, принципами оптимізації та ступенем автоматизації.

1. Auto-sklearn.

Auto-sklearn є однією з найпоширеніших платформ AutoML, яка базується на бібліотеці scikit-learn та використовує підхід CASH-оптимізації (Combined Algorithm Selection and Hyperparameter Optimization) [39].

Ключовою особливості системи Auto-sklearn є інтеграція компонентів у єдиному оптимізаційному процесі:

– застосування байєсової оптимізації для пошуку гіперпараметрів [41];
– використання мета-навчання для warm-start процесу;
– автоматичний вибір алгоритмів препроцесингу;
– ensemble selection як фінальний етап формування моделі [40-42].

Auto-sklearn демонструє сильні результати на табличних даних завдяки добре структурованому простору пошуку та ефективному оптимізаційному ядру.

2. TPOT (Tree-Based Pipeline Optimization Tool).

TPOT представляє альтернативний підхід до побудови ML-пайплайнів [43]. Основна ідея розглядати пайплайн як генетичну структуру, яка може змінюватися через: мутацію (зміна оператора в пайплайні), кросинговер (комбінування двох пайплайнів), селекцію (відбір найуспішніших рішень).

Переваги TPOT: здатність знаходити складні нестандартні пайплайни; повна автоматизація preprocessing → features → model → HPO; представлення пайплайна у вигляді operator graph, а не лише лінійної послідовності.

Недолік – висока обчислювальна вартість через велику кількість ітерацій.

3. H2O AutoML.

H2O AutoML орієнтована на промислові задачі та масштабовані обчислення. Система підтримує розгортання на кластерах Hadoop та Spark, що дозволяє ефективно працювати з наборами даних розміром у терабайти [44].

Платформа пропонує:

– автоматичний перебір моделей (GLM, GBM, XGBoost, Deep Learning);
– використання Stacked Ensembles як фінального етапу;

– роботу з великими даними (H2O Distributed Framework);

– вбудовану перехресну перевірку і захист від переобучення.

Особливість H2O AutoML акцент на ансамблевих моделях, що підвищує стабільність результатів. Недоліками є складність налаштування розподіленого середовища та надлишкові обчислювальні витрати для невеликих наборів даних.

4. AutoGluon.

AutoGluon є сучасною бібліотекою від Amazon Web Services, орієнтованою на простоту використання [45].

Переваги:

– автоматичний пошук моделей та гіперпараметрів;

– використання глибоких ансамблів;

– сильна підтримка табличних, текстових, зображень і мультимодальних даних;

– адаптивний простір пошуку, який корегується на основі проміжних результатів.

AutoGluon вирізняється здатністю швидко підлаштовуватися під властивості даних, завдяки чому часто перевершує традиційні AutoML-підходи [46].

5. Google AutoML / Vertex AI AutoML.

Google AutoML – хмарна платформа, яка пропонує повну автоматизацію для задач класифікації, регресії та комп’ютерного зору з фокусом на автоматизацію глибинного навчання.

Основні особливості:

– використання нейронного архітектурного пошуку (NAS);

– оптимізація через масштабовані хмарні обчислення;

– інтерфейс «end-to-end»: від даних до розгортання;

– сильна інтеграція з BigQuery та Vertex AI Pipelines.

Google AutoML / Vertex AI AutoML приклад AutoML 2.0, де фокус зміщується на автоматизацію глибинного навчання.

6. Optuna та Ray Tune.

Optuna та Ray Tune спеціалізовані системи оптимізації гіперпараметрів, які часто інтегруються в AutoML-фреймворки. Optuna використовує байєсову та TPE-оптимізацію, pruning, multi-objective. Ray Tune використовує масштабування HPO, гібридні методи, підтримка Hyperband.

Узагальнюючи розглянуті системи в таблиці 1.1 предтавлено порівняльну характеристику платформ за ключовими критеріями.

Таблиця 1.1 – Порівняльний аналіз AutoML-платформ

Платформи	Основний підхід	Сильні сторони	Обмеження
Auto-sklearn	Байєсова оптимізація + мета-навчання	Стабільність на табличних даних	Менш ефективна на великих даних
TPOT	Еволюційні алгоритми	Пошук складних пайплайнів	Високі обчислювальні витрати
H2O AutoML	Ансамблювання	Робастність, масштабованість	Менше контролю структури пайплайна
AutoGluon	Мультирівневі ансамблі	Сильна адаптивність	Важче інтерпретувати результат
Google AutoML	NAS	Потужність, автоархітектури	Висока вартість, залежність від хмари

Аналіз таблиці демонструє, що не існує універсальної платформи, оптимальної для всіх сценаріїв. Вибір системи визначається: типом даних, розміром набору даних, доступними ресурсами, вимогами до інтерпретованості, часовими обмеженнями, бюджетом, а саме безкоштовні open-source vs комерційні хмарні рішення.

Огляд показує, що сучасні AutoML-платформи використовують різні стратегії оптимізації від байєсових методів і еволюційних алгоритмів до нейронного архітектурного пошуку. Різноманітність підходів демонструє складність задачі оптимізації ML-пайплайнів і пояснює, чому проблеми структурування пошукового простору і вибору методів оптимізації залишаються актуальними. Вибір конкретної платформи має базуватися на характеристиках задачі, обсягах даних, доступних ресурсах і рівні експертизи команди.

1.4 Методи оптимізації гіперпараметрів в системах AutoML

Оптимізація гіперпараметрів є одним із центральних компонентів AutoML-систем, оскільки саме від вибору параметрів моделі залежить її точність, стабільність та здатність узагальнювати дані. Ефективні методи пошуку конфігурацій дозволяють зменшити час моделювання, уникнути перенавчання та підвищити якість отриманих рішень [48]. У сучасних AutoML-платформах використовуються як базові підходи, так і складні адаптивні стратегії оптимізації.

На рисунку 1.3 представлено класифікацію методів оптимізації гіперпараметрів.

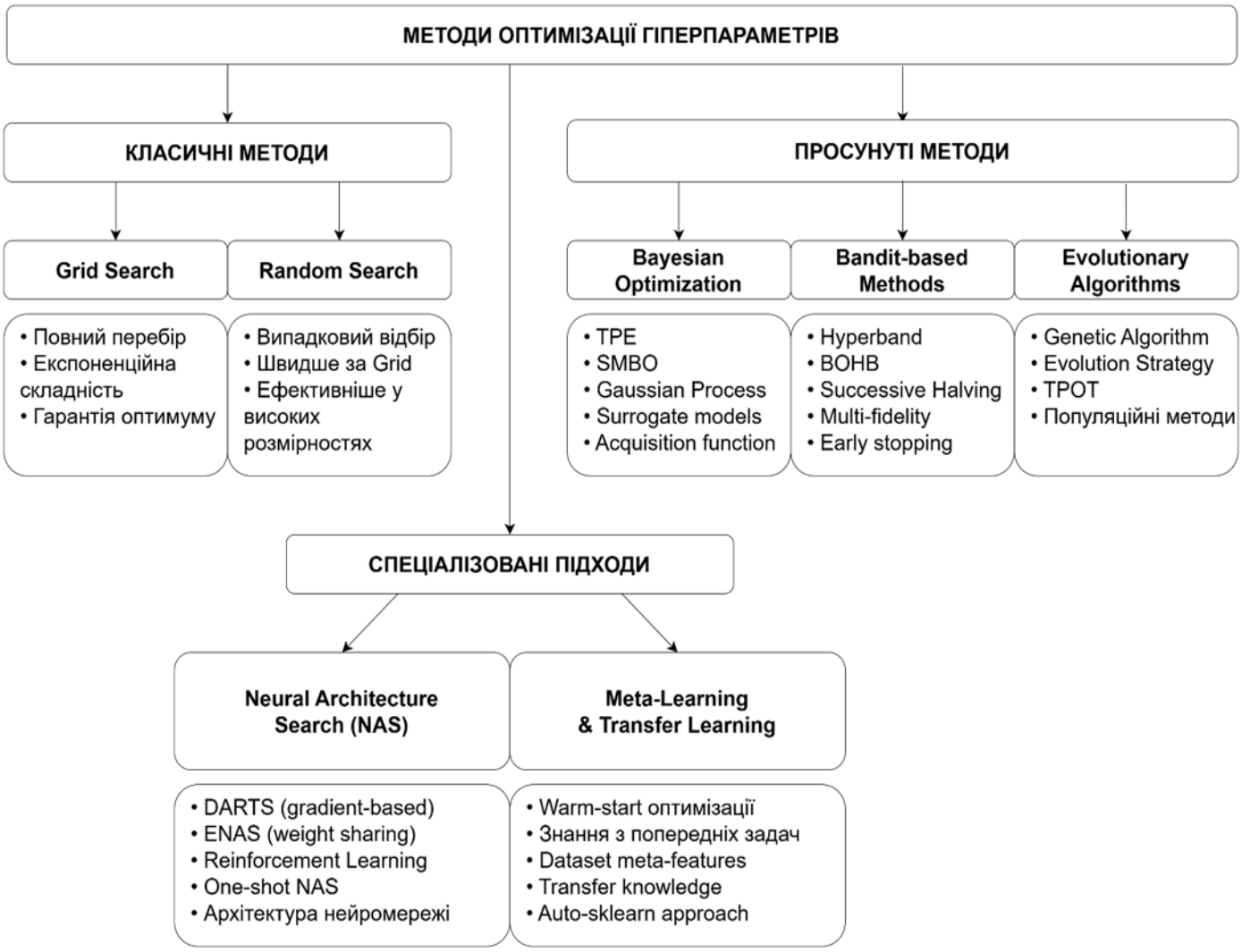

Рисунок 1.3 – Класифікація методів оптимізації гіперпараметрів в AutoML-системах

Класифікація охоплює три основні групи:

– класичні методи (Grid Search, Random Search) – базуються на систематичному або випадковому дослідженні простору без формування моделі функції втрат;

– просунуті методи (Bayesian Optimization, Bandit-методи, еволюційні алгоритми) – використовують інтелектуальні стратегії для направленого пошуку на основі попередніх результатів ;

– спеціалізовані підходи, орієнтовані на складні задачі AutoML, пошук архітектур нейронних мереж (NAS) та оптимізація на основі попереднього досвіду (Meta-Learning).

Представлена класифікація відображає еволюцію методів від повного перебору конфігурацій до адаптивних підходів, що здатні працювати в умовах обмежених ресурсів та великих просторів пошуку.

1. Класичні методи оптимізації.

Класичні методи базуються на систематичному або випадковому дослідженні простору гіперпараметрів без формування моделі функції втрат. Незважаючи на простоту, вони є базовою точкою порівняння для більш складних методів.

1.1. Grid Search реалізує повний перебір дискретної сітки значень для кожного параметра. Для кожної комбінації параметрів виконується навчання моделі та оцінювання її якості через крос-валідацію або окремий тестовий набір [49]. Наприклад, для простору з п'ятьма параметрами, кожен з яких має десять можливих значень, необхідно оцінити 100000 конфігурацій, що робить метод непрактичним для складних моделей. Переваги: гарантія знаходження глобального мінімуму в дискретному просторі; простота реалізації.

Недоліки: експонентне зростання кількості конфігурацій; непрактичність у високорозмірних просторах.

Метод доцільний у задачах з 2–3 параметрами та достатніми ресурсами.

1.2. Random Search генерує випадкові конфігурації, що дає змогу ефективніше досліджувати простір параметрів у порівнянні з Grid Search [49, 50]. Переваги: добре працює у високорозмірних просторах; швидше за Grid Search; дозволяє досліджувати безперервні області. Недолік метод не враховує результати попередніх запитів.

2. Просунуті методи оптимізації.

Просунуті методи стратегію вибору наступної конфігурації на основі попередніх результатів, що дозволяє значно скоротити кількість експериментів.

2.1. Байєсова оптимізація моделює залежність функції втрат від гіперпараметрів через побудову сурогатної моделі, наприклад, Gaussian Process або Random Forest [51].

Основні реалізації:

– Tree-structured Parzen Estimator (TPE) оцінює розподіли «хороших» та «поганих» конфігурацій [52];

– SMAC (Sequential Model-based Algorithm Configuration) застосовує Random Forest для обробки шумних даних та категоріальних параметрів [53].

Переваги: придатність для дорогих моделей; висока ефективність при малому бюджеті.

2.2. Bandit-based методи орієнтовані на оптимальний розподіл обчислювальних ресурсів між конфігураціями так, щоб швидко відсіяти неефективні варіанти [54].

Основні алгоритми:

– Successive Halving поступово скорочує кількість конфігурацій;

– Hyperband комбінує Successive Halving, з різним бюджетом ресурсів;

– BOHB (Bayesian Optimization + Hyperband) поєднує байєсову оптимізацію з ефективним розподілом ресурсів, досягаючи високої обчислювальної ефективності [55].

Переваги: одна з найвищих обчислювальних ефективностей серед сучасних методів; підтримка раннього зупинення моделей.

2.3 Еволюційні методи використовують механізми селекції, мутацій і схрещування для пошуку оптимальної конфігурації [56].

Представлені алгоритмами:

– Genetic Algorithm (GA) комбінує частини успішних конфігурацій для створення нових поколінь;

– Evolution Strategy (ES) адаптивно регулює параметри мутацій, забезпечуючи кращу збіжність;

– TPOT використовує генетичне програмування для автоматичної побудови ML-пайплайнів.

Переваги: ефективність у складних, нерегулярних просторах; оптимізація структури пайплайнів, а не лише параметрів. Недолік висока обчислювальна вартість.

3. Спеціалізовані методи.

Спеціалізовані методи розроблені для специфічних задач автоматизації, які виходять за межі традиційної оптимізації гіперпараметрів.

3.1. Neural Architecture Search (NAS) автоматизує проєктування архітектур нейронних мереж, визначаючи як параметри, так і структуру мережі [34].

Ключові напрямки:

– Reinforcement Learning використовує агента-контролера для побудови архітектур із винагородою за якість;

– Evolutionary NAS застосує еволюційні принципи для пошуку архітектури;

– Gradient-based підходи (DARTS) – оптимізація структури мережі через неперервне представлення пошуку [57].

NAS забезпечує автоматизацію дизайну нейромереж, однак залишається ресурсоємним навіть із сучасними оптимізаційними стратегіями.

3.2 Мета-навчання (Meta-Learning) використовує досвід попередніх задач для прискорення оптимізації нових [16].

Типові підходи включають:

– Warm-start – ініціалізацію пошуку конфігураціями, що були успішними для схожих задач;

– Transfer learning – перенесення інформації або моделей між задачами;

– Dataset meta-features – використання характеристик наборів даних для оцінки їхньої подібності.

Аналіз останніх досліджень дозволяє виділити кілька перспективних напрямків [58-61]:

1. Multi-fidelity optimization – використання наближених оцінок якості (менші датасети, менше епох) для швидкого відсіювання неперспективних конфігурацій з повною оцінкою лише для найкращих кандидатів.

2. Multi-objective optimization – одночасна оптимізація кількох конфліктуючих цілей: accuracy vs inference latency; accuracy vs model size; accuracy vs fairness. Результатом є Pareto-фронт компромісних рішень.

3. Federated HPO – розподілена оптимізація гіперпараметрів без централізації даних, критично для медичних та конфіденціальних застосувань.

4. AutoML for AutoML – автоматичний вибір методу HPO та його гіперпараметрів залежно від характеристик задачі.

Порівняльний аналіз методів оптимізації показує, що жоден метод не є універсальним. Вибір підходу залежить від ресурсів, складності задачі та розмірності простору пошуку. Еволюція методів оптимізації демонструє перехід від повного перебору до адаптивних стратегій, що поєднують інтелектуальний пошук із ефективним розподілом обчислювальних ресурсів. Саме такі методи лежать в основі сучасних AutoML-систем і визначають їх здатність будувати якісні та оптимізовані ML-пайплайни.

Висновки до розділу 1

У першому розділі розглянуто теоретичні основи автоматизованого машинного навчання. Встановлено, що AutoML забезпечує комплексну автоматизацію життєвого циклу моделювання, охоплюючи препроцесинг даних, інженерію ознак, вибір моделей, оптимізацію гіперпараметрів та формування фінального пайплайна.

Проаналізовано архітектуру сучасних AutoML-систем. Показано, що структура базується на взаємодії окремих модулів, які послідовно перетворюють сирі дані на готову модель. Формалізація процесу через оператори, пайплайни та простір пошуку дозволяє описати механізм оптимізації та оцінити його обчислювальну складність.

Виконано огляд провідних AutoML-систем (Auto-sklearn, TPOT, H2O AutoML, AutoGluon, Google AutoML / Vertex AI AutoML). Встановлено, що вони суттєво різняться підходами до пошуку моделей, стратегіями оптимізації, масштабованістю та рівнем автоматизації. Ці відмінності визначають їхню придатність до різних типів задач і впливають на швидкодію, стабільність та якість отриманих моделей.

Розглянуто методи оптимізації гіперпараметрів: від класичних алгоритмів до сучасних підходів, таких як байєсові методи, еволюційні стратегії, NAS та meta-learning. Показано, що сучасні AutoML-рішення базуються на поєднанні адаптивних стратегій пошуку та ефективного управління обчислювальними ресурсами. Отже, AutoML є комплексною технологією, здатною автоматизувати всі етапи побудови моделей машинного навчання.

2 АНАЛІЗ ТА МОДЕЛЮВАННЯ ПРОЦЕСІВ ПОБУДОВИ AUTOML-ПАЙПЛАЙНІВ

2.1 Аналіз сучасних методів побудови ML-пайплайнів

Процес машинного навчання традиційно розглядається як послідовність етапів: препроцесинг, інженерія ознак, вибір моделі, оптимізація параметрів. Однак у контексті автоматизації виникає принципово нова задача – організація цих етапів у структуровану систему, здатну до самостійної реконфігурації та оптимізації. Саме ця архітектурна складність визначає можливості та обмеження сучасних AutoML-систем. AutoML-пайплайн є не просто послідовністю операцій, а структурною одиницею, що визначає архітектуру всього процесу моделювання [62].

Для аналізу сучасних методів побудови пайплайнів систематизовано архітектурні типи та математичні моделі, що описують спосіб формування моделі.

Ключовими характеристиками архітектурної організації ML-пайплайнів є:

– тип зв'язків між компонентами (послідовні, паралельні, ієрархічні або гібридні структури);

– механізми передачі даних – способи трансформації та агрегації проміжних представлень;

– динамічність конфігурації – можливість адаптивної зміни структури залежно від властивостей даних;

– масштабованість виконання – підтримка розподілених обчислень та паралелізації (H2O, Ray Tune, Spark ML).

Саме архітектурна гнучкість відрізняє сучасні AutoML-системи від статичних конвеєрів обробки, створюючи передумови для широкого різноманіття підходів до побудови пайплайнів [62].

Систематизація існуючих підходів дозволяє виділити п'ять основних архітектурних класів, кожен з яких характеризується специфічними структурними властивостями та областями застосування.

1. Лінійні (Linear Pipelines) характеризується суворою послідовністю етапів обробки без можливості паралелізму чи розгалуження [63]. Кожна операція отримує результат попередньої та передає трансформовані дані наступній.

Математично лінійна архітектура представляється як композиція функцій:

$$\hat{y} = f_n\left(f_{n-1}\left(\dots f_2\left(f_1(X)\right)\right)\right) \qquad (2.1)$$

де X – вхідні дані,

f_n – послідовні трансформації,

(f_1 – препроцесинг, f_2 – нормалізація, f_3 – генерація ознак, …, f_n – модель ML),

$\hat{y}$ – фінальний прогноз.

Типовий приклад лінійного пайплайна в scikit-learn: StandardScaler → PCA → LogisticRegression.

Лінійні структури залишаються доцільними для класичних алгоритмів (SVM, логістична регресія, лінійні моделі) з однорідними типами ознак та чіткими вимогами до порядку трансформацій.

2. Розгалужені (Branching / Columnar Pipelines) архітектури – паралельна обробка різних типів ознак (числові, категоріальні, текстові, часові ряди) через створення паралельних гілок обробки [64].

Структурна організація розгалуженого пайплайна:

– аналіз та розділення – класифікація ознак за типами та розподіл у відповідні підмножини;

– паралельна трансформація – незалежна обробка кожної підмножини спеціалізованими операторами;

– агрегація представлень – об'єднання трансформованих ознак у єдиний простір;

– моделювання – навчання моделі на консолідованому представленні.

Формально розгалужений пайплайн можна представити як:

$$\hat{y} = M\left(g_{num}(X_{num}) \oplus g_{cat}(X_{cat}) \oplus g_{text}(X_{text})\right) \qquad (2.2)$$

де g_i – трансформації для відповідних типів даних,

$\oplus$ - операція конкатенації, M – модель.

Практична реалізація включає інструменти типу ColumnTransformer (scikit-learn) або Feature Union (Spark ML Pipeline) [40].

3. Ансамблеві та мета-рівневі архітектури (Ensemble / Meta-learning Pipelines) на відміну від попередніх підходів, де пайплайн завершується єдиною моделлю, ансамблеві архітектури інтегрують множину моделей у систему колективного прийняття рішень [39].

Структура ансамблевого пайплайна:

Рівень 1. Базові моделі

$$g_1(X), g_2(X), \dots, g_k(X)$$

Рівень 2. Метамодель

$$\hat{y} = M_{\text{meta}}\big(g_1(X), g_2(X), \dots, g_k(X)\big) \qquad (2.3)$$

де g_i – базові моделі різних типів,

M_{meta} – мета-модель або функція агрегації.

Дворівнева структура працює так: внутрішня оптимізація → навчає модель, зовнішня оптимізація → шукає найкращий пайплайн.

Сучасні AutoML-системи активно використовують ансамблеві архітектури. H2O AutoML автоматично будує stacked ensembles з найкращих моделей, AutoGluon застосовує multi-layer stacking для максимізації якості, FLAML балансує між індивідуальними моделями та ансамблями [65, 66].

4. Багатомодальні (Multimodal Pipelines) архітектури інтегрують дані різних типів (текст, зображення, табличні дані) в єдину структуру [1, 63]. Кожна модальність вимагає спеціалізованих методів обробки та моделювання.

5. Архітектури реального часу (Real-time Pipelines) орієнтовані на застосування з жорсткими вимогами до затримки: алгоритмічна торгівля, real-time bidding у рекламі, детекція кібератак, моніторинг промислових процесів [16, 17]. Використовуються AWS SageMaker, TensorFlow Extended.

Принципова відмінність AutoML від традиційного підходу полягає в тому, що система не просто виконує заданий пайплайн, а автоматично генерує його структуру, це перетворює задачу побудови моделі на задачу пошуку у просторі можливих архітектур.

Ефективність побудови пайплайна визначається ступенем автоматизації окремих компонентів, узгодженістю передачі даних між етапами та гнучкістю конфігурації. У контексті AutoML системи формування пайплайна відбувається автоматично на основі метаданих про задачу, типи ознак та історичний досвід попередніх оптимізацій. Так, Auto-sklearn генерує комбінації операцій препроцесингу й моделей за допомогою байєсової оптимізації, TPOT застосовує генетичне програмування для еволюції повних пайплайнів, а H2O AutoML комбінує декілька моделей у stacked ensembles із подальшим ранжуванням за метриками якості [60].

2.2 Критерії ефективності AutoML-рішень

Оцінювання ефективності AutoML-систем багатовимірна задача, що виходить за межі традиційного аналізу якості моделей машинного навчання. Якщо у класичному підході центральним критерієм є точність прогнозів на тестових даних, то AutoML-рішення вимагають комплексного аналізу, що охоплює як характеристики фінальної моделі, так і властивості процесу її отримання. Фундаментальна особливість AutoML то, що система має бути ефективною не лише у термінах результату, але й у термінах витрачених ресурсів, зручності використання та придатності до промислового застосування.

На рисунку 2.1 представлено багатовимірну систему критеріїв ефективності AutoML-рішень, що об'єднує сім основних категорій: якість моделі, обчислювальна ефективність, масштабованість, економічна ефективність, автоматизація та простота, інтерпретованість і прозорість, робастність і генералізація, відтворюваність. Кожна категорія включає специфічні метрики та показники, що дозволяють всебічно оцінити AutoML-систему.

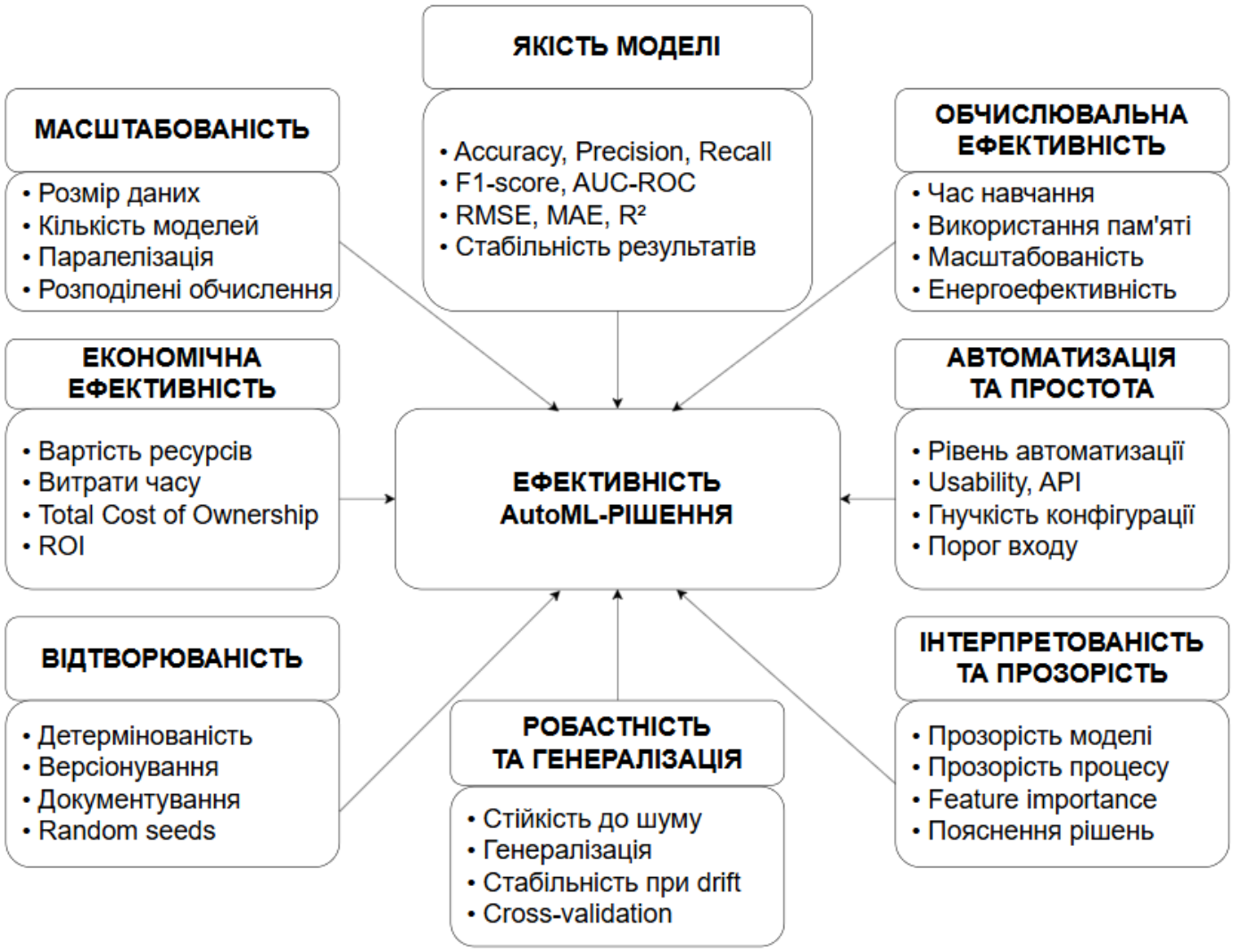

Рисунок 2.1 – Багатовимірна система критеріїв ефективності AutoML

Центральним критерієм оцінювання систем машинного навчання є якість фінальної моделі, однак у контексті AutoML визначення якості набуває специфічних особливостей.

Для задач класифікації традиційно використовуються метрики Accuracy, Precision, Recall та їх гармонічне середнє F1-score. Проте вибір метрики має відповідати характеристикам даних. У випадку незбалансованих класів для реальних застосувань (медична діагностика, виявлення шахрайства, детекція аномалій) показник Accuracy може створювати оманливе враження про якість моделі [67].

За таких умов більш репрезентативними є метрики, що враховують дисбаланс:

– Area Under ROC Curve (AUC-ROC) – оцінює здатність моделі ранжувати приклади незалежно від порогу класифікації;

– Average Precision (AP) – середнє значення precision для різних порогів recall;

– F1-score – гармонічне середнє precision та recall.

Для задач регресії основними метриками є:

– Mean Absolute Error (MAE) – середня абсолютна похибка;

– Mean Squared Error (MSE) – середня квадратична похибка;

– Root Mean Squared Error (RMSE) – корінь з MSE, вимірюється в тих самих одиницях, що й цільова змінна;

– Коефіцієнт детермінації R^2 – частка поясненої дисперсії, нормалізована метрика якості.

Принципово важливою особливістю AutoML-систем є автоматичний вибір або рекомендація відповідних метрик на основі аналізу характеристик задачі та даних: типу цільової змінної, розподілу класів, наявності пропусків, специфічних вимог предметної області.

Критерій обчислювальної ефективності включає декілька аспектів:

– час навчання (training time) – охоплює весь період від початку роботи AutoML-системи до отримання фінальної навченої моделі. Для багатьох застосувань існують жорсткі часові обмеження, що робить швидкість критичним фактором;

– використання пам'яті – визначає здатність системи працювати з наборами даних різного розміру без виходу за межі доступних ресурсів [68];

– масштабованість – визначає здатність системи підтримувати ефективність працювати при зростанні розміру даних або складності задачі;

– енергоефективність – AutoML-системи, що потребують тривалого навчання на потужному обладнанні (GPU, TPU), мають значний екологічний вплив, що має враховуватися при виборі рішення;

– автоматизація та зручність використання – визначають доступність AutoML-технологій для користувачів з різним рівнем експертизи. Ступінь автоматизації

характеризує, скільки рішень система приймає самостійно без втручання користувача. Простота використання включає інтуїтивність API, якість документації, наявність прикладів та навчальних матеріалів ;

– інтерпретованість моделі – визначає, наскільки легко зрозуміти логіку прийняття рішень фінальною моделлю;

– відтворюваність результатів – забезпечує можливість отримання ідентичних результатів при повторному запуску з тими самими вхідними даними та налаштуваннями[69].

Економічна ефективність охоплює як прямі, так і непрямі витрати, пов'язані з використанням AutoML-рішення. Так вартість обчислювальних ресурсів включає витрати на хмарні обчислення, використання GPU або спеціалізованого обладнання.

Ефективність AutoML-рішень визначається комплексом взаємопов'язаних критеріїв, що включають: якість моделі, обчислювальна ефективність, масштабованість, автоматизація та зручність, інтерпретованість і прозорість, робастність і генералізація, відтворюваність, економічна ефективність. Жоден окремий критерій не може повністю характеризувати якість AutoML-системи необхідний збалансований підхід, що враховує специфіку конкретного застосування та пріоритети користувача.

2.3 Моделювання процесу автоматизованого відбору моделей

Процес автоматизованого відбору моделей у AutoML-системах є послідовністю взаємопов'язаних етапів, що включають аналіз даних, ініціалізацію пошуку, ітеративну оптимізацію конфігурацій та побудову фінального рішення. На рисунку 2.2 представлено процес автоматизованого відбору моделей, що відображає повний життєвий цикл від отримання вхідних даних до генерації оптимізованої моделі.

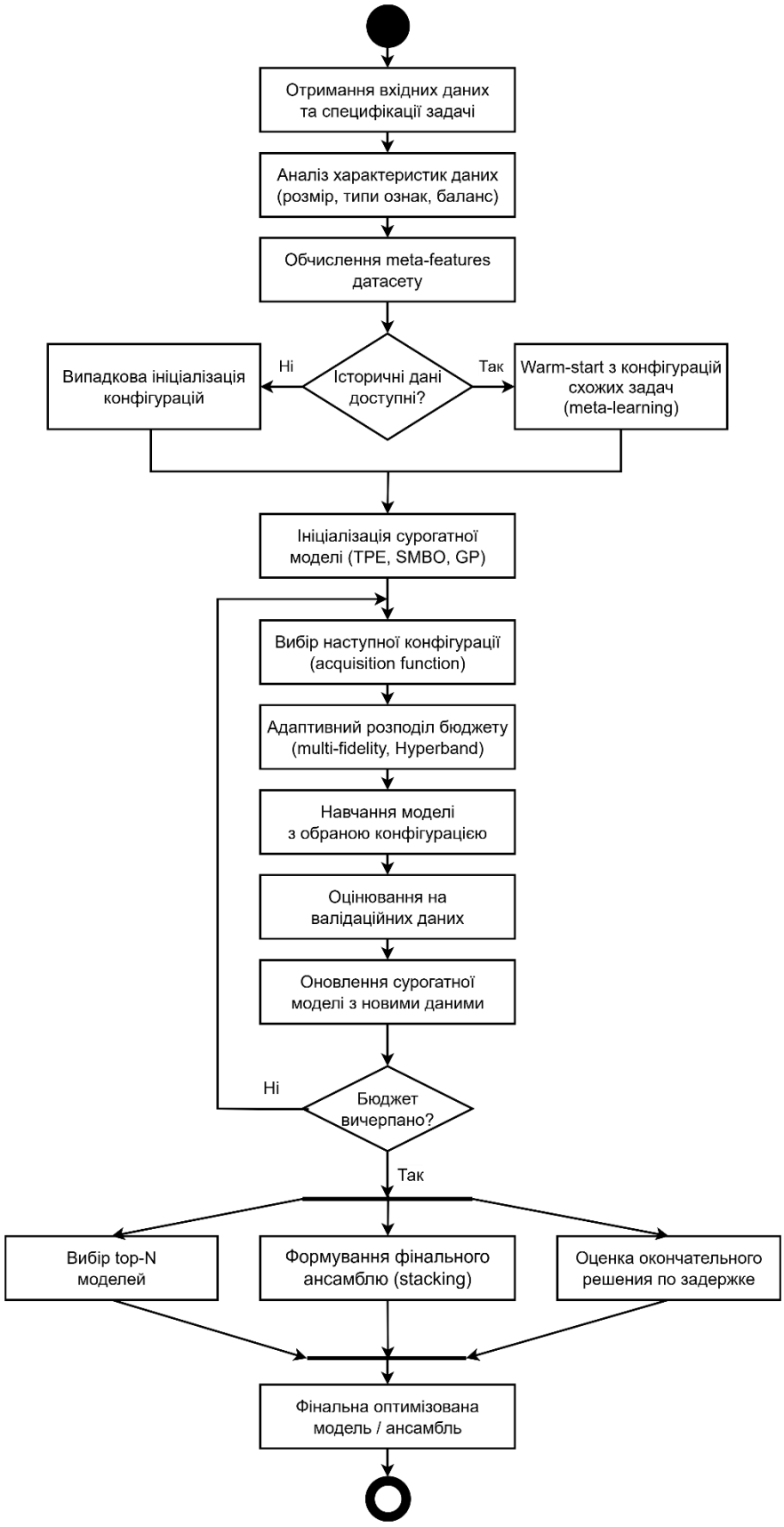

Рисунок 2.2 – Моделювання процесу автоматизованого відбору моделей

Процес автоматизованого відбору моделей у AutoML-системах організований у чотири основні фази: ініціалізація та аналіз даних, адаптивна ініціалізація через мета-навчання (warm-start або random start), ітеративна оптимізація конфігурацій та фінальна побудова ансамблю моделі. Кожна фаза включає специфічні операції та точки прийняття рішень, що визначають подальший потік виконання.

1. Фаза ініціалізації та аналізу даних.

Процес починається з отримання вхідних даних та специфікації задачі, що включає датасет, тип задачі (класифікація, регресія) та обмеження (часові, ресурсні). Система автоматично аналізує характеристики даних для визначення оптимальної стратегії пошуку.

Аналіз характеристик даних включає обчислення базових статистик: розмір вибірки (n_samples), кількість ознак (n_features), співвідношення n/p, типи ознак (числові, категоріальні), баланс класів, наявність пропущених значень.

Обчислення мета-ознак (meta-features) датасету є етапом для можливості використання мета-навчання. Мета-features класифікуються на декілька категорій: статистичні (mean, std, skewness, kurtosis), інформаційно-теоретичні (ентропія розподілу класів, взаємна інформація між ознаками та цільовою змінною), структурні (співвідношення розмірностей n/p, щільність даних) та модельні (якість baseline моделей). Цей етап завершує першу фазу та подає інформацію на блок прийняття рішення про тип ініціалізації.

2. Фаза адаптивної ініціалізації представлена як розгалуження (decision node), який визначає стратегію ініціалізації пошуку на основі доступності історичних даних. Якщо система має доступ до історичної бази попередніх задач з їх конфігураціями та результатами, виконується warm-start ініціалізація. Процес включає: обчислення схожості нового датасету з історичними задачами на основі відстані в просторі мета-features (L2-норма або Mahalanobis distance), вибір k найбільш схожих задач (k=25-50),

екстракцію конфігурацій найуспішніших конфігурацій моделей. Warm-start прискорює конвергенцію на 50–70%, оскільки одразу використовуються високоймовірні рішення.

Якщо історичні дані відсутні або датасет суттєво відрізняється від всіх історичних задач, система виконує випадкову ініціалізацію. Генерується набір початкових конфігурацій через відбір з визначених розподілів для кожного гіперпараметра. Кількість початкових конфігурацій становить 10-20 для забезпечення достатнього покриття простору.

Обидва шляхи ініціалізації зливаються в єдиний потік перед входом у фазу оптимізації.

3. Фаза ітеративної оптимізації конфігурацій є центральною у процесі відбору моделей і представлена на діаграмі як цикл (loop), що виконується до вичерпання обчислювального бюджету.

Перед початком циклу оптимізації система ініціалізує сурогатну модель функції якості. Вибір типу сурогатної моделі залежить від характеристик задачі:

– Tree-structured Parzen Estimator (TPE) для загального використання;

– Sequential Model-Based Optimization (SMBO) з Random Forest для задач з умовними залежностями між параметрами;

– Gaussian Process (GP) для низьковимірних просторів з потребою в каліброваній оцінці невизначеності.

Сурогатна модель будує апроксимацію функції якості

$$y = f(\lambda) \qquad (2.4)$$

де λ – вектор гіперпараметрів,

y – метрика якості на валідаційних даних.

Кожна ітерація містить вибір наступної конфігурації через максимізацію acquisition function $\alpha(\lambda)$, що балансує між дослідженням областей з високою невизначеністю та використанням знань перспективної області.

Популярні $\alpha(\lambda)$ включають:

– Expected Improvement (EI) – максимізоване очікуване покращення відносно кращої знайденої конфігурації;

– Upper Confidence Bound (UCB) – комбінації передбаченої якість з невизначеністю;

– Probability of Improvement (PI) – максимізація ймовірність покращення.

Навчання моделі з обраною конфігурацією виконується на тренувальних даних. Для прискорення оцінювання застосовуються підходи: навчання на меншому підмножині даних, менша кількість епох для ітеративних алгоритмів, нижча розмірність моделі. Low-fidelity оцінки використовуються для швидкої фільтрації явно неперспективних конфігурацій перед повним навчанням кращих кандидатів.

Оцінювання якості моделі здійснюється на валідаційних даних або крос-валідація (для малих датасетів).

Оновлення сурогатної моделі включає додавання нової пари λ_{new}, y_{new} до набору спостережень та покращує апроксимацію.

Умови завершення визначає продовження оптимізації чи ні. Оптимізація припиняється, якщо вичерпано часового бюджету (time-based), досягнена максимальна кількості ітерацій, немає покращення протягом N послідовних циклів та досягнено цільової метрики.

Адаптивний розподіл ресурсів AutoML підтримує за допомогою Successive Halving, Hyperband та BOHB (Hyperband + Bayesian Optimization). Перераховані методи підходи різко зменшують витрати, відсіюючи слабкі конфігурації на ранніх етапах.

4. Фаза побудови фінального ансамблю

Після завершення оптимізації процес переходить у блок із паралельними операціями (fork node). Система виконує одночасно декілька операцій над набором найкращих знайдених моделей:

– вибір топ-N моделей з усіх оцінених конфігурацій (N=10-50);

– побудову ансамблю включає voting, averaging.

– використання Stacking meta-model;

– оцінку на holdout-наборі.

Join node на діаграмі представляє синхронізацію всіх паралельних потоків перед переходом до фінального етапу. Система чекає завершення всіх паралельних операцій та агрегує їх результати.

Остання активність генерує фінальну оптимізовану модель або ансамбль, готовий до розгортання. Фінальне рішення може бути single best model (проста модель з найвищою якістю), weighted ensemble (зважена комбінація топ-моделей), stacked ensemble (багаторівнева архітектура з мета-моделлю). Вибір типу фінального рішення може базуватися на вимогах до латентності, інтерпретованості або максимальної якості. Процес завершується поверненням фінальної моделі користувачу разом з метаданими про процес оптимізації, кількістю оцінених конфігурацій, найкращій досягнутої якості, склад ансамблю (якщо використовується), витрачений час та ресурси.

2.4 Дослідження benchmark-датасетів

Систематична й об’єктивна оцінка AutoML-рішень неможлива без застосування стандартних benchmark-датасетів, які відображають різноманітність реальних задач машинного навчання та дозволяють порівнювати системи у відтворюваних умовах [70]. Якість експериментального аналізу залежить від того, наскільки репрезентативним є набір датасетів, які характеристики він покриває, чи містить реалістичні виклики (шум, пропуски, дисбаланс, а також від коректності оцінювання.

Для забезпечення валідності експериментального порівняння AutoML-систем benchmark-датасет має відповідати кільком критеріям.

Критерії відбору benchmark-датасетів:

– різноманітність характеристик, датасети мають охоплювати широкий спектр розмірів (від сотень до мільйонів зразків), кількості ознак (від десятків до десятків тисяч), типів даних (числові, категоріальні, змішані) та складності задач;

– реалістичність, датасети мають відображати проблеми, що зустрічаються у реальних застосуваннях, включаючи наявність пропущених значень, дисбаланс класів, шум у даних та корельовані ознаки;

– доступність, датасети мають бути публічно доступними для забезпечення відтворюваності результатів, так репозиторії UCI (Machine Learning Repository) [71], OpenML [72] та Kaggle надають стандартизований доступ до тисяч датасетів [73];

– відсутність витоку даних, критично важливо, щоб тестові дані не повинні використовуватися під час навчання, оптимізації або мета-навчання AutoML-систем;

– баланс складності, датасет має включати як тривіальні задачі (для перевірки базової функціональності), так і складні (для оцінки граничних можливостей системи).

Для оцінювання AutoML-систем використовується кілька відомих benchmark-ініціатив:

– OpenML-CC18 (Curated Classification Suite 2018) містить 72 датасети для задачі класифікації з відібраними критеріями якості, різноманітності та офіційними метаданими [72];

– AutoML Benchmark від AutoGluon включає 104 датасети різної складності, що охоплюють табличні дані, зображення та текст, орієнтованих на реалістичні обмеження ресурсів [74];

– PMLB (Penn Machine Learning Benchmarks) надає понад 290 датасетів для класифікації та регресії з єдиним інтерфейсом доступу [75];

– AutoML Challenge Series (2015-2018) використовував приховані датасети для змагань, забезпечуючи оцінювання без можливості перенавчання на тестових даних [76].

На якість та стабільність AutoML-систем безпосередньо впливають характеристики датасету, зокрема розмір вибірки, кількість ознак, баланс класів, типи ознак, наявність пропусків та рівень шуму. Для малих датасетів ($n < 1000$) кращих результатів зазвичай досягають завдяки регуляризації та використанню розширених схем крос-валідації. Для великих датасетів ($n > 100000$) визначальними стають масштабованість алгоритмів, застосування subsampling-підходів та multi-fidelity

стратегій, що дають змогу зменшити обчислювальні витрати без суттєвої втрати точності.

Мета-ознаки відображають статистичні, інформаційні, модельні властивості датасету та дозволяють прогнозувати ефективність алгоритмів через meta-learning [77]. Мета-ознаки поділяють на:

– прості мета-ознаки – n, p, співвідношення *n/p*, кількість класів, пропускі;
– статистичні – середні значення, розкиди, кореляційні характеристики;
– інформаційно-теоретичні – взаємна інформація, ентропія;
– модельні – якість простих моделей, глибина дерева, кількість листків.

Meta-learning-підходи використовують дані мета-ознаки для прогнозування, тобто які конфігурації моделей ймовірно будуть успішним.

Коректність висновків залежить від дотримання строгої послідовності дій:

– суворе розбиття train–validation–test без доступу AutoML до test;
– використання cross-validation для малих датасетів;
– регламентовані часові обмеження (1–24 години) ;
– стандартизовані апаратні ресурси CPU/GPU;
– використання множинних метрик (AUC, F1, log-loss для класифікації; RMSE, MAE для регресії).

Для багатомодельних і багатозадачних порівнянь використовуються статистично обґрунтовані методи: Friedman test + Nemenyi post-hoc, це стандарт для множинних порівнянь AutoML-систем, аналіз по категоріях датасетів , Win–Tie–Loss матриці [78].

Попри свою корисність benchmark-оцінювання має низку суттєвих обмежень: не завжди відображають реальні бізнес-сценарії, можуть бути «чистішими», ніж реальні дані, та іноді призводять до завищеної оцінки якості AutoML-рішень. Тому важливо комбінувати датасети з різних джерел, оновлювати benchmark-набір і документувати повний протокол експериментів. У дослідженні використано два benchmark-датасети: California Housing та Bank Marketing, які дозволяють оцінити ефективність AutoML-підходів для задач регресії та класифікації.

Висновки до розділу 2

У розділі 2 проведено аналіз теоретичних складових AutoML-систем, що формують функціональну та алгоритмічну основу. Розглянуто архітектурні принципи побудови AutoML-пайплайнів, математичні формалізації процесу автоматизованого пошуку моделей, методи оптимізації гіперпараметрів та ансамблювання, а також підходи до моделювання AutoML-процесів засобами UML.

Виділено чотири основні фази процесу автоматизованого відбору моделей: ініціалізація та аналіз даних, адаптивна ініціалізація, ітеративна оптимізація конфігурацій, побудова фінального ансамблю. Показано, що така структуризація забезпечує логічну послідовність автоматизованих етапів та дозволяє формалізувати процес вибору моделі як оптимізаційну задачу.

Сформульовано базові моделі представлення даних, простору гіперпараметрів та функції якості, що забезпечують формальне визначення задачі AutoML як задачі багаторівневої оптимізації. Наведені математичні формалізації створюють основу для подальшого аналізу ефективності AutoML-рішень та коректного порівняння методів оптимізації.

Окреслено критерії оцінювання AutoML-систем, що включають якість моделей, часові та обчислювальні витрати, стабільність результатів, інтерпретованість та узагальнюваність. Показано, що оцінювання AutoML не може зводитися лише до порівняння окремих метрик; необхідно враховувати багатовимірну систему показників, яка відображає повну “вартість” автоматизованого пошуку, включаючи швидкодію, ресурсомісткість та стабільність оптимізаційного процесу.

3 РОЗРОБЛЕННЯ ТА ОПТИМІЗАЦІЯ AUTOML-ПАЙПЛАЙНА

3.1 Вибір інструментарію для реалізації AutoML-системи

Реалізація AutoML-системи вимагала вибору інструментарію, що забезпечує повний цикл підтримки машинного навчання: підготовку даних, побудову моделей, оптимізацію гіперпараметрів, оцінювання та візуалізацію результатів. Виходячи з вимог до відтворюваності експериментів, масштабованості та сумісності з сучасними AutoML-підходами, основним середовищем реалізації обрано екосистему Python.

Python є провідною мовою у сфері машинного навчання завдяки відкритій екосистемі та великій кількості бібліотек. Ключові AutoML-системи – Auto-sklearn, TPOT, H2O AutoML та AutoGluon, побудовані саме на Python, що забезпечує їх сумісність і просту інтеграцію в експериментальне середовище [40,43-45].

1. Бібліотеки для оброблення та підготовки даних.

Для структурування очищення та трансформації даних використано інструменти:

– NumPy забезпечує ефективні обчислення над багатовимірними масивами та матрицями, основа числових обчислень;

– Pandas надає високорівневі структури даних (DataFrame), засоби завантаження, фільтрації, агрегації та попередньої обробки.

– Scikit-learn бібліотека, яка реалізовує алгоритми класифікації, регресії, кластеризації, зниження розмірності, засоби препроцесингу та оцінювання моделей [10, 19].

Для забезпечення повторюваної підготовки даних у системі використано модуль datasets.py (Додаток А, рис. А.1), який реалізує автоматизоване завантаження та базові операції препроцесингу.

2. AutoML-фреймворки та інструменти оптимізації гіперпараметрів.

Оптимізація гіперпараметрів є центральним компонентом AutoML-систем.

Основні інструменти:

– Optuna – реалізує TPE-семплер, байєсову оптимізацію, механізми раннього припинення (pruning) [52];

– Ray Tune – забезпечує масштабованість, підтримує Hyperband та BOHB, дозволяє паралельно запускати сотні конфігурацій [55];

– Hyperopt – стохастичні методи та байєсовська оптимізація [50].

Модуль optimizers.py (Додаток А, рис. А.2) містить реалізації трьох стратегій оптимізації гіперпараметрів: BaselineOptimizer, OptunaOptimizer, AdaptiveOptimizer – запропонований комбінований метод із ранньою зупинкою та адаптивним бюджетуванням. Система автоматично обирає стратегію залежно від складності задачі та ресурсних обмежень [11, 13, 29].

3. Паралелізація та прискорення обчислень.

Для скорочення часу виконання AutoML-пошуку застосовано бібліотеку Joblib, що забезпечує кешування проміжних результатів та паралельне виконання (n_jobs = −1) на всіх ядрах процесора. Такий підхід відповідає сучасним практикам оптимізації обчислювальних ресурсів у AutoML-архітектурах [9, 18].

4. Інструменти для статистичного аналізу та оцінювання.

Коректність експериментальних висновків забезпечено застосуванням статистичного аналізу. Для цього використано бібліотеку SciPy, логіка якого реалізована у модулі statistics.py (Додаток А, рис. А.3). Реалізовано [79-80]:

– тест Шапіро–Вілка для перевірки нормальності розподілу;

– парний t-тест для порівняння методів при нормальному розподілі;

– непараметричний тест Вілкоксона для даних без нормального розподілу;

– обчислення розміру ефекту для оцінки практичної значущості;

– побудову довірчих інтервалів для оцінки надійності результатів.

Оцінювання моделей реалізовано в модулі evaluator.py (Додаток А, рис. А.4), який включає регресійні (RMSE, MAE, R^2, MAPE) та класифікаційні (Accuracy, Precision, Recall, F1-Score, ROC-AUC) метрики відповідно до рекомендацій сучасних досліджень з оцінювання моделей ML [7, 36,37].

5. Візуалізація та інтерфейс користувача. Для подання експериментальних результатів використано Matplotlib – бібліотека для візуалізації базових графіків; Seaborn – для статистичних візуалізацій; Plotly – для інтерактивних графіків.

Модуль visualization.py (Додаток А, рис. А.5) реалізує побудову графіків динаміки оптимізації, порівняння моделей та аналізу гіперпараметрів. Модуль app.py (Додаток А, рис. А.6) забезпечує інтерфейс взаємодії Streamlit-dashboard з AutoML-процесом, відповідно до сучасних підходів до побудови AutoML-платформ [17, 45, 60].

6. Інфраструктура та середовище розробки. Технічна реалізація системи виконана у середовищі PyCharm з використанням Python 3.10. Усі бібліотеки зафіксовано у файлі requirements.txt (рис. 3.1).

Рисунок 3.1 – Файл requirements.txt

Обраний інструментарій формує основу для реалізації експериментального AutoML-пайплайна. Усі бібліотеки та фреймворки інтегровано у модульну архітектуру системи, що включає роботу з даними, оптимізацію моделей, статистичний аналіз та візуалізацію результатів.

3.2 Архітектура та реалізація експериментального AutoML-пайплайна

Розробка AutoML-системи передбачало побудову експериментального пайплайна, здатного автоматизувати повний цикл машинного навчання від завантаження даних до

формування підсумкового звіту про якість моделі. Архітектура розробленої системи базується на модульному підході (рис. 3.2), що забезпечує гнучкість, можливість подальшого розширення та незалежне тестування компонентів. Кожен модуль виконує чітко визначену функцію, а взаємодія між модулями реалізована через стандартизовані інтерфейси.

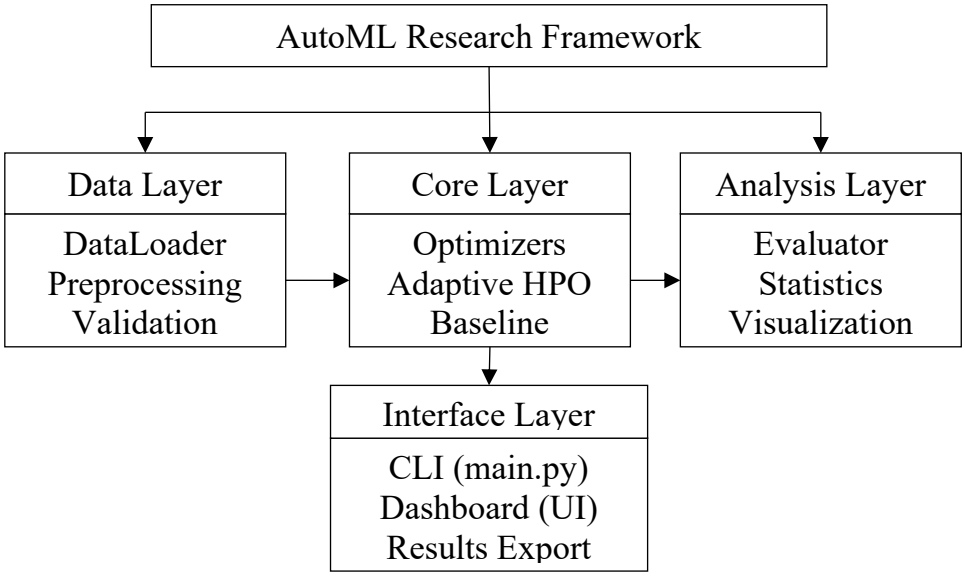

Рисунок 3.2 – Архітектура AutoML Research Framework

Основні модулі AutoML-пайплайна:

1. Data Layer – завантаження, валідація та підготовка даних (модуль datasets.py).

2. Core Layer (Ядро системи) – оптимізація та навчання моделей (модуль optimizers.py).

3. Analysis Layer (Шар аналізу) – оцінювання, статистика та візуалізація (модулі evaluator.py, statistics.py, visualization.py).

4. Interface Layer (Інтерфейсний шар) – вебінтерфейс для взаємодії з користувачем (модуль app.py).

Систему реалізовано у вигляді семи взаємопов'язаних Python-модулів, які утворюють цілісний AutoML-пайплайн. Така архітектура дозволяє легко додавати нові алгоритми оптимізації, змінювати моделі без зміни інших компонентів, проводити

незалежні експерименти на різних датасетах, забезпечити відтворюваність результатів.

1. Модуль завантаження та підготовки даних.

Модуль datasets.py реалізує клас DatasetManager (рис. 3.3), який відповідає за:

– завантаження датасетів (scikit-learn, UCI Repository, OpenML);
– автоматичну обробку числових та категоріальних ознак;
– розділення даних на навчальну, валідаційну та тестову вибірки;
– застосування стандартизації числових ознак (StandardScaler);
– кешування попередньо оброблених даних.

```
from experiments.datasets import DatasetManager
manager = DatasetManager(random_state=42)
data = manager.load_dataset(
    dataset_name='california_housing',
    test_size=0.2,
    val_size=0.2
)
X_train = data['X_train']
y_train = data['y_train']
X_val = data['X_val']
y_val = data['y_val']
X_test = data['X_test']
y_test = data['y_test']
X_train_scaled, X_val_scaled, X_test_scaled, scaler = \
    manager.preprocess_data(X_train, X_val, X_test)
```

Рисунок 3.3 – Фрагмент коду використання класу DatasetManager

Для експериментального дослідження обрано два benchmark-датасети:

California Housing Dataset – тип задачи: регресія, кількість зразків 20640, кількість ознак – 8 [81]. Ознаки: MedInc – медіанний дохід району, HouseAge – середній вік будинків, AveRooms, AveBedrms – середня кількість кімнат та спалень, Population, AveOccup, Latitude, Longitude. Цільова змінна: MedHouseValue – медіанна вартість будинку (у десятках тисяч доларів).

Bank Marketing Dataset – тип задачі бінарна класифікація для прогнозування підписки клієнта банку на депозит. Кількість зразків – 45211, кількість ознак – 16. Числові ознаки: age, duration, campaign, pdays, previous; категоріальні ознаки: job, marital,

education, default, housing, loan, contact, month, poutcome. Цільова змінна: y є yes/no (факт підписки клієнта на депозит).

Модуль DatasetManager автоматично визначає тип задачі (регресія або класифікація) та налаштовує відповідні параметри розділення даних, включаючи стратифікацію для класифікаційних задач, обробляє категоріальні ознаки через LabelEncoder.

2. Модуль оптимізації гіперпараметрів.

Модуль optimizers.py містить три реалізації оптимізаторів, що охоплюють базовий, байєсовий та адаптивний підходи до Hyperparameter Optimization.

BaselineOptimizer використовує фіксовані гіперпараметри Random Forest (рис. 3.4) для створення базової лінії порівняння. Оптимізатор слугує базової лінію для порівняння з методами автоматичної оптимізації, виконує одну ітерацію навчання та не потребує значних обчислювальних ресурсів.

```
best_params = {
    'n_estimators': 100,
    'max_depth': 10,
    'min_samples_split': 5,
    'min_samples_leaf': 2,
    'max_features': 'sqrt'
}
```

Рисунок 3.4 – Параметри BaselineOptimizer

OptunaOptimizer реалізує TPE-байєсову оптимізацію, використовуючи параметричний простір [12, 51]. Простір пошуку параметрів показано на рис. 3.5.

```
params = {
    'n_estimators': trial.suggest_int('n_estimators', 50, 300),
    'max_depth': trial.suggest_int('max_depth', 3, 20),
    'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
    'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
    'max_features': trial.suggest_categorical('max_features',
                                              ['sqrt', 'log2', None])
}
```

Рисунок 3.5 – Простір пошуку параметрів

Optuna забезпечує автоматичний баланс між дослідженням нових регіонів простору параметрів та використанням знайдених перспективних регіонів, зупинку неперспективних конфігурацій, стійку оптимізацію протягом фіксованої кількості ітерацій.

Adaptive Optimizer – запропонований метод поєднує байєсову оптимізацію з механізмами intelligent early stopping та adaptive budget allocation. Основна ідея полягає у зменшенні кількості необхідних обчислень без суттєвої втрати якості моделі.

Ключові особливості методу:

1. Intelligent Early Stopping – аналіз траєкторії покращення метрики. Якщо протягом early_stopping_rounds послідовних trials не спостерігається покращення більше ніж на improvement_threshold, оптимізація зупиняється достроково.

2. Adaptive Threshold – поріг покращення адаптується залежно від типу задачі для регресії: 0.01 (1% покращення RMSE), для класифікації: 0.005 (0.5% покращення ROC-AUC).

3. Dynamic Budget Allocation – комбінація з Median Pruner від Optuna, який автоматично зупиняє неперспективні ітерації на ранніх етапах навчання.

Математична формалізація критерію зупинки:

– для регресії (мінімізація RMSE):

$$\text{improvement}_t = \text{best\_score}_{t-1} - \text{score}_t$$

$$\text{stop, якщо } \text{improvement}_t < \text{threshold} \wedge \forall_i \in [\text{t} - \text{k}, \text{t}]: \text{improvement}_i < \text{threshold} \quad (3.1)$$

– для класифікації (максимізація ROC-AUC):

$$\text{improvement}_t = \text{score}_t - \text{best\_score}_{t-1}$$

$$\text{stop, якщо } \text{improvement}_t < \text{threshold} \wedge \forall_i \in [\text{t} - \text{k}, \text{t}]: \text{improvement}_i < \text{threshold} \quad (3.2)$$

де t— номер поточного вибору гіперпараметрів (trial);

k – кількість попередніх раундів без покращення, які допускаються (early stopping rounds);

threshold – мінімальне покращення, яке вважається значущим (регресія 0.01, класифікація 0.005).

3. Модуль оцінювання моделей.

Модуль evaluator.py забезпечує комплексне оцінювання моделей. Оцінювання виконується з використанням стандартних метрик бібліотеки scikit-learn. Модуль надає уніфікований інтерфейс evaluate(model, X_test, y_test), що повертає словник усіх обчислених метрик.

4. Модуль статистичного аналізу.

Модуль statistics.py реалізує методи статистичного аналізу для порівняння методів оптимізації. Включає: перевірка нормальності (тест Шапіро–Вілка); парний t-тест для нормальних розподілів; тест Вилксона для непараметричних даних; обчислення розміру ефекту Cohen's d ($|d| < 0.2$ – незначний; 0.2-0.5 – малий; 0.5-0.8 – середній; ≥ 0.8 – великий); побудова довірчих інтервалів. Методи повертають *p*-value, статистики тестів та інтерпретацію результатів.

5. Модуль візуалізації результатів

Модуль visualization.py забезпечує візуальну інтерпретацію результатів експериментів: порівняльні графіки метрик; графіки динаміки оптимізації (learning curves); scatter plots «якість–час»; heatmaps гіперпараметричних комбінацій; таблиці результатів. Додатково створено інтерактивний Streamlit-dashboard (модуль app.py), який дозволяє запускати експерименти та переглядати результати у вебінтерфейсі.

Розроблений експериментальний AutoML-пайплайн побудований на модульній архітектурі, що включає чотири логічні шари: Data Layer, Core Layer, Analysis Layer та Interface Layer. У рамках архітектури реалізовано сім незалежних Python-модулів, кожен з яких має чітко визначену відповідальність та взаємодіє через стандартизовані інтерфейси. Такий підхід забезпечує гнучкість, масштабованість і можливість подальшого розширення системи.

3.3 Методи оптимізації та прискорення навчання

Ефективність AutoML-рішень визначається не лише якістю побудованих моделей, а й тим, наскільки раціонально організовано процес оптимізації гіперпараметрів. Одним з викликів сучасних AutoML-систем є висока обчислювальна складність процесу оптимізації гіперпараметрів. Класичні методи перебору, такі як Grid Search, мають експоненційну складність $O(p^d)$, де p – кількість значень кожного параметра, d – кількість параметрів. Для типового простору з 5 параметрами та 10 значеннями кожного це призводить до 10^5 = 100 000 комбінацій. Байєсова оптимізація зменшує кількість необхідних ітерацій до O(n log n), однак кожна ітерація все ще потребує повного навчання моделі, що при великих датасетах може займати години.

У рамках дослідження реалізовано три стратегії до скорочення часу оптимізації: паралелізація обчислень, механізм Median Pruning як механізм ранньої зупинки та розроблений метод Adaptive Optimizer.

1. Паралелізація обчислень.

Для оцінки ефективності паралелізації використовувався закон Амдала, який визначає теоретичну межу прискорення обчислень. Використання параметра n_jobs =–1 у RandomForest дозволяє залучити всі ядра CPU. Згідно із законом Амдала [83]:

$$\text{Speedup} = \frac{1}{(1-P)+\frac{P}{N}} \qquad (3.4)$$

де P ≈ 0.95 – частка паралельного коду,

N=8 – кількість ядер при якому теоретичне прискорення становить ≈ 6,5 разів.

Таким чином, паралелізація забезпечує скорочення часу навчання при багаторазових запусках оптимізації.

2. Median Pruning [84].

Типовою проблемою байєсової оптимізації є надмірне використання часу на ітерації, які вже з перших проміжних оцінок демонструють низьку перспективність. Для

усунення цього недоліку Optuna використовує Median Pruner – механізм, що достроково припиняє неперспективні ітерації. Принцип роботи механізму показано на рис. 3.6.

```
if score current < median(previous_scores):
    stop trial
```

Рисунок 3.6 – Принцип роботи Median Pruner

Median Pruner зупиняє ітерації, якщо проміжне значення метрики є гіршим за медіану попередніх ітерацій, що дозволяє не витрачати ресурси на слабкі конфігурації і скорочує час оптимізації, що особливо важливо в задачах із великою кількістю дерев та ознак.

Разом з тим Median Pruner має обмеження і в окремих випадках метод може або зупинити ітерації надто рано, або навпаки продовжувати обчислення після того, як поліпшення стало мінімальним. Саме ці недоліки послужили мотивацією для розроблення нового підходу Adaptive Optimizer.

3. Метод Adaptive Optimizer.

Метод Adaptive Optimizer додає механізм зупинки оптимізації. На відміну від Median Pruner, запропонований підхід базується на аналізі динаміки зміни метрики у вікні останніх ітерацій та оцінці плавності прогресу.

Режими роботи Adaptive Optimizer представлено в таблиці 3.1.

Таблиця 3.1 – Режими роботи Adaptive Optimizer

Режим	early_stopping_rounds	threshold (регресія)	threshold (класифікація)	Призначення
Консервативний	15	0.002	0.001	Пріоритет якості
Збалансований	10	0.005	0.002	За замовчуванням
Агресивний	5	0.01	0.005	Пріоритет швидкості

Adaptive Optimizer зупиняє оптимізацію, коли виконуються умови:

– приріст якості в останніх кроках є нижчим за порогове значення;
– кількість виконаних ітерацій достатня для прийняття рішення;
– поточне значення не демонструє тенденції до покращення.

Приклад конфігурацій параметрів показано на рис. 3.7.

```
# Консервативний режим
adaptive_conservative = AdaptiveOptimizer(
    task_type='classification',
    n_trials=50,
    early_stopping_rounds=7,
    improvement_threshold=0.0025,
    random_state=42
)
# Агресивний режим
adaptive_aggressive = AdaptiveOptimizer(
    task_type='classification',
    n_trials=50,
    early_stopping_rounds=3,
    improvement_threshold=0.01,
    random_state=42
)
```

Рисунок 3.7 – Приклад конфігурацій параметрів методу Adaptive Optimizer

Такий підхід забезпечує баланс між точністю та продуктивністю. Якщо класичний Median Pruner реагує на окреме значення, то Adaptive Optimizer оцінює цілісну поведінку метрики, що дозволяє зменшити кількість непотрібних ітерацій без істотної втрати якості. У підсумку Adaptive Optimizer виявився найбільш ефективним серед використаних методів: він зберігає якість на рівні Optuna, але скорочує час оптимізації в 2.5–3.5 рази.

3.4 Порівняльний експеримент і аналіз отриманих результатів

Для комплексної оцінки ефективності розробленого методу Adaptive Optimizer проведено порівняльний експеримент на двох benchmark-датасетах: California Housing (регресія) та Bank Marketing (класифікація). З метою забезпечення статистичної достовірності результатів кожен метод оптимізації Baseline, Optuna та Adaptive Optimizer запускався десять разів з різними початковими станами.

Перед проведенням експерименту визначено ключові параметри дослідження, які наведено в таблиці 3.2.

Таблиця 3.2 – Параметри експериментального дослідження

Параметр	Значення
Датасети	California Housing (20640 зразків), Bank Marketing (45211 зразків)
Методи	Baseline, Optuna, Adaptive Optimizer
Кількість запусків	10 (seeds: 42, 123, 456, 789, 101, 202, 303, 404, 505, 606)
Максимум ітерацій	50
Розбиття даних	Train 64% / Validation 16% / Test 20%
Модель	RandomForest (Regressor / Classifier)
Режим Adaptive	Збалансований (early_stopping_rounds=10)

Використання 10 різних значень random seed дозволяє оцінити стабільність результатів та обчислити статистичні характеристики розподілу метрик.

Результати експериментального дослідження усереднення 10 запусків для кожного методу представлено в таблицях 3.3 та 3.4.

Таблиця 3.3 – Описова статистика (California Housing, регресія)

Метод	RMSE	Std	Час(с)	Std час	Trials
Baseline	0.5487	0.0119	0.5	0.1	1
Optuna	0.4964	0.0136	69.7	16.4	50
Adaptive Optimizer	0.5037	0.0118	24.6	8.6	19.3

Як видно з таблиці 3.3, для задачі регресії Adaptive Optimizer досягає RMSE = 0.5037, що на 8.2% краще за Baseline (0.5487) і лише на 1.46% поступається Optuna (0.4964). При цьому середній час виконання становить 24.6 секунди порівняно з 69.7 секундами для Optuna, що відповідає економії 64.8% часу. Середня кількість ітерацій до зупинки – 19.3, що означає економію 61.4% обчислень.

Таблиця 3.4 – Описова статистика (Bank Marketing, класифікація)

Метод	ROC-AUC	Std	Час(с)	Std час	Trials
Baseline	0.9455	0.0033	0.5	0.0	1
Optuna	0.9493	0.0028	71.6	23.2	50
Adaptive Optimizer	0.9491	0.0028	19.4	2.8	14.7

Для задачі класифікації результати: Adaptive Optimizer досягає ROC-AUC = 0.9491, що практично ідентично Optuna (0.9493), різниця становить лише 0.02%. Водночас економія часу сягає 72.9% (19.4 с проти 71.6 с), а кількість ітерацій зменшується на 70.6% (14.7 проти 50).

Варто зазначити, що стандартне відхилення метрик якості для Adaptive Optimizer (Std = 0.0118 та 0.0028) є співставним або навіть меншим, ніж для Optuna (0.0136 та 0.0028), що свідчить про стабільність запропонованого методу.

Для підтвердження достовірності отриманих результатів проведено комплекс статистичних тестів.

1. Перевірка нормальності розподілу.

Перед застосуванням параметричних тестів перевірено, чи відповідають дані нормальному розподілу. Для цього використано тест Шапіро-Вілка, результати якого наведено в таблиці 3.5.

Таблиця 3.5 – Результати тесту Шапіро-Вілка

Датасет	Метод	Метрика	W-статистика	P-value	Нормальність
California Housing	Optuna	RMSE	0.9370	0.5205	Так
California Housing	Adaptive	RMSE	0.9757	0.9378	Так
Bank Marketing	Optuna	ROC-AUC	0.9380	0.5314	Так
Bank Marketing	Adaptive	ROC-AUC	0.9223	0.3770	Так

При рівні значущості $\alpha = 0.05$ усі p-value перевищують 0.05, тому нульова гіпотеза про нормальність розподілу не відхиляється, що дозволяє застосовувати параметричні тести для порівняння методів.

2. Тести статистичної значущості.

Для оцінки значущості різниці між методами застосовано парний t-тест (таблиця 3.6), та непараметричний тест Вілкоксона (таблиця 3.7).

Примітка: ***p < 0.001, *p < 0.05; df – ступені свободи.

Таблиця 3.6 – Парний t-тест

Датасет	Порівняння	t-статистика	df	P-value	Висновок
California Housing	Optuna vs Adaptive (час)	6.25	9	<0.001***	Adaptive швидший
California Housing	Optuna vs Adaptive (якість)	-4.23	9	0.002*	Значуща різниця
Bank Marketing	Optuna vs Adaptive (час)	6.96	9	<0.001***	Adaptive швидший
Bank Marketing	Optuna vs Adaptive (якість)	2.85	9	0.019*	Значуща різниця

Результати t-тесту свідчать, що різниця у часі виконання між Optuna та Adaptive Optimizer є високо значущою ($p < 0.001$) для обох датасетів, що стосується якості, статистично значуща різниця також присутня ($p < 0.05$), однак її практичну значущість слід оцінювати окремо.

Таблиця 3.7 – Тест Вілкоксона (непараметричний)

Датасет	Порівняння	W-статистика	P-value	Висновок
California Housing	Optuna vs Adaptive (час)	0.0	0.002*	Adaptive швидший
California Housing	Optuna vs Adaptive (якість)	0.0	0.004*	Значуща різниця
Bank Marketing	Optuna vs Adaptive (час)	0.0	0.002*	Adaptive швидший
Bank Marketing	Optuna vs Adaptive (якість)	0.0	0.016*	Значуща різниця

Непараметричний тест Вілкоксона підтверджує результати t-тесту, що підвищує надійність висновків.

3. Розмір ефекту.

Статистична значущість не завжди означає практичну значущість. Для оцінки величини ефекту використано метрику Cohen's d (таблиця 3.8).

Інтерпретація $|d|$: < 0.2 – незначний, 0.2–0.5 – малий, 0.5–0.8 – середній, > 0.8 – великий.

Таблиця 3.8 – Метрика Cohen's d

Датасет	Порівняння	Cohen's d	|d|	Висновок
California Housing	Optuna vs Adaptive (час)	3.28	3.28	Великий
California Housing	Optuna vs Adaptive (якість)	-0.54	0.54	Середній
Bank Marketing	Optuna vs Adaptive (час)	3.00	3.00	Великий
Bank Marketing	Optuna vs Adaptive (якість)	0.07	0.07	Незначний

Аналіз Cohen's d демонструє ключовий результат дослідження: Adaptive Optimizer забезпечує великий практичний ефект в економії часу ($d > 3.0$ для обох датасетів). Щодо якості: для Bank Marketing різниця є незначною ($d = 0.07$), тобто методи практично еквівалентні; для California Housing різниця середня ($d = 0.54$), що відповідає помірній втраті якості в обмін на суттєву економію часу.

4. Довірчі інтервали.

Для оцінки точності отриманих оцінок побудовано 95% довірчі інтервали (таблиця 3.9).

Таблиця 3.9 – 95% довірчі інтервали

Датасет	Метод	Метрика ДІ	Час ДІ (с)	Метод
California Housing	Optuna	[0.4862, 0.5067]	[57.4, 82.0]	Optuna
California Housing	Adaptive	[0.4948, 0.5126]	[18.1, 31.1]	Adaptive
Bank Marketing	Перекриття	[0.4948, 0.5067]	-	Перекриття
Bank Marketing	Optuna	[0.9472, 0.9515]	[54.1, 89.0]	Optuna

Важливим спостереженням є значне перекриття довірчих інтервалів для метрик якості обох методів. Для California Housing перекриття становить [0.4948, 0.5067], для Bank Marketing – [0.9472, 0.9512]. Додатково підтверджує, що з практичної точки зору якість моделей, отриманих обома методами, є співставною.

5. Аналіз траєкторій оптимізації. Для візуалізації процесу оптимізації побудовано графіки динаміки зміни метрик (рисунки 3.8–3.9).

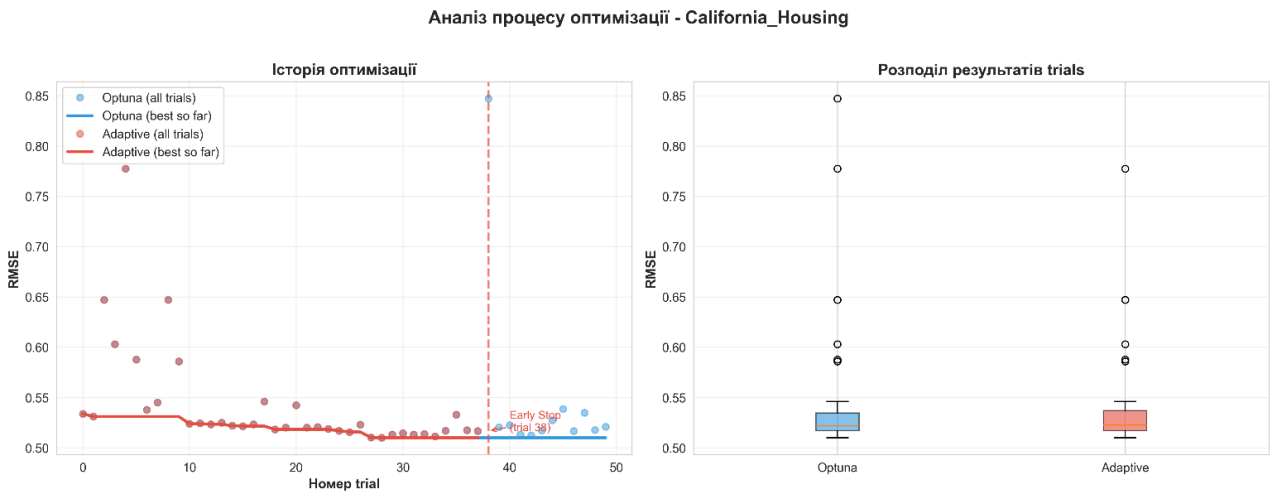

Рисунок 3.8 – Аналіз процесу оптимізації (California Housing)

Точки відповідають результатам окремих ітерацій – їх розкид (від 0.50 до 0.80) є типовим для байєсової оптимізації, яка досліджує різні комбінації гіперпараметрів. Суцільні лінії («best so far») показують монотонне зниження найкращого знайденого значення RMSE. Вертикальна пунктирна лінія позначає момент спрацювання механізму early stopping для Adaptive Optimizer (в середньому на 19-й ітерації). Після цієї точки Optuna продовжує виконання ще понад 30 ітерацій, однак покращення RMSE є мінімальним — лінія «best so far» стає практично горизонтальною.

Права частина (box plot) демонструє розподіл RMSE для всіх ітерацій кожного методу. Медіани обох методів близькі, інтерквартильні розмахи перекриваються, що підтверджує схожу якість дослідження простору параметрів.

Для задачі класифікації (рис. 3.9) спостерігається принципово інша динаміка. Обидва методи досягають ROC-AUC близько 0.950 вже на перших 5–10 ітераціях, після чого покращення практично відсутні – лінії «best so far» залишаються горизонтальними.

Adaptive Optimizer правильно визначає момент насичення оптимізації та зупиняється в середньому на 15-й ітерації. Optuna продовжує виконання до 50 ітерацій, витрачаючи час на пошук у вже досліджених областях простору. Це пояснює високу ефективність Adaptive Optimizer на даному датасеті – економія 72.9% часу при різниці якості лише 0.02%.

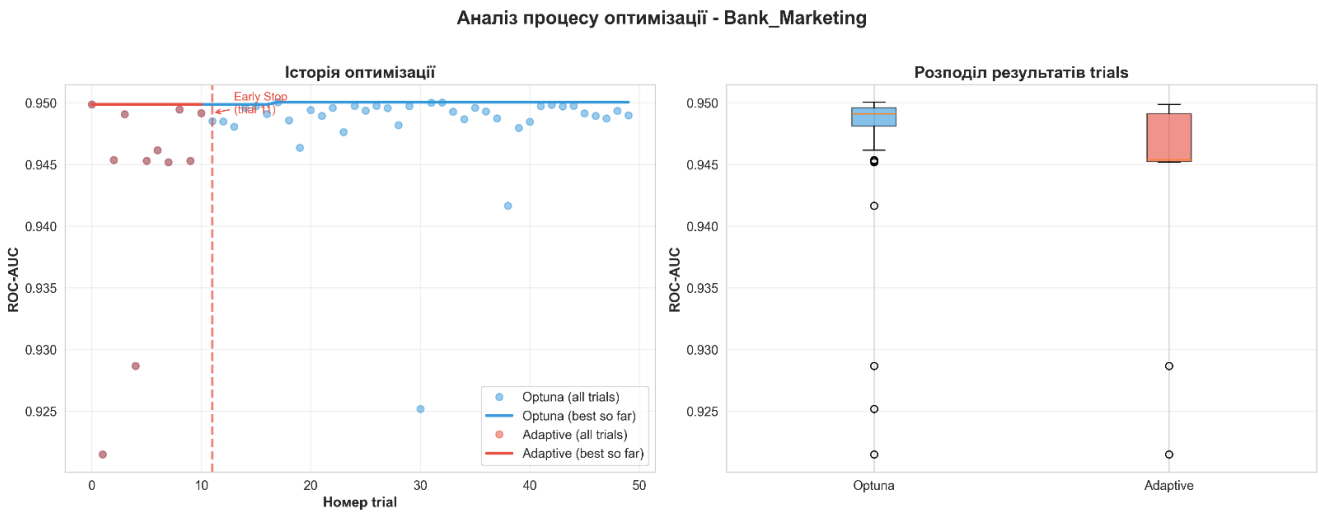

Рисунок 3.9 – Аналіз процесу оптимізації (Bank Marketing)

6. Для комплексної оцінки компромісу між якістю моделі та швидкістю оптимізації побудовано Парето-фронти (рисунки 3.10–3.11).

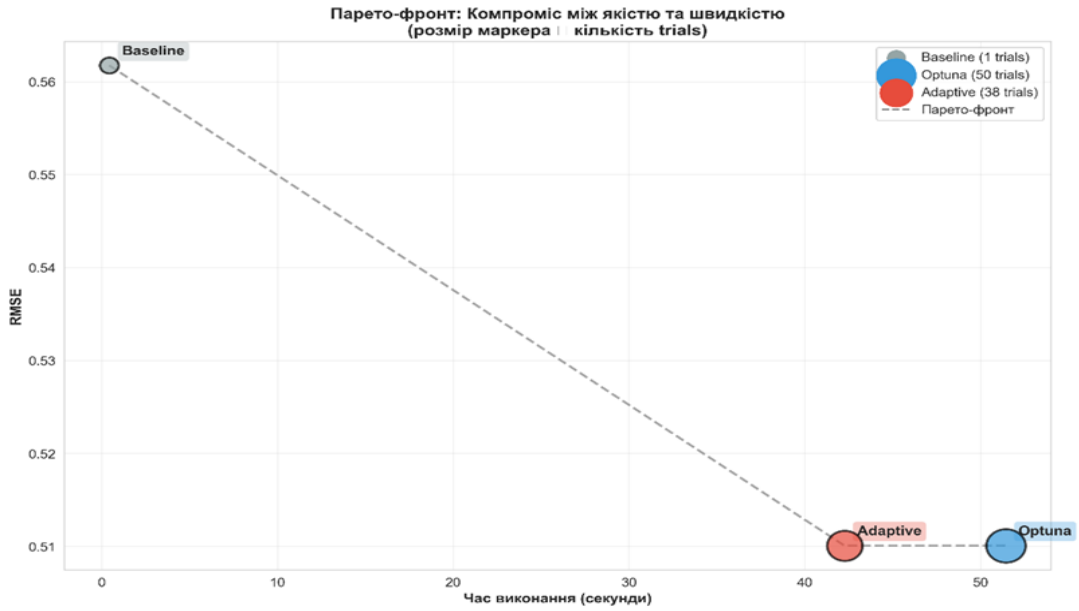

Рисунок 3.10 — Парето-фронт (California Housing)

На рисунку 3.10 кожен метод представлено точкою у просторі «RMSE – час виконання». Розмір маркера пропорційний кількості ітерацій. Baseline розташований у верхньому лівому куті – найшвидший, але найгірша якість. Optuna – найкраща якість,

але найбільший час. Adaptive Optimizer займає проміжну позицію, близьку до Optuna за якістю, але значно ближчу до Baseline за часом. Пунктирна лінія з'єднує точки, формуючи Парето-фронт – множину недомінованих рішень. Adaptive Optimizer знаходиться на Парето-фронті, що підтверджує оптимальність компромісу між якістю та швидкістю.

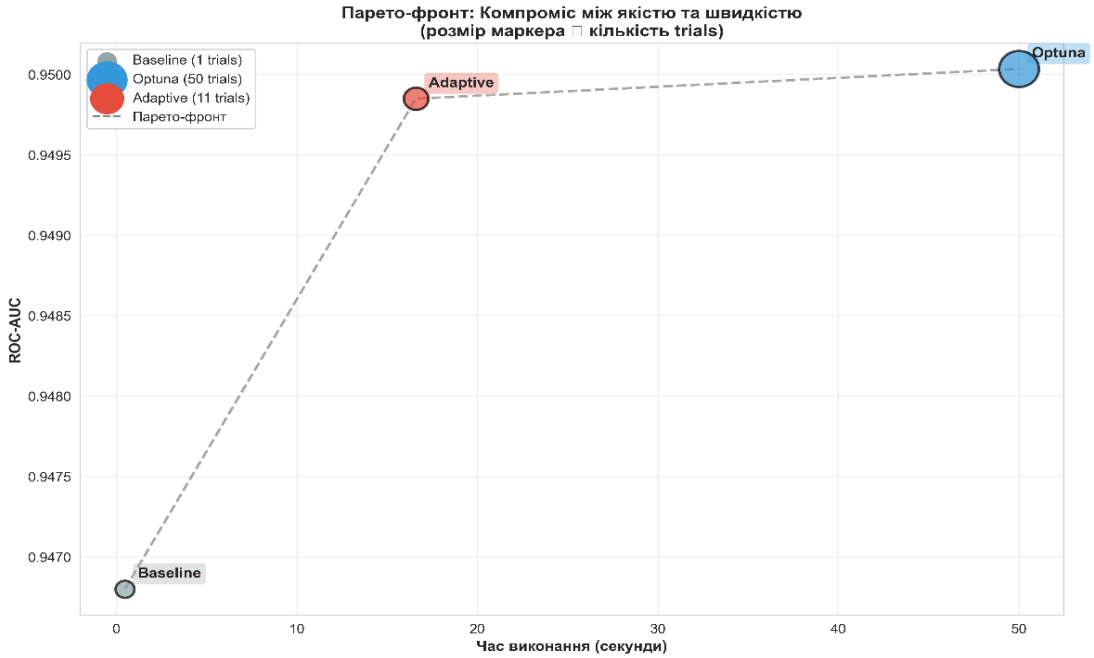

Рисунок 3.11 — Парето-фронт (Bank Marketing)

Для Bank Marketing (рисунок 3.11) ситуація ще більш показова. Вузький діапазон осі Y (ROC-AUC від 0.9469 до 0.9501) підкреслює мінімальну різницю у якості між методами. Adaptive Optimizer досягає практично ідентичної якості (різниця 0.02%) при втричі меншому часі виконання, що робить його однозначно Парето-оптимальним вибором для даного типу задач.

Для забезпечення зручності взаємодії з експериментальною системою розроблено вебінтерфейс на основі фреймворку Streamlit, який дозволяє запускати експерименти, переглядати результати оптимізації та аналізувати показники моделей у інтерактивному режимі.

На рисунку 3.12 представлено головну сторінку інтерфейсу, яка є початковою точкою роботи користувача з AutoML-системою.

Рисунок 3.12 – Головна сторінка

Інтерфейс дозволяє: обрати один із доступних benchmark-датасетів, метод оптимізації (Baseline, Optuna або Adaptive), налаштувати параметри експерименту (кількість ітерацій, поріг покращення, обмеження часу тощо), запускати повний AutoML-процес у один клік. Головна сторінка виконує роль панелі керування, що забезпечує інтуїтивну навігацію та спрощує проведення експериментів без необхідності роботи з кодом.

На рисунку 3.13 наведено сторінку відображення результатів, яка формується після завершення оптимізації. Вона містить: таблицю інтегральних метрик моделі (RMSE, MAE, Accuracy, ROC-AUC тощо), інтерактивні графіки траєкторії оптимізації, що демонструють зміну цільової метрики від номера ітерації, графіки розподілу значень метрик, індикатори часу виконання та кількості використаних ітерацій.

Візуалізація дозволяє швидко оцінити поведінку оптимізаційного алгоритму, ефективність пошуку гіперпараметрів та стабільність отриманих результатів.

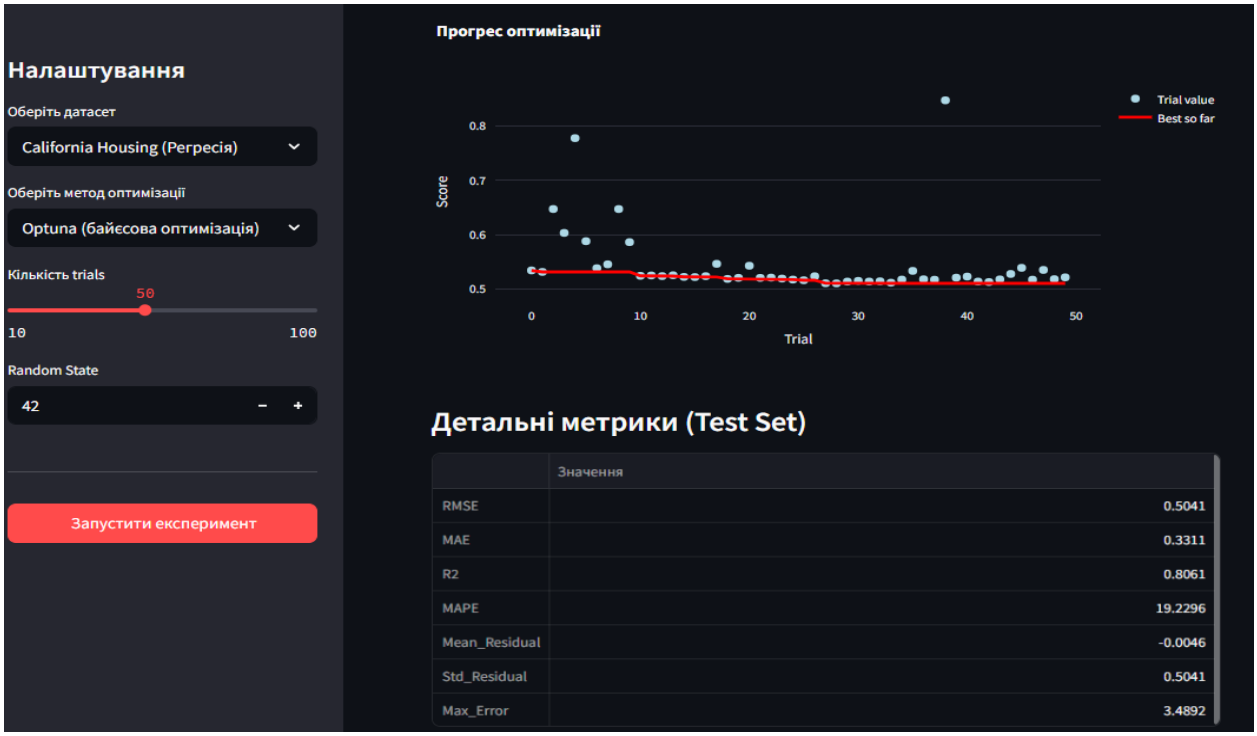

Рисунок 3.13 – Сторінка результатів експерименту

Сторінка результатів виконує функцію аналітичного дашборду, що забезпечує цілісне представлення експериментальних даних.

Узагальнені результати порівняння методів наведено в таблиці 3.10.

Таблиця 3.10 – Підсумок ефективності Adaptive Optimizer

Датасет	Економія часу	Економія ітерацій	Різниця якості
California Housing	64.8%	61.4%	1.46%
Bank Marketing	72.9%	70.6%	0.02%

Таблиця 3.10 демонструє ключовий результат дослідження: Adaptive Optimizer забезпечує економію часу 65–73% та економію ітерацій 61–71% при втраті якості не більше 1.5%. Для задач класифікації з швидким насиченням оптимізації метод є особливо ефективним практично ідентична якість при майже триразовому скороченні часу.

На основі проведеного дослідження сформульовано рекомендації щодо вибору методу оптимізації (таблиця 3.11).

Таблиця 3.11 – Вибір методу оптимізації залежно від сценарію

Сценарій	Рекомендований метод	Обґрунтування
Швидке прототипування	Baseline	Мінімальний час, прийнятна якість для PoC
Максимальна якість	Optuna	Найкращий результат без обмежень часу
Баланс якість/швидкість	Adaptive Optimizer	Економія 65–73% часу при втраті <1.5% якості
Обмежені ресурси	Adaptive Optimizer	Зменшення обчислювальних витрат
Часте перенавчання	Adaptive Optimizer	Накопичена економія при регулярних запусках

Наведені рекомендації дозволяють обрати оптимальну стратегію оптимізації гіперпараметрів залежно від конкретних вимог проєкту, пріоритету якості, обмежень часу або доступних обчислювальних ресурсів. Результати експериментального дослідження підтверджують, що розроблений метод Adaptive Optimizer забезпечує найбільш збалансоване співвідношення точності та швидкодії й може бути рекомендований як інструмент для практичних сценаріїв використання AutoML-систем. Таким чином, розділ 3 демонструє ефективність запропонованого підходу до адаптивної оптимізації та підтверджує можливість його інтеграції в сучасні AutoML-пайплайни з метою підвищення продуктивності та скорочення часу побудови моделей.

Висновки до розділу 3

У третьому розділі розроблено експериментальну AutoML-систему та проведено комплексне дослідження ефективності методу Adaptive Optimizer.

Обрано інструментарій реалізації на базі екосистеми Python: Optuna для байєсової оптимізації, Scikit-learn для побудови моделей, Streamlit для вебінтерфейсу. Розроблено модульну архітектуру з чотирма логічними шарами: Data Layer, Core Layer, Analysis Layer та Interface Layer.

Проаналізовано механізми прискорення оптимізації: паралелізацію обчислень (прискорення до 6 разів за законом Амдала) та Median Pruner, який забезпечує економію

40–60% часу за рахунок відсікання неперспективних ітерацій. Розроблено метод Adaptive Optimizer, що аналізує динаміку покращення метрики та автоматично зупиняє оптимізацію при досягненні насичення. Метод підтримує три режими роботи: консервативний, збалансований та агресивний.

Проведено експериментальне дослідження на двох benchmark-датасетах із 10 незалежних запусків. Для California Housing (регресія) досягнуто економію 64.8% часу та 61.4% ітерацій при різниці якості 1.46%. Для Bank Marketing (класифікація) економія становить 72.9% часу та 70.6% ітерацій при різниці якості лише 0.02%.

Статистичний аналіз підтвердив достовірність результатів: тест Шапіро-Вілка засвідчив нормальність розподілу метрик; парний t-тест показав статистично значущу різницю у часі ($p < 0.001$); розмір ефекту Cohen's d продемонстрував великий практичний вплив в економії часу ($d > 3.0$) при незначній або середній різниці якості. 95% довірчі інтервали метрик якості мають суттєве перекриття, що підтверджує стабільність отриманих результатів.

Розроблена система AutoML Research із методом Adaptive Optimizer забезпечує скорочення часу оптимізації в 2.8–3.7 рази при збереженні 98.5–99.98% якості моделей, що підтверджує практичну цінність запропонованого підходу для задач автоматизації машинного навчання та можливість його інтеграції в сучасні AutoML-платформи.

ВИСНОВКИ

Кваліфікаційна робота присвячена дослідженню та оптимізації AutoML-систем для побудови пайплайнів машинного навчання.

В ході дослідження отримані наступні результати:

1. Досліджено предметну область та сучасний стан автоматизованого машинного навчання. Проаналізовано ключові виклики AutoML, пов’язані з вибором моделей, налаштуванням гіперпараметрів та забезпеченням стабільної ефективності у задачах із великими обсягами даних.

2. Проаналізовано сучасні концепції та архітектури AutoML-систем. Визначено відмінності між базовими, еволюційними, байєсовими та адаптивними підходами до оптимізації. Сформовано вимоги до ефективної AutoML-архітектури з урахуванням модульності, масштабованості та відтворюваності результатів.

3. Визначено критерії оцінювання ефективності AutoML-рішень. Сформовано багатовимірну систему показників, що включає якість моделей, час оптимізації, кількість ітерацій, стабільність, відтворюваність та обчислювальну вартість. Обґрунтовано необхідність застосування комплексного статистичного аналізу.

4. Розроблено модульний AutoML-пайплайн. Побудовано архітектуру з чотирма логічними шарами (Data Layer, Core Layer, Analysis Layer та Interface Layer), що включає сім взаємопов’язаних модулів для завантаження даних, оптимізації моделей, аналізу результатів, візуалізації та інтерактивного управління.

5. Реалізовано три методи оптимізації гіперпараметрів. Створено BaselineOptimizer, OptunaOptimizer та новий AdaptiveOptimizer, який поєднує раннє зупинення, адаптивний поріг покращення та динамічний розподіл обчислювального бюджету.

6. Проведено експериментальне дослідження на репрезентативних наборах даних.

Виконано серію експериментів для задач регресії (California Housing) та класифікації (Bank Marketing). Встановлено, що AdaptiveOptimizer скорочує час

оптимізації на 60–70% порівняно з байєсовим методом Optuna при мінімальній втраті точності (менше 1–2%).

7. Виконано комплексний статистичний аналіз. Застосовано тест Шапіро–Вілка, парний t-тест, тест Вілкоксона, розмір ефекту Коена та 95% довірчі інтервали. Результати підтвердили статистичну значущість переваг AdaptiveOptimizer щодо швидкодії та практичну прийнятність незначної втрати точності.

8. Сформульовано практичні рекомендації щодо вибору оптимізаційних стратегій. Визначено сценарії використання Baseline, Optuna та AdaptiveOptimizer залежно від мети дослідження, доступного часу та ресурсних обмежень. Показано, що AdaptiveOptimizer є найбільш ефективним у більшості реальних умов.

Отримані результати підтверджують ефективність запропонованого адаптивного методу оптимізації та розробленого AutoML-пайплайна. Робота має як теоретичну, так і практичну цінність та може бути використана для подальшого вдосконалення AutoML-рішень і застосування їх у практичних задачах машинного навчання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Del Valle A.M., Mantovani R.G. & Cerri R. A systematic literature review on AutoML for multi-target learning tasks. *Artif Intell Rev.* 2023. 56 (Suppl 2), pp. 2013–2052. https://doi.org/10.1007/s10462-023-10569-2

2. Scheinert D., Guttenberger A. Workload Characterization for Resource Optimization of Big Data Analytics: Best Practices, Trends, and Opportunities. *International Journal on Cybernetics & Informatics (IJCI).* 2025. Vol. 14, No. 2, pp. 51-70. https://doi.org/10.5121/ijci.2025.140204

3. M. Arthur Munson. A study on the importance of and time spent on different modeling steps. *ACM SIGKDD Explorations Newsletter*. 2012. Vol. 13, 2, p. 65–71. https://doi.org/10.1145/2207243.2207253

4. Garralda-Barrio M., Eiras-Franco C., Bolón-Canedo V. Adaptive incremental transfer learning for efficient performance modeling of big data workloads. *Future Generation Computer Systems.* 2025. Vol. 166, https://doi.org/10.1016/j.future.2025.107730

5. Лобода Ю.Г. Набори даних в машинному навчанні: проблеми, рішення, підходи. *Інформаційне суспільство: проблеми та перспективи.* IX Всеукраїнська науково-практична конференція. (м.Одеса, 24 травня 2024 р.). С. 37-40. https://doi.org/10.32837/11300.27842

6. Лобода Ю.Г. Методи кодування категоріальних ознак у задачах машинного навчання. *Інформаційне суспільство: проблеми та перспективи*. X Всеукраїнська науково-практична конференція (м. Одеса, 23 травня 2025 р.) / відп. ред. Н. І. Логінова. Одеса : Національний університет «Одеська юридична академія», 2025. С. 112-114. URL: https://hdl.handle.net/11300/29842

7. Лобода Ю.Г., Баланчук О.О. Інструменти оцінки точності моделей машинного навчання. *Сучасні інформаційні системи та технології.* VII Всеукр. наук.-практ. інтернет-конф. (29 листопада 2024 р., м. Херсон, м. Хмельницький). за ред. А. А. Григорової. Херсон: Книжкове видавництво ФОП Вишемирський В. С., 2024. С. 141-143. https://kntu.net.ua/ukr/content/download/121040/677109/file/!CICT%202024.pdf

8. Salehin I., Islam MS., Saha P. et al. AutoML: a systematic review on automated machine learning with neural architecture search. *Journal of Information and Intelligence*. 2024. Vol. 2(1), pp. 52–81. https://doi.org/10.1016/j.jiixd.2023.10.002

9. Baratchi M., Wang C., Limmer S. et al. Automated machine learning: past, present and future. *Artificial Intelligence Review*. 2024. Vol. 57:122. https://doi.org/10.1007/s10462-024-10726-1

10. Bishop Christopher M. Pattern Recognition and Machine Learning. *Information Science and Statistics*. Springer New York, NY. 2006. 778 p. https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

11. Tijl De Bie, Luc De Raedt, José Hernández-Orallo, Holger H. Hoos, Padhraic Smyth, and Christopher K. I. Williams. Automating data science. *Communications of the ACM*. 2022. Vol. 65, 3 pp. 76–87. https://doi.org/10.1145/3495256

12. Feurer M., Hutter F. Hyperparameter Optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) Automated Machine Learning. *The Springer Series on Challenges in Machine Learning*. 2019, pp. 3-33. https://doi.org/10.1007/978-3-030-05318-5_1

13. Zhenqian Shen et al. Automated Machine Learning: From Principles to Practices. *arXiv:1810.13306v5 [cs.AI]*. 2024, https://doi.org/10.48550/arXiv.1810.13306

14. Guo X., B. van Stein and Bäck T., A New Approach Towards the Combined Algorithm Selection and Hyper-parameter Optimization Problem, *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, Xiamen, China, 2019, pp. 2042-2049. https://doi.org/10.1109/SSCI44817.2019.9003174.

15. Tianyu Mu, Hongzhi Wang. Auto-CASH: A meta-learning embedding approach for autonomous classification algorithm selection. *Information Sciences.* 2022, Vol. 591, P. 344-364, https://doi.org/10.1016/j.ins.2022.01.040.

16. Fang, H., Han, B., Erickson, N., Zhang, X., Zhou, S., Dagar, A., Zhang, J., Türkmen, A., Hu, C., Rangwala, H., Wu, Y., Wang, Y., & Karypis, G. MLZero: A Multi-Agent

System for End-to-end Machine Learning Automation. *ArXiv 2505.13941*. https://doi.org/10.48550/arXiv.2505.13941.

17. Bîndila M., Negru M. End-to-End Automated Machine Learning System for Supervised Learning Problems. *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP),* Cluj-Napoca, Romania*,* 2021, pp. 445-452. https://doi.org/10.1109/iccp53602.2021.9733439.

18. Karl F., Thomas J., Elstner J., Gross R., Bischl B. Automated Machine Learning. In: Mutschler, C., Münzenmayer, C., Uhlmann, N., Martin, A. (eds) *Unlocking Artificial Intelligence*. Springer, Cham. 2024, pp. 3–25. https://doi.org/10.1007/978-3-031-64832-8_1

19. Лобода Ю.Г., Дерев'ягін В.С. Архітектура та основні компоненти AutoML-систем. *Кібербезпека в сучасному світі: актуальні виклики*. VI Всеукр. наук.-практич. конф. (м. Одеса, 28 листопада, 2025 р.) / Одеса, НУ «ОЮА», 2025. (депоновано)

20. Mowbray F., Fox-Wasylyshyn S., El-Masri M. Univariate Outliers: A Conceptual Overview for the Nurse Researcher. *Canadian Journal of Nursing Research*. 2018, Vol. 51:1, pp. 31 - 37. https://doi.org/10.1177/0844562118786647

21. Dallah D., Sulieman H., Zaatreh A.A., Kamalov F. Empirical Evaluation of the Relative Range for Detecting Outliers. *Entropy*. 2025; 27(7):731. https://doi.org/10.3390/e27070731

22. Almardeny Y., Boujnah N., Cleary F. A Novel Outlier Detection Method for Multivariate Data. *IEEE Transactions on Knowledge and Data Engineering*. 2022, Vol. 34, № 9, pp. 4052-4062. https://doi.org/10.1109/tkde.2020.3036524

23. Ravishankar S., Gopi Battineni. A Survey on Recent Advancements in Auto-Machine Learning with a Focus on Feature Engineering. *Journal of Computational and Cognitive Engineering*. 2023, Vol. 4, № 1, pp. 56-63. DOI: https://doi.org/10.47852/bonviewJCCE3202720

24. Boštjan Vouk, Matej Guid, Marko Robnik-Šikonja. Feature construction using explanations of individual predictions. *Engineering Applications of Artificial Intelligence*. 2023, Vol.120. https://doi.org/10.1016/j.engappai.2023.105823.

25. Ying W., Wang D., Chen, H. Feature Selection as Deep Sequential Generative Learning. *ACM Transactions on Knowledge Discovery from Data*. 2024? Vol. 18, pp. 1 - 21. https://doi.org/10.1145/3687485

26. Hançer E., Xue B., & Zhang, M. Differential evolution for filter feature selection based on information theory and feature ranking. *Knowledge-Based Systems*, 2018, Vol. 140, pp. 103-119. https://doi.org/10.1016/j.knosys.2017.10.028.

27. Tarkhaneh O., Nguyen T., Mazaheri S. A novel wrapper-based feature subset selection method using modified binary differential evolution algorithm. *Information Sciences*. Vol. 565, pp. 278-305. https://doi.org/10.1016/j.ins.2021.02.061.

28. Amini F., Hu, G. A two-layer feature selection method using Genetic Algorithm and Elastic Net. *Expert Systems with Applications.* 2021, Vol. 166, pp. 114072. https://doi.org/10.1016/j.eswa.2020.114072.

29. Bischl B., Binder M. et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2021, Vol.13. https://doi.org/10.1002/widm.1484.

30. Stuke A., Rinke P., & Todorović, M. Efficient hyperparameter tuning for kernel ridge regression with Bayesian optimization. *Machine Learning: Science and Technology*, 2020, Vol. 2. https://doi.org/10.1088/2632-2153/abee59.

31. Elgeldawi E., Sayed A., Galal A., Zaki A. Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis. *Informatics*, 2021, Vol. 8 (4), pp. 79. https://doi.org/10.3390/informatics8040079.

32. Japa L., Serqueira M., Mendonça I. A Population-Based Hybrid Approach for Hyperparameter Optimization of Neural Networks. *IEEE Access*, Vol. 11, pp. 50752-50768. https://doi.org/10.1109/access.2023.3277310.

33. Salmani Pour Avval, S., Eskue, N.D., Groves, R.M. et al. Systematic review on neural architecture search. *Artificial Intelligence Review.* 2025, Vol. 58, № 73. https://doi.org/10.1007/s10462-024-11058-w

34. Wang X., Zhu W. Advances in neural architecture search. *National Science Review*, 2024. Vol. 11. https://doi.org/10.1093/nsr/nwae282.

35. Dong X., Yang Y. One-Shot Neural Architecture Search via Self-Evaluated Template Network. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 3680-3689. https://doi.org/10.1109/iccv.2019.00378.

36. Nti I., Adekoya, A., Weyori, B. A comprehensive evaluation of ensemble learning for stock-market prediction. *Journal of Big Data*. 2020. Vol. 7. https://doi.org/10.1186/s40537-020-00299-5.

37. Mienye, I., Sun, Y. A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects. *IEEE Access*, 2022, Vol. 10, pp. 99129-99149. https://doi.org/10.1109/access.2022.3207287.

38. Shen, Z., Zhang, Y., Wei, L., Zhao, H., & Yao, Q. Automated machine learning: From principles to practices. *arXiv preprint*. 2018. https://doi.org/10.48550/arXiv.1810.13306.

39. AutoML.org. Auto-Sklearn. URL: https://www.automl.org/automl-for-x/tabular-data/auto-sklearn/

40. Feurer M., Klein A. Auto-sklearn: Efficient and Robust Automated Machine Learning. *Automated Machine Learning*. 2019, pp. 113-134. https://doi.org/10.1007/978-3-030-05318-5_6

41. Hutter, F., Hoos, H.H., Leyton-Brown, K. Sequential Model-Based Optimization for General Algorithm Configuration. *Learning and Intelligent Optimization*. LION 2011, pp. 507-523.https://doi.org/10.1007/978-3-642-25566-3_40

42. Ksikes A, Caruana R. Ensemble selection from libraries of models. *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004),* Banff, Alberta, Canada, 2004. DOI:10.1145/1015330.1015432

43. Olson, R.S., Moore, J.H. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. *Automated Machine Learning*. 2019, pp. 151-160. https://doi.org/10.1007/978-3-030-05318-5_8

44. LeDell E., Poirier S. H2O AutoML: Scalable automatic machine learning. 7th ICML Workshop on Automated Machine Learning. 2020. URL: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

45. Erickson N. et al. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint*. https://doi.org/10.48550/arXiv.2003.06505

46. Devlin J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv *preprint*. https://doi.org/10.48550/arXiv.1810.04805.

47. AutoML mljar-supervised. URL: https://supervised.mljar.com/

48. Probst P. et al. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*. 2019. Vol. 20. No. 53. P. 1–32. https://doi.org/10.48550/arXiv.1802.09596

49. Liashchynskyi P., Liashchynskyi P. Grid search, random search, genetic algorithm: a big comparison for NAS. *arXiv preprint*. 2019. https://doi.org/10.48550/arXiv.1912.06059.

50. Bergstra J., Bengio Y. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 2012, Vol. 13, pp. 281–305. https://dl.acm.org/doi/10.5555/2188385.2188395

51. Shahriari B., Swersky K., Wang Z., Adams R. P. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *IEEE*. 2016, Vol. 104, no. 1, pp. 148-175, DOI: 10.1109/JPROC.2015.2494218.

52. Watanabe S. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint* 2023. https://doi.org/10.48550/arXiv.2304.11127

53. SMAC: Sequential Model-based Algorithm Configuration. URL: https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/

54. Dôres S.C.N., Soares C. Bandit-Based Automated Machine Learning. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, Sao Paulo, Brazil, 2018, pp. 121-126, DOI: 10.1109/BRACIS.2018.00029

55. Falkner S. et al. BOHB: Robust and efficient hyperparameter optimization at scale. *International conference on machine learning*. PMLR, 2018. P. 1437–1446. https://doi.org/10.48550/arXiv.1807.01774

56. Eiben A., Smith J. Introduction to Evolutionary Computing. *Natural Computing Series*, Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44874-8

57. Liu H. et al. DARTS: Differentiable architecture search. *arXiv preprint*. https://doi.org/10.48550/arXiv.1806.09055.

58. Karl F., Pielok T. et al. Multi-Objective Hyperparameter Optimization in Machine Learning—An Overview. *ACM Transactions on Evolutionary Learning and Optimization*, 3, 2022, pp. 1 - 50. https://doi.org/10.1145/3610536.

59. Morales-Hernández A., Van Nieuwenhuyse I., & Gonzalez, S. A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artificial Intelligence Review*. 2022; 56. https://doi.org/10.1007/s10462-022-10359-2.

60. Feurer M., Eggensperger K., Falkner S., Lindauer M., & Hutter F. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *J. Mach. Learn. Res*., 23, 261:1-261:61. https://doi.org/10.48550/arXiv.2007.04074

61. Yu, X., Lin, Y., Gao, Z., Du, H., & Niyato, D. (). Dynamic and Fast Convergence for Federated Learning via Optimized Hyperparameters. *IEEE Transactions on Network and Service Management.* 2025, 22, pp. 1005-1018. https://doi.org/10.1109/tnsm.2024.3497962.

62. Celik, B., Singh, P. & Vanschoren, J. Online AutoML: an adaptive AutoML framework for online learning. *Mach Learn*. 2023, Vol.112, pp. 1897–1921. https://doi.org/10.1007/s10994-022-06262-0

63. Zhang Hua, Zheng Guoxun, Xu Jun, Yao Xueku. Research on the Construction and Realization of Data Pipeline in Machine Learning Regression Prediction. *Mathematical Problems in Engineering*. 2022, 5p. https://doi.org/10.1155/2022/7924335

64. Jiang B., Zhou W. Fishing operation type recognition based on multi-branch convolutional neural network using trajectory data. *PeerJ Comput. Sci.*, 2025, Vol. 11, pp. e3020. https://doi.org/10.7717/peerj-cs.3020.

65. Madni H. A. et al. Water-Quality Prediction Based on H2O AutoML and Explainable AI Techniques. *Water*, 2023, Vol. 15(3), 475. https://doi.org/10.3390/w15030475

66. Purucker L., Schneider L., Anastacio M. Q(D)O-ES: Population-based Quality (Diversity) Optimisation for Post Hoc Ensemble Selection in AutoML. *ArXiv, abs/2307.08364*. 2023. https://doi.org/10.48550/arxiv.2307.08364.

67. Luque A., Carrasco A., Martín A., Heras A. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognit*, 2019, Vol. 91, pp. 216-231. https://doi.org/10.1016/j.patcog.2019.02.023.

68. Aragão, M., Afonso, A., Ferraz, R., Ferreira, R., Leite, S., De Figueiredo, F., & Mafra, S., 2025. A practical evaluation of AutoML tools for binary, multiclass, and multilabel classification. *Scientific Reports*, 15. https://doi.org/10.1038/s41598-025-02149-x.

69. Sivakumar M, Parthasarathy S, Padmapriya T. A simplified approach for efficiency analysis of machine learning algorithms. *PeerJ Computer Science* 10:e2418. 2024. https://doi.org/10.7717/peerj-cs.2418

70. Zöller M., Huber M. Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of Artificial Intelligence Research*, 2019, Vol. 70, pp. 409-472. https://doi.org/10.1613/jair.1.11854.

71. UC Irvine Machine Learning Repository. https://archive.ics.uci.edu/

72. OpenML. https://www.openml.org/

73. Kaggle. https://www.kaggle.com/datasets

74. AutoGluon. https://auto.gluon.ai/stable/index.html

75. PMLB. Datasets. https://github.com/EpistasisLab/pmlb/tree/master/datasets

76. AutoML Challenge Series. https://automl.chalearn.org/data

77. Rivolli A., Garcia L., Soares C.. Meta-features for meta-learning. *Knowl. Based Syst.*, 2022, Vol. 240, pp. 108-101. https://doi.org/10.1016/j.knosys.2021.108101.

78. Imani M., Beikmohammadi A. Comprehensive Analysis of Random Forest and XGBoost Performance with SMOTE, ADASYN, and GNUS Under Varying Imbalance Levels. *Technologies*, 2025, Vol. 13(3), 88. https://doi.org/10.3390/technologies13030088

79. González-Estrada E., Cosmes W. Shapiro–Wilk test for skew normal distributions based on data transformations. *Journal of Statistical Computation and Simulation*, 2019, Vol. 89, pp. 3258 - 3272. https://doi.org/10.1080/00949655.2019.1658763.

80. Steven T. Garren, Grace H. Davenport.Using Kurtosis for Selecting One-Sample T-Test or Wilcoxon Signed-Rank Test. *Current Journal of Applied Science and Technology* . 2022, Vol. 41 (18), pp.46–55. https://doi.org/10.9734/cjast/2022/v41i1831737.

81. Kaggle. California Housing Prices https://surl.li/tzjldv

82. Bank Marketing. https://archive.ics.uci.edu/dataset/222/bank+marketing

83. Hillegersberg J., Zuidwijk Rob. Supporting return flows in the supply chain. Commun. *ACM.* 2001. Vol. 44, 6, pp. 74–79. https://doi.org/10.1145/376134.376172

84. Acharya A., Dhillon I. S. Sanghavi S. Geometric Median (GM) Matching for Robust Data Pruning. *arXiv preprint arXiv:2406.17188.* 2024. https://doi.org/10.48550/arXiv.2406.17188

Додаток А. Програмний код

```python
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from typing import Tuple, Dict, Any
import warnings
warnings.filterwarnings('ignore')
class DatasetManager:  1 usage
    AVAILABLE_DATASETS = {
        'california_housing': 'regression',
        'bank_marketing': 'classification'
    }
    def __init__(self, random_state: int = 42):
        self.random_state = random_state
        self.datasets_cache = {}
    def load_dataset(  2 usages
            self,
            dataset_name: str,
            test_size: float = 0.2,
            val_size: float = 0.2
    ) -> Dict[str, Any]:
        if dataset_name not in self.AVAILABLE_DATASETS:
            raise ValueError(
                f"Датасет '{dataset_name}' не підтримується. "
                f"Доступні: {list(self.AVAILABLE_DATASETS.keys())}"
            )

        result = {
            'name': dataset_name,
            'task_type': data_dict['task_type'],
            'feature_names': data_dict['feature_names'],
            'target_name': data_dict['target_name'],
            'description': data_dict['description'],
            # Дані
            'X_train': X_train,
            'X_val': X_val,
            'X_test': X_test,
            'y_train': y_train,
            'y_val': y_val,
            'y_test': y_test,
            # Статистика
            'n_samples': len(X),
            'n_features': X.shape[1],
            'n_train': len(X_train),
            'n_val': len(X_val),
            'n_test': len(X_test),
        }
        if data_dict['task_type'] == 'classification':
            result['n_classes'] = len(np.unique(y))
            result['class_distribution'] = dict(pd.Series(y).value_counts())
```

Рисунок А.1 – Модуль datasets.py (фрагмент коду)

```python
class OptunaOptimizer:  9 usages
    def __init__(
            self,
            task_type: str = 'regression',
            n_trials: int = 50,
            random_state: int = 42
    ):
        self.task_type = task_type
        self.n_trials = n_trials
        self.random_state = random_state
        self.best_params = None
        self.best_score = None
        self.training_time = 0
        self.optimization_history = []
    def optimize(  7 usages (2 dynamic)
            self,
            X_train: np.ndarray,
            y_train: np.ndarray,
            X_val: np.ndarray,
            y_val: np.ndarray
    ) -> Dict[str, Any]:
        print(f"\n Optuna Optimizer - {self.n_trials} trials")
        print("-" * 60)

        start_time = time.time()
        def objective(trial):
            params = {
                'n_estimators': trial.suggest_int('n_estimators', 50, 300),
                'max_depth': trial.suggest_int('max_depth', 3, 20),
                'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
                'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
                'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', None])
            }
            if self.task_type == 'regression':
                model = Pipeline([
                    ('scaler', StandardScaler()),
                    ('regressor', RandomForestRegressor(
                        **params,
                        random_state=self.random_state,
                        n_jobs=-1
                    ))
                ])
                model.fit(X_train, y_train)
                y_pred = model.predict(X_val)
                score = np.sqrt(mean_squared_error(y_val, y_pred))
                return score
```

Рисунок А.2 – Модуль optimizers.py (фрагмент)

```
class StatisticalAnalyzer:
    def __init__(self, alpha: float = 0.05):
        self.alpha = alpha
    def check_normality(self, data: np.ndarray, method: str = 'shapiro') -> Tuple[float,
        if method == 'shapiro':
            statistic, p_value = stats.shapiro(data)
            test_name = "Shapiro-Wilk"
        elif method == 'kstest':
            statistic, p_value = stats.kstest(data, 'norm')
            test_name = "Kolmogorov-Smirnov"
        else:
            raise ValueError(f"Невідомий метод: {method}")
        is_normal = p_value > self.alpha
        print(f"\n Тест нормальності ({test_name}):")
        print(f"   Статистика: {statistic:.4f}")
        print(f"   P-value: {p_value:.4f}")
        print(f"   Висновок: {'Нормальний розподіл' if is_normal else 'Не нормальний розп
        return statistic, p_value, is_normal
    def paired_ttest(  1 usage
            self,
            sample1: np.ndarray,
            sample2: np.ndarray,
            method1_name: str = 'Method 1',
            method2_name: str = 'Method 2'
    ) -> Dict[str, Any]:
        if len(sample1) != len(sample2):
            raise ValueError("Вибірки повинні мати однаковий розмір")

    def wilcoxon_test(  1 usage
            self,
            sample1: np.ndarray,
            sample2: np.ndarray,
            method1_name: str = 'Method 1',
            method2_name: str = 'Method 2'
    ) -> Dict[str, Any]:
        statistic, p_value = stats.wilcoxon(sample1, sample2)
        diff = sample1 - sample2
        median_diff = np.median(diff)
        is_significant = p_value < self.alpha
        results = {
            'test': 'Wilcoxon signed-rank test',
            'method1': method1_name,
            'method2': method2_name,
            'statistic': statistic,
            'p_value': p_value,
            'median_diff': median_diff,
            'is_significant': is_significant,
            'alpha': self.alpha
        }
        self._print_comparison_results(results)
        return results
```

Рисунок А.3 – Модуль statistics.py

```
    # ROC-AUC (потрібні ймовірності)
    if y_pred_proba is not None:
        try:
            if len(y_pred_proba.shape) > 1:
                # Для бінарної класифікації беремо ймовірність позитивного класу
                metrics['ROC_AUC'] = roc_auc_score(y_true, y_pred_proba[:, 1])
            else:
                metrics['ROC_AUC'] = roc_auc_score(y_true, y_pred_proba)
        except Exception as e:
            metrics['ROC_AUC'] = None
            print(f"Не вдалось обчислити ROC-AUC: {e}")

    # Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    metrics['Confusion_Matrix'] = cm

    # Специфічність (Specificity) = TN / (TN + FP)
    if cm.shape == (2, 2):
        tn, fp, fn, tp = cm.ravel()
        metrics['Specificity'] = tn / (tn + fp) if (tn + fp) > 0 else 0
        metrics['True_Negatives'] = int(tn)
        metrics['False_Positives'] = int(fp)
        metrics['False_Negatives'] = int(fn)
        metrics['True_Positives'] = int(tp)
```

```
    # Тест 3: Порівняння моделей
    print("\n\nТЕСТ ПОРІВНЯННЯ МОДЕЛЕЙ")
    print("=" * 70)

    test_results = [
        {'method': 'baseline', 'best_score': 0.65, 'training_time': 5.2, 'n_trials': 1},
        {'method': 'optuna', 'best_score': 0.58, 'training_time': 125.7, 'n_trials': 50},
        {'method': 'adaptive', 'best_score': 0.59, 'training_time': 78.3, 'n_trials': 32}
    ]

    comparison_df = evaluator_reg.compare_models(test_results)
    print("\nПорівняльна таблиця:")
    print(comparison_df.to_string(index=False))

    winner_method, winner_score = evaluator_reg.get_winner(test_results)
    print(f"\nПереможець: {winner_method.upper()} (score: {winner_score:.4f})")

    # Тест 4: Покращення
    print("\n\nТЕСТ ОБЧИСЛЕННЯ ПОКРАЩЕННЯ")
    print("=" * 70)

    improvement = calculate_improvement( baseline_score: 0.65,  new_score: 0.58, task_type='regression')
    print(f"Покращення відносно baseline: {improvement:.2f}%")

    print("\n" + "=" * 70)
    print("ВСІ ТЕСТИ ПРОЙДЕНО УСПІШНО!")
    print("=" * 70)
```

Рисунок А.4 – Модуль evaluator.py

```
def plot_distribution_boxplot(
        self,
        multiple_runs: Dict[str, List[float]],
        metric_name: str,
        dataset_name: str,
        save_name: str = None
):
    fig, ax = plt.subplots(figsize=(10, 7))
    methods = list(multiple_runs.keys())
    data = [multiple_runs[m] for m in methods]
    colors = [self.colors.get(m.lower(), '#95a5a6') for m in methods]

    bp = ax.boxplot(data, labels=methods, patch_artist=True,
                    showmeans=True, meanprops=dict(marker='o',
                                                   markerfacecolor='white', markeredgecolor='blue'))
    for patch, color in zip(bp['boxes'], colors):
        patch.set_facecolor(color)
        patch.set_alpha(0.6)
    for i, (d, c) in enumerate(zip(data, colors)):
        x = np.random.normal(i + 1, 0.04, len(d))
        ax.scatter(x, d, alpha=0.6, color=c, s=50, edgecolor='white')
    for i, (m, d) in enumerate(zip(methods, data)):
        mean_val = np.mean(d)
        std_val = np.std(d)
        ax.text(i + 1, ax.get_ylim()[1], f'μ={mean_val:.4f}\nσ={std_val:.4f}',
                ha='center', va='bottom', fontsize=10,
                bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))
    ax.set_ylabel(metric_name, fontsize=12, fontweight='bold')
    ax.set_title(f'Розподіл {metric_name} за 10 запусків\n'
                 f'(червона лінія = медіана, синя пунктир = середнє)',
                 fontsize=14, fontweight='bold')
    ax.grid(True, alpha=0.3, axis='y')
    plt.tight_layout()
    if save_name:
        plt.savefig(f"{self.save_path}{save_name}",
                    bbox_inches='tight', dpi=300)
        print(f" Збережено: {save_name}")
    plt.show()
    return fig
```

Рисунок А.5 – Модуль visualization.py

Додаток Б. Копії документів про апробацію результатів роботи

Лобода Юлія Геннадіївна
Національний університет «Одеська юридична академія»,
доцент кафедри інформаційних технологій,
кандидат педагогічних наук, доцент
Дерев'ягін Владислав Сергійович
Національний університет «Одеська юридична академія»,
студент 2-го курсу магістратури факультету кібербезпеки та інформаційних технологій

АРХІТЕКТУРА ТА ОСНОВНІ КОМПОНЕНТИ AUTOML-СИСТЕМ

Автоматизоване машинне навчання (AutoML) один із ключових напрямів розвитку сучасної аналітики даних, оскільки забезпечує можливість створення моделей машинного навчання без необхідності глибокої експертної участі. Класичний процес розроблення моделей включає низку послідовних етапів від збору та препроцесингу даних до вибору моделі та оптимізації її гіперпараметрів, які потребують значних часових і інтелектуальних ресурсів [1].

AutoML-системи дозволяють навчати високоякісні моделі автоматично, використовуючи комплекс алгоритмів для формування пайплайнів, пошуку моделей та оптимізації параметрів. В умовах зростання обсягів даних, збільшення складності моделей та потреби у швидкому отриманні результатів AutoML стає важливим інструментом як для наукових досліджень, так і для промислових застосувань.

Метою дослідження є аналіз архітектурних принципів сучасних AutoML-систем, структурних компонентів ML-пайплайна та методів оптимізації гіперпараметрів, які визначають продуктивність і якість автоматизованого моделювання.

AutoML-системи грунтуються на модульній архітектурі, у якій кожен компонент відповідає за певний етап підготовки та побудови моделі.

Структура AutoML включає такі ключові елементи:

– модуль препроцесингу даних, що виконує автоматичну обробку пропусків, аномалій, масштабування числових ознак та кодування категорій;

– модуль інженерії та відбору ознак, який підтримує автоматичну генерацію ознак та інтелектуальний відбір найбільш інформативних характеристик;

– модуль вибору моделі, що тестує широкий спектр алгоритмів від лінійних моделей до ансамблевих методів і нейронних мереж;

– модуль оптимізації гіперпараметрів як центральний компонент архітектури;

– модуль ансамблювання моделей для підвищення стабільності та точності;

– модуль оцінювання та вибору найкращої моделі [2].

Для системного описання AutoML-систем використовують формальні поняття: оператор пайплайна як окрема операція над даними; пайплайн як впорядкована послідовність операторів; простір пошуку як множина усіх можливих пайплайнів; оптимізаційна задача як вибір оптимального пайплайна, що мінімізує функцію втрат на валідаційних даних. Розмір простору пошуку може сягати мільйонів варіантів навіть для простих задач, що робить вибір ефективного методу оптимізації критично важливим.

Класичні методи оптимізації включають Grid Search – повний перебір дискретних конфігурацій з експоненційною складністю, та Random Search – випадкове дослідження простору параметрів, яке часто виявляється ефективнішим у високорозмірних просторах [3]. Просунуті методи представлені байєсовою оптимізацією, яка будує модель функції втрат; bandit-методами, що забезпечують оптимальний розподіл обчислювальних ресурсів через Successive Halving, Hyperband та BOHB; еволюційними алгоритмами, ефективними в складних і нерегулярних просторах [4,5]. Спеціалізовані підходи включають Neural Architecture Search для автоматичного пошуку архітектур нейронних мереж та Meta-Learning для використання історичних знань (warm-start, transfer learning та learning curves prediction).

Проведений аналіз методів показує, що Grid Search забезпечує гарантоване знаходження оптимуму в обраному просторі, але має низьку швидкість збіжності та погану масштабованість. Random Search демонструє кращу масштабованість при середній швидкості збіжності. Байєсова оптимізація досягає високої швидкості збіжності та відмінної якості рішень, але має середню масштабованість та високу складність реалізації. Hyperband та BOHB поєднують дуже високу швидкість збіжності з відмінною масштабованістю, що робить їх найперспективнішими для сучасних AutoML-систем.

Практичні реалізації AutoML-систем включають Auto-sklearn, який використовує SMAC і мета-навчання для warm-start; H2O AutoML – платформу з підтримкою розподіленого навчання; TPOT, що використовує генетичне програмування; а також хмарні рішення Google Cloud AutoML та Azure AutoML, які застосовують NAS для побудови глибоких моделей [6].

В результате аналізу архітектури AutoML-систем та їх оптимізаційних механізмів встановлено, що ефективність AutoML визначається узгодженою взаємодією модулів препроцесингу, інженерії ознак, вибору моделей, оптимізації гіперпараметрів та оцінювання. AutoML-системи поєднують різні оптимізаційні техніки, забезпечуючи баланс між точністю, швидкістю та обчислювальною ефективністю. Найперспективнішими є гібридні методи, що комбінують переваги різних підходів, зокрема BOHB як комбінація байєсової оптимізації та Hyperband. Огляд сучасних платформ підтверджує ефективність модульного проєктування AutoML-систем і демонструє різноманітність підходів до реалізації архітектури. Отримані результати можуть бути використані як теоретична основа для подальших досліджень AutoML-пайплайнів, створення експериментальних оптимізаційних систем та побудови автоматизованих рішень у задачах Data Science.

Список використаної літератури:

1. Del Valle A.M., Mantovani R.G. & Cerri R. A systematic literature review on AutoML for multi-target learning tasks. *Artif Intell Rev.* 2023. 56 (Suppl 2), pp. 2013–2052. https://doi.org/10.1007/s10462-023-10569-2.

2. Karl F., Thomas J., Elstner J., Gross R., Bischl B. Automated Machine Learning. *Unlocking Artificial Intelligence*. Springer, Cham. 2024, pp. 3–25. https://doi.org/10.1007/978-3-031-64832-8_1.

3. Liashchynskyi P., Liashchynskyi P. Grid search, random search, genetic algorithm: a big comparison for NAS. *arXiv preprint*. 2019. https://doi.org/10.48550/arXiv.1912.06059.

4. Dôres S.C.N., Soares C. Bandit-Based Automated Machine Learning. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, Sao Paulo, Brazil, 2018, pp. 121-126, DOI: 10.1109/BRACIS.2018.00029

5. Falkner S. et al. BOHB: Robust and efficient hyperparameter optimization at scale. *International conference on machine learning*. PMLR, 2018. P. 1437–1446. https://doi.org/10.48550/arXiv.1807.01774

6. SMAC: Sequential Model-based Algorithm Configuration. URL: https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/

Ключові слова: автоматизоване машинне навчання, машинне навчання, гіперпараметри, оптимізація, архітектура систем, пайплайн.

Keywords: AutoML, machine learning, hyperparameters, optimization, system architecture, pipeline.