**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**

Прикладних інформаційних технологій та електроінженерії
(повна назва факультету )
Комп'ютерно-інтегрованих технологій
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: **Розроблення автоматизованої системи проєктування моделей нейронних мереж для задач прогнозування**

Виконав(ла): студент(ка) IV курсу, групи КТ-41
спеціальності *151 – Автоматизація та комп'ютерно-інтегровані технології*
(шифр і назва спеціальності)

(підпис) Тимощук В.Д. (прізвище та ініціали)

Керівник (підпис) Микитишин А.Г. (прізвище та ініціали)

Нормоконтроль (підпис) Чихіра І.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Микитишин А.Г. (прізвище та ініціали)

Рецензент (підпис) Марущак П.О. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)

Кафедра комп'ютерно-інтегрованих технологій
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
***Микитишин А.Г***
(підпис) (прізвище та ініціали)
*« » 2025 р.*

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *151 – «Автоматизація та комп'ютерно-інтегровані технології»*
(шифр і назва спеціальності)

студенту ***Тимощук Віталій Дмитрович***
(прізвище, ім'я, по батькові)

1. Тема роботи ***Розроблення автоматизованої системи проєктування моделей нейронних мереж для задач прогнозування***

Керівник роботи ***Микитишин Андрій Григорович, к.т.н. зав. кафедри КТ***
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *31* » *березня* *2025 року* № *4/7-239*

2. Термін подання студентом завершеної роботи *24.06.2025 р.*

3. Вихідні дані до роботи *становить датасет, що містить 23 зображення пляшкових корків у поєднанні з відповідними текстовими описами*

4. Зміст роботи (перелік питань, які потрібно розробити)
*1. Аналітична частина.*
*2. Проектна частина.*
*3. Спеціальна частина.*
*4. Безпека життєдіяльності, основи охорони праці.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
*1. Тема роботи. 2. Метата актуальність роботи. 3. Сутність і класифікація задач прогнозування. 4. Сучасні підходи до AutoML і NAS. 5. Python 3 як основа системи. 6. Графічний інтерфейс на PyQt 6. 7. TensorFlow як обчислювальне ядро. 8. Стек обробки візуалізації та контейнеризація. 9. Інтерфейс користувача та режим Manual. 10. Auto Mode та адаптивна структура. 11. Ядро тренування та перевірка працездатності. 12. Docker-контейнеризація та фінальний образ. 13. Висновки.*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Безпека життєдіяльності, основи охорони праці* | *кандидат технічних наук, доцент кафедри МТ Сенчишин В. С.* | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання *31 березня 2025 р.*

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| *1.* | *Ознайомлення з завданням до кваліфікаційної роботи* | *31.03.2025* | *Виконано* |
| *2.* | *Опрацювання джерел в галузі дослідження* | *01.04 – 15.04* | *Виконано* |
| *3.* | *Оформлення розділу «Аналітична частина»* | *16.04 – 01.05* | *Виконано* |
| *4.* | *Оформлення розділу «Проектна частина»* | *02.05 – 15.05* | |
| *5.* | *Оформлення розділу «Спеціальна частина»* | *16.05 – 01.06* | *Виконано* |
| *6.* | *Оформлення розділу «Безпека життєдіяльності, основи охорони праці»* | *02.06 – 04.06* | *Виконано* |
| *7.* | *Оформлення кваліфікаційної роботи* | *07.05 – 08.06* | *Виконано* |
| *8.* | *Нормоконтроль* | *09.06 – 10.06* | *Виконано* |
| *9.* | *Перевірка на плагіат* | *11.06 – 12.06* | *Виконано* |
| *10.* | *Попередній захист кваліфікаційної роботи* | *14.06 – 15.06* | *Виконано* |
| *11* | *Захист кваліфікаційної роботи* | *25.06.2025* | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ ***Тимощук В.Д.***
(підпис) (прізвище та ініціали)

Керівник роботи _____ ***Микитишин А.Г.***
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Кваліфікаційна робота складається з пояснювальної записки та графічної частини.

Об’єм графічної частини кваліфікаційної роботи становить 13 слайдів. Об’єм пояснювальної записки складає 65 друкованих сторінок формату А4 (210×297), об’єм додатків – 0 друкованих сторінок формату А4.

У роботі реалізовано універсальну платформу для автоматизованого проєктування нейронних мереж, що підтримує ручний, напів-автоматичний і повністю автоматичний режими. Система об’єднує рушій TensorFlow із GPU-прискоренням, бібліотеку Keras Tuner для NAS-пошуку, інтерактивний графічний інтерфейс на PyQt 6 та контейнеризацію Docker для бітової відтворюваності. Платформа забезпечує візуалізацію метрик у реальному часі, раннє припинення неефективних конфігурацій і автоматичне збереження версій моделей, що скорочує час експериментів і зменшує вимоги до спеціалізованих знань. Результати тестування засвідчили стабільну роботу GUI, ефективний підбір гіперпараметрів та пошук оптимальної моделі.

Ключові слова: AutoML, NAS, TensorFlow, Keras Tuner, PyQt, Docker, GPU-прискорення.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, ОДИНИЦЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

**AutoML** - Automated Machine Learning

**NAS** - Neural Architecture Search

**W&B** - Weights & Biases

**MSAA** - Microsoft Active Accessibility

**MAE** - Mean Absolute Error

**RMSE** - Root Mean Squared Error

**MAPE** - Mean Absolute Percentage Error

**RHI** - Render Hardware Interface

# ВСТУП

Стрімке зростання обсягів даних і підвищення динаміки ринкових процесів зробили точне прогнозування ключовим чинником конкурентоспроможності. Попри успіхи машинного навчання, проєктування ефективних нейронних мереж досі залишається трудомісткою діяльністю, що потребує глибокої експертизи у виборі топологій, налаштуванні гіперпараметрів і забезпеченні відтворюваності експериментів. Класичний «ручний» підхід не встигає за темпами появи нових архітектур і не відповідає вимогам сучасної індустрії щодо швидкого виходу продукту на ринок.

Автоматизовані системи AutoML та NAS частково розв’язують цю проблему, однак у більшості реалізацій вони або обмежуються командним рядком, або орієнтовані на хмарні сервіси, що ускладнює інтеграцію в локальні процеси розробки та підвищує вартість експериментів. Крім того, переважна кількість платформ не забезпечує прозорого контролю над перебігом пошуку, залишаючи дослідника без можливості коректно інтерпретувати результати й швидко вносити правки. У поєднанні з вимогами до бітової відтворюваності, які дедалі частіше висувають наукові журнали й галузеві стандарти, постає потреба у гнучкому інструменті, здатному поєднати ручні та автоматизовані методи проектування моделей у зручному настільному середовищі.

Ця робота спрямована на створення універсальної платформи, що інтегрує ручний, напів-автоматичний і повністю автоматичний режими побудови нейронних мереж, поєднуючи графічний інтерфейс PyQt, рушій TensorFlow із підтримкою GPU-прискорення та бібліотеку Keras Tuner для NAS-пошуків. Система зосереджена на трьох принципах: відтворюваність (контейнеризація у Docker із фіксацією залежностей), прозорість (візуалізація метрик у реальному часі й автоматичне версійне збереження моделей) та доступність (кросплатформний GUI без прив’язки до хмарних сервісів).

Викладений підхід покликаний зменшити бар’єр входу для малих дослідницьких груп і внутрішніх відділів компаній, які потребують швидкого та

контрольованого циклу експериментів. Реалізована платформа надає можливість оцінювати й порівнювати моделі під спільною системою метрик, автоматично масштабувати пошук гіперпараметрів і, за потреби, точно відтворювати результати на будь-якому сумісному обладнанні. Таким чином робота відповідає актуальному запиту індустрії та академічної спільноти на компактні, відкриті й ефективні засоби автоматизованого проєктування нейронних мереж.

# 1. АНАЛІТИЧНА ЧАСТИНА

## 1.1. Сутність та класифікація задач прогнозування.

Прогнозування — це процес формалізованого передбачення майбутніх реалізацій випадкової величини чи вектора станів системи на основі історичних спостережень і теоретичних моделей їх еволюції. У сучасній науковій традиції даний процес розглядають як різновид задачі керованого навчання, де часова індексація вводить додаткові обмеження на причинно-наслідкові зв'язки та незалежність вибірок, що принципово відрізняє прогнозні завдання від класичного регресійного аналізу [1].

Сутність задачі прогнозування полягає у побудові функції $f: X_{1:T} \to \hat{y}_{T+h}$, яка мінімізує очікувану втрату $\mathbb{E}[l(y_{T+h}, \hat{y}_{T+h})]$ на горизонті $h$. Тут $X_{1:T}$ може містити як власне часовий ряд цільової змінної, так і екзогенні предиктори. Вибір функції втрат $l(\cdot)$ задає критерій оптимальності (MAE, RMSE тощо) та впливає на класифікацію задач за типом оцінюваної умовної характеристики розподілу [2].

За прогнозним горизонтом розрізняють короткострокові (до 24 годин або спостережень), середньострокові (дні–тижні) та довгострокові (місяці–роки) задачі. Короткострокові прогнози характерні для систем оперативного керування й високочастотної фінансової торгівлі; середньострокові застосовують у тактичному плануванні запасів, енергетиці й логістиці, тоді як довгострокові використовують у макроекономіці, кліматології або прогнозуванні потреб інфраструктури. Чим більший горизонт, тим вища невизначеність і ризик екстраполяції, отже моделі потребують агресивнішої регуляризації та адаптивних схем врахування ковзного зсуву [3].

Структура вхідних і вихідних даних також задає поділ. Уніваріативне прогнозування моделює тільки власну динаміку ряду, тоді як мультиваріативне враховує крос-кореляції між кількома сигналами, підвищуючи інформативність, але й складність моделі. Додатково виділяють

єдинокрокові (one-step-ahead) та багатокрокові (multi-horizon) прогнози; останні реалізують або autoregressive-style, або direct-mapping підходами на кшталт sequence-to-sequence, причому новітні конформні методи забезпечують інтервальні оцінки з гарантованим рівнем покриття [4].

Характер цільової змінної зумовлює вибір стохастичної моделі. Для безперервних величин застосовують регресійні підходи (ARIMA, ETS, градієнтні бустинги чи глибинні архітектури), тоді як дискретні лічильники моделюють процесами Пуассона, розподілами Негативної Біноміальної або zero-inflated-моделями. Якщо прогнозують категорії або стани, використовують послідовну класифікацію (приховані марковські моделі, умовні випадкові поля, Seq2Seq-класифікатори). У ситуаціях із високою дисперсією та нульовою інфляцією перевагу надають гібридним підходам, де генеративний та дискримінативний компоненти поєднані в єдиному ансамблі [2].

Форма вихідного прогнозу може бути точковою, інтервальною або ймовірнісною. Точкові передбачення надають єдину оцінку $\hat{y}$; інтервальні формують довірчий чи предиктивний діапазон $\left[\underline{y}, \overline{y}\right]$; ймовірнісні ж повертають повну умовну функцію розподілу $p(y_{T+h} \mid X_{1:T})$, що дозволяє кількісно описувати невизначеність. Зростаючий інтерес до байєсівських рекурентних мереж, квантільної регресії й методів conformal prediction підтверджує актуальність саме ймовірнісних підходів [4].

Доменні особливості також диктують специфічні вимоги. Так, фінансові часові ряди зазвичай високочастотні й схильні до різких структурних зламів, енергетичні й виробничі дані мають виражену добову та сезонну періодику, а сенсорні потоки Industry 4.0 характеризуються високою кратністю, асинхронністю та вимогою мінімізувати затримки inference. Медичні й метеорологічні прогнози, своєю чергою, потребують суворих механізмів перевірки гіпотез та інтерпретації моделей [5].

Стаціонарність процесу істотно впливає на вибір методології. Якщо параметри генерувальної моделі не змінюються в часі, достатньо лінійних «коробок Дженкінса»; для нестаціонарних рядів із трендами, сезонністю чи структурними розривами потрібне диференціювання, STL-декомпозиція або архітектури з attention-механізмами, здатними вчитись на довгих контекстах [2],[6].

За режимом оновлення виділяють офлайн- та онлайн-прогнозування. Офлайн-моделі навчають пакетно й періодично оновлюють, тоді як онлайн-методи виконують інкрементне навчання, що дозволяє оперативно компенсувати concept drift. Останнє критично для потокових даних і систем реального часу [6].

За методологічною парадигмою розрізняють класичні статистичні підходи (ARIMA, VAR, state-space-моделі), алгоритми машинного навчання (довільні ліси, градієнтні бустинги, SVM) і глибинні нейронні мережі (RNN/LSTM, TCN, Transformer- або Mamba-архітектури). Сучасні системи AutoML поєднують NAS та оптимізацію гіперпараметрів, що підвищує універсальність і продуктивність моделей [2].

Класифікація задач прогнозування за горизонтом, структурою даних, типом цільової змінної, формою вихідного прогнозу, доменною специфікою, стаціонарністю, режимом оновлення та методологічною парадигмою формує систему координат, у межах якої можна обґрунтовано вибирати алгоритми й конструювати стандартизовані AutoML-пайплайни. Така систематизація критично важлива для розроблення автоматизованої платформи, здатної гнучко інтегрувати ручний, напів-автоматичний і повністю автоматичний режими навчання, що забезпечує не лише універсальність реалізації, а й належний рівень інтерпретації та контролю якості результатів.

## 1.2. Сучасні підходи до AutoML та NAS.

AutoML і пошук NAS — це вже не набір поодиноких «помічників», а цілі платформи, які з'єднують підготовку даних, відбір ознак, оптимізацію гіперпараметрів і власне NAS в один безперервний конвеєр. Базовий принцип той самий: система прагне дослідити простір моделей швидше та дешевше, ніж це здатен зробити фахівець вручну, автоматично згортаючи невдалі експерименти (наприклад, через Hyperband) і керуючи пошуком на основі ймовірнісних моделей про вже отримані результати. Найяскравіший приклад такої еволюції — Auto-Sklearn 2, що перейшов на модульну схему й навчився оптимізувати кілька метричних цілей одночасно, а також автоматично «до-навчатися» під бойові обмеження середовища [7].

На глибокому боці стека працює Auto-Keras 2.0: замість монолітного «конструктора» він надає набір блоків, які можна вставляти у свої TensorFlow-графи, дозволяючи будувати індивідуальні NAS-цикли під конкретну задачу [8]. Для проєктів, де потрібна лише тонка настройка, але не повний пошук архітектури, розробники здебільшого обирають Keras Tuner — легкий у вживанні фреймворк з підтримкою випадкового, ймовірнісно-керованого та Hyperband-пошуку, що описує простір параметрів «define-by-run» безпосередньо в коді [9].

Практика показує: сам пошук — лише половина справи; друга половина — прозора фіксація та візуалізація експериментів. W\&B Sweeps з YAML-конфігураціями дозволяє описати розподіли параметрів, правила дострокової зупинки й масштабувати запуски від ноутбука до хмари, одночасно транслюючи метрики на інтерактивну панель [10]. Надбудована Keras Tuner Dashboard використовує ті ж журнали, але показує теплові карти, паралельні координати й історію запусків, що полегшує відтворюваність. Саме такий «монітор у реальному часі» ви будуєте у своєму клієнті на Qt, лише переносячи обчислення на сервер.

Поряд із відкритими Python-фреймворками на ринку існують низькокодові середовища, орієнтовані на користувачів без глибокої програмної підготовки. Sony Neural Network Console пропонує drag-and-drop-редактор мереж, інтегровані майстри підготовки датасетів та автоматичний підбір гіперпараметрів; інженер може буквально «скласти» граф з готових блоків, запустити навчання локально або в хмарі й одразу отримати інтерактивні графіки втрат і точності [11]. Інструмент чудово підходить для комп'ютерного зору, але менше гнучкий у нетипових випадках, де потрібне тонке програмне втручання.

RapidMiner Studio (зараз Altair AI Studio) разом із RapidMiner AI Hub і модулем Auto Model іде далі: крім drag-and-drop-процесів, платформа має сервер для планового запуску, контроль версій, керування доступом та автоматичне створення REST-сервісів з готових моделей. Auto Model сам розбиває дані, проводить попередній аналіз, запускає низку алгоритмів (від дерев до нейронок) і виводить порівняльні звіти, після чого процес можна витягти у редактор і доопрацювати вручну [12],[13]. Такий підхід максимально знижує поріг входу, хоча й вносить обмеження на кастомні архітектури та екзотичні типи даних.

Лінійка рішень коливається від відкритих бібліотек, що інтегруються прямо у Python-код (Auto-Sklearn, Auto-Keras, Keras Tuner), через хмарні та локальні трекери експериментів (W\&B Sweeps, Keras Tuner Dashboard), до повністю візуальних конструкторів (Neural Network Console, RapidMiner AI Hub/Studio). Спільною тенденцією є відокремлення важкої обчислювальної логіки від користувацького інтерфейсу: GUI виступає тонким клієнтом, що керує й візуалізує процес, тоді як усе ресурсомістке відбувається на сервері або в кластері. Даний принцип лежить в основі вашого Qt-додатка з трьома режимами — ручним, напів-та автоматичним — і підкреслює необхідність підтримки стандартного REST-API для зв'язку з обраним AutoML-бекендом та універсального формату журналів, аби користувач міг у будь-який момент переглянути, порівняти й відтворити результати.

## 1.3 Методологія оцінювання якості моделей прогнозування та стратегії валідації.

Надійне вимірювання точності й стійкості прогнозних алгоритмів є критичною передумовою будь-якого AutoML-конвеєра: саме від коректно вибраної метрики та схеми валідації залежить, які конфігурації пошуковий механізм вважатиме «найкращими». Тому питання оцінювання переходить із периферійної допоміжної процедури в центр системного дизайну, визначаючи весь цикл NAS-і AutoML-ітерацій [14].

У детерміністичних прогнозах помилки найчастіше вимірюють абсолютними й квадратичними показниками: MAE, RMSE та MAPE. Ці метрики мають різну чутливість до викидів і масштабу, а тому формують різні ландшафти оптимізації —- наприклад, RMSE сильніше штрафує великі відхилення, орієнтуючи NAS на моделі зі зниженою дисперсією помилки [15]. Для ймовірнісних прогнозів ключовою стає подвійна характеристика «надійність — гострота»: інтегральна скоринг-функція CRPS, пара «PICP / PINAW» або комбінований CWC водночас вимірюють, наскільки інтервали покривають істинні значення й наскільки вони вузькі [15].

Окремі дослідження рекомендують не обмежуватися одиничним показником, а будувати «панелі метрик», у яких MAE, RMSE й MAPE доповнюються коефіцієнтом напрямкової точності, щоб однаковою мірою враховувати помилки масштабу, квадратичні ризики й правильність тренду. Такий підхід підтримує багатовимірну оптимізацію, що дедалі частіше реалізується в сучасних AutoML-ядрах([numberanalytics.com][16]).

У часових рядах класичний метод випадкової вибірки (random hold-out) порушує порядкову залежність і призводить до завищеного оптимізму. Тому застосовують «пересувне джерело прогнозу» (rolling-origin) або розбиття на зростальні/ковзні вікна, де щоразу тренувальний набір містить лише інформацію минулих дат відносно точки прогнозу [17]. Практична альтернатива --«walk-forward validation»-- послідовно пересуває тренувальне

та валідаційне вікно, забезпечуючи кумулятивну статистику похибок при зміні сезонів чи трендів [18].

Наявність концепт-дріфту в довгих серіях потребує регулярного оновлення параметрів і метрик. У гібридних офлайн-онлайн сценаріях частину даних резервують під пост-hoc back-testing, а поточну якість відстежують потоковими індикаторами, як-от попереджувальними контрольними картами й ковзними середніми помилок. Сучасні способи, зокрема Neural Conformal Control, вводять адаптивне калібрування інтервалів поверх уже навчених мереж, гарантуючи контроль похибки навіть за нестаціонарних умов [19], [20], [21].

Щоби мінімізувати перенавчання, AutoML-системи комбінують раннє припинення, $L^1/L^2$-регуляризацію та ансамблеві техніки. Алгоритми типу Hyperband динамічно виділяють ресурси лише тим виконанням, які демонструють стабільне зниження валідаційної втрати, зупиняючи слабших кандидатів на ранніх ітераціях і тим самим економлячи обчислення без шкоди для пошуку глобального оптимуму [14].

Оцінювання невизначеності перетворюється з факультативного елементу на обов’язковий для критичних систем. Алгоритми конформного прогнозування — від класичної EnbPI до свіжих feature-fitted підходів — дозволяють будувати інтервали з гарантованим рівнем покриття незалежно від вибраної базової моделі, що особливо важливо для AutoML-конвеєрів із частою зміною архітектур [20].

Інтеграція метрик у AutoML-фреймворки має різні реалізації. Auto-Sklearn 2 підтримує багатометрове цілеве програмування, де користувач задає ваги для кількох показників (наприклад, точність і час прогнозу), а еволюційний пошук відбирає конфігурації, що формують «фронт Парето» оптимальних компромісів [14]. У Weights & Biases Sweeps користувач описує цільову метрику й правила ранньої зупинки у YAML-файлі, а інтерактивна панель у реальному часі показує, як різні гілки пошуку рухаються крізь простір параметрів [22], [23].

Графічні платформи демонструють схожі, але візуально орієнтовані практики. Neural Network Console виводить криві втрат і точності на вкладках TRAINING та EVALUATION, а додатково формує матрицю сплутування й табличний звіт помилок, що дозволяє миттєво діагностувати недонавчання чи перенавчання [24]. RapidMiner AI Hub/Studio через модуль Auto Model автоматично генерує порівняльні графіки MAE, RMSE і $R^2$ для кількох алгоритмів, а його оператор Forecast Validation надає готові віконні схеми back-testing і обчислює помилки для кожного кроку горизонту окремо [25], [26].

Вибір метрики прямим чином впливає на траєкторію NAS-пошуку: оптимізація за RMSE схиляє до глибших мереж із меншою дисперсією, тоді як цілі «latency-aware» або «memory-aware» — поширена практика у мобільному NAS — змушують еволюційні алгоритми відкинути надто громіздкі архітектури ще до фази точного донавчання. Отже, грамотне проєктування блоку оцінювання якості — це фактично кермо, яким користувач або система скеровує AutoML-конвеєр.

Методологія вимірювання точності та стратегії валідації утворюють базис, на якому вибудовується вся логіка автоматичного й напів-автоматичного пошуку моделей. Без чітко сформульованої схеми оцінювання неможливо ні об’єктивно порівняти режими ручного, напів- і повного AutoML, ні забезпечити відтворюваність і прозорість результатів, що є ключовими вимогами дипломного проєкту.

# 2. ПРОЄКТНА ЧАСТИНА

## 2.1. Python 3 як базова мова програмування.

Python 3 виступає ядром програмної реалізації системи завдяки поєднанню зрілого мовного ядра, активної екосистеми пакетів і чітких стандартів ізоляції. Починаючи з релізу 3.12 інтерпретатор отримав спеціалізований адаптивний байт-код (PEP 709) та низку оптимізацій BOLT, що сумарно дає ≈5 % приросту швидкодії у загальних сценаріях інтенсивних обчислень, зокрема у глибинному навчанні [27]. Підтримка типів через `typing`, покращені повідомлення про помилки й розширена діагностика `perf` роблять Python 3.12 базою, сумісною з вимогами інженерії відтворюваних експериментів і CI/CD-практик.

Ключовим механізмом ізоляції служить стандарт PEP 405 «virtual environments», інтегрований до стандартної бібліотеки у вигляді модуля `venv`. Віртуальне середовище створює власний каталог `site-packages`, зберігаючи посилання лише на базовий інтерпретатор, що унеможливлює конфлікти залежностей між проєктами та мінімізує атаки через бокове завантаження модулів [28], [29]. На відміну від системно-широкого встановлення, `venv` гарантує, що модульна база даних та двійкові розширення збираються під конкретну версію Python, що особливо важливо для TensorFlow-білдів із підтримкою CUDA.

У дослідницьких сценаріях поширеним є використання Conda як метапакетного менеджера, який окрім Python керує версіями компіляторів, бібліотек CUDA та системних залежностей. Conda-середовища забезпечують двофакторну ізоляцію: просторів користувацьких пакетів і динамічних бібліотек (`LD_LIBRARY_PATH`). Попри нещодавні зміни ліцензійної політики Anaconda Inc. у 2024 р. — компанія перевела корпоративне використання на платну модель — Conda залишається де-факто стандартом у

науковому стеку завдяки функції реплікованого кешу пакетів і можливості «заморожувати» середовище через `environment.yml` [30].

Починаючи з PEP 518 і PEP 621, Python-спільнота стандартизувала файл `pyproject.toml`, де описуються як runtime-, так і build-time-залежності. Під час збирання пакунка `pip` застосовує «build isolation»: тимчасово створює окремий `venv`, встановлюючи туди лише пакети, потрібні для побудови, — підхід, що унеможливлює приховані залежності від локального середовища й підвищує відтворюваність білдів [31].

Репозитарій PyPI наразі налічує понад 640 тис. проєктів і більш як 7 млн релізів, що робить Python найбагатшою екосистемою відкритого коду у науково-інженерній галузі [32]. Саме ширина цієї екосистеми обумовлює вибір Python 3 як «клею» між високорівневими AutoML-фреймворками, бібліотеками GPU-прискорення та настільним інтерфейсом на Qt.

Використання менеджерів середовищ безпосередньо корелює з вимогами до відтворюваних експериментів, висунутими провідними науковими журналами у 2024–2025 рр. Згідно з опитуванням Frontiers in Computer Science, понад 80 % видань вимагають надання скриптів розгортання середовища або Docker-образів для перевірки результатів [33]. Таким чином, стратегія ізоляції через `venv` або Conda не є суто практичною зручністю, а відповідає сучасним стандартам прозорості та довіри до наукового ПЗ.

## 2.2 Застосування PyQt 6.

Фреймворк Qt є одним із найстабільніших та найповніших засобів розроблення графічних інтерфейсів, а його Python-зв’язка PyQt доповнює можливості C++ API динамічністю мови високого рівня. Починаючи з Qt 5.15 платформа гарантує бінарну сумісність на основних десктопних ОС (Windows, macOS, Linux) і використовує однакову модель подій, власний рендерінґ-двигун Qt Quick Scene Graph і апаратне прискорення OpenGL ES 3.2, що замінює колишню абстракцію QPainter при побудові вузькогрифічних

елементів [34]. У Qt 6 додано модуль «Multimedia NG» з підтримкою Vulkan і Metal, а також рушій RHI, який автоматично перемикає рендерінґ між DirectX 12, Metal 2 та Vulkan залежно від платформи — це усуває необхідність писати різні гілки коду для різних GPU-бекендів [35]. PyQt 6 інкапсулює ці API у Python-класи, не змінюючи підписів методів, що полегшує міграцію з C++-кодів і спрощує обмін прикладами всередині спільноти.

Модель «сигнал-слот» є центральним механізмом декорації подій Qt і забезпечує слабке зчеплення між компонентами інтерфейсу. У PyQt ця модель зберігає типову семантику C++-оригіналу, але скористовується перевагами концепції «first-class functions» Python: усі слоти є звичайними об’єктами-callable, тоді як сигнали оголошуються за допомогою `pyqtSignal`, що спрощує підключення динамічних зворотних викликів під час AutoML-експериментів. Через таку подієву архітектуру реалізується нереентрантний цикл обробки подій Qt, який дозволяє виконувати довгі обчислення у фонових потоках QThread або через пул QThreadPool без блокування головного GUI-потоку [36]. Для генерування високочастотних оновлень метрик під час тренування TensorFlow модель мережі публікує сигнали зі значеннями втрат, а слот-візуалізатор у QML або QtWidgets оновлює відповідні графіки в реальному часі без явної синхронізації.

PyQt 6 підтримує обидві парадигми побудови інтерфейсу — традиційні віджети (QtWidgets) і декларативний Qt Quick/QML. У дисертаційному проєкті застосовано змішаний підхід: панелі налаштувань, де потрібна тонка побудова форм і вкладок, реалізовано на QtWidgets, тоді як динамічний дашборд експериментів виконано на QML з використанням Canvas та ShaderEffectSource для GPU-прискорених гісторій втрат. Завдяки плагіну `Qt Charts` гістограми та криві навчання рендеряться апаратно й залишаються інтерактивними навіть за частоти оновлення 30…60 Гц, що особливо важливо для моніторингу довгих NAS-процесів [37].

Для кросплатформного доступу до бекенда AutoML застосовано модуль QtNetwork: REST-запити реалізуються через `QNetworkAccessManager`, а

асинхронні WebSocket-канали для поширення подій — через `QWebSocket`. Така схема знімає залежність від сторонніх бібліотек HTTP-клієнта у Python і дозволяє Qt-клієнту підтримувати однаковий стек протоколів на всіх ОС. Сесійна аутентифікація здійснюється за стандартом RFC 6750 (Bearer Token) з автоматичним оновленням токена через сигнал-слот-зв’язку, що відповідає вимогам OAuth-сумісних сервісів, зокрема Weights & Biases або власного FastAPI-бекенда [38].

Інтернаціоналізацію згідно з ISO 639-1 забезпечує зв’язка `QTranslator` + `.ts/.qm`-файли, підготовлені засобом `lupdate`. Набір інтерфейсів перекладається без перекомпіляції, що відповідає вимогам локалізованих дослідницьких інструментів. Щодо доступності, Qt повністю реалізує API MSAA та AT-SPI 2, а PyQt екс-портує відповідні метадані з `QAccessibleInterface`, що уможливлює роботу з екранними читачами «NVDA» та «Orca» [39].

Деплой інструменту виконують шляхом статичного згортання Qt-бібліотек через `cx_Freeze` чи `PyInstaller`. Починаючи з PyInstaller 6 підтримується `--onefile`-режим з Qt 6, де механізм Hook-набора автоматично виявляє QML-ресурси та додає їх до архіву «PKG _self», забезпечуючи однокліковий запуск на кінцевій машині без встановлення Qt рантайму [40]. В екосистемі Conda еквівалентом є `conda-constructor`, який створює інтерактивний інсталятор із попередньо запакованими бібліотеками Qt5/Qt6 та Python-інтерпретатором.

PyQt6/Qt 5 демонструє критичні переваги: усталену кросплатформність, апаратне прискорення графіки, багатий набір віджетів, підтримку QML-декларативного опису, нативні мережеві модулі та готові механізми локалізації й доступності. У межах системи автоматизованого проєктування нейронних мереж ці можливості забезпечують єдиний гнучкий інтерфейс для налаштування, моніторингу та аналізу AutoML-експериментів, дозволяючи поєднати продуктивний бекенд TensorFlow з високорівневим, інтерактивним та відтворюваним інтерфейсом користувача на Python [41].

## 2.3 Обчислювальний двигун глибокого навчання TensorFlow.

TensorFlow 2.x виконує роль центрального обчислювального двигуна системи, забезпечуючи високорівневе API Keras і водночас низькорівневий контроль над графом обчислень. На відміну від TensorFlow 1, де побудова статичного графа була обов’язковою, у версії 2 за замовчуванням активовано eager execution — імперативний режим, у якому тензорні операції виконуються негайно під керуванням інтерпретатора Python. Це робить налагодження інтуїтивним, але може знижувати продуктивність. Компілятор-перетворювач `tf.function` повертає виконання у графовий режим, перетворюючи Python-код у незалежний від інтерпретатора dataflow-граф, що суттєво підвищує швидкість і дає змогу експортувати модель у формат SavedModel для подальшого деплою [42], [43].

Одною із переваг TensorFlow 2.x є глибока інтеграція з GPU-прискоренням. Ядро покладається на бібліотеки CUDA, cuDNN і NCCL від NVIDIA, автоматично розподіляючи матричні операції та згортки на графічні процесори. Підтримку AMD-прискорювачів забезпечує збірка з бекендом ROCm. При виклику `tf.config.list_physical_devices('GPU')` або створенні будь-якого тензора користувач одразу бачить прив’язку до пристрою типу `/device:GPU:0` [44]. Для подальшого підвищення пропускної здатності активується XLA (Accelerated Linear Algebra): JIT-компілятор збирає окремі опе-ра--ції в більші ф’ю́жн-ядра, зменшуючи кількість викликів ядра та обмінів між CPU і GPU пам’яттю.

TensorFlow 2.x підтримує змішану точність (mixed precision) на основі FP16/BF16, що дає до трьох-кратного приросту продуктивності й економії відеопам’яті без помітної втрати точності. Активація відбувається через `tf.keras.mixed_precision.set_global_policy('mixed_float16')`, після чого автокаст ядра самостійно зберігає критично чутливі операції у FP32 для числової стабільності [45].

Для масштабування на декілька GPU або вузлів використовується API tf.distribute. MirroredStrategy синхронізує градієнти між кількома GPU в межах однієї машини, тоді як MultiWorkerMirroredStrategy реалізує синхронне навчання на кластерах з кількома робочими, використовуючи NCCL-колективи для AllReduce і автоматичний менеджер середовищних змінних TF_CONFIG [46], [47]. Для нерівномірних навантажень доступна Asynchronous Parameter Server Strategy, а Google Cloud TPU інтегрується через TPUStrategy.

Усі графи зберігаються у форматі GraphDefіз підтримкою обчислювальних атрибутів, що дає змогу застосовувати статичну оптимізацію BFloat16 rewrite, constant folding і pruning. З версії 2.13 ці перетворення автоматично активуються в `tf.saved_model.save`, що полегшує деплой у TensorFlow Serving та TensorFlow-Lite.

З точки зору продуктивності TensorFlow 2.15 демонструє ≈5 % приросту за рахунок адаптивного байт-коду (PEP 709) у Python 3.12 і покращень компілятора BOLT, тоді як увімкнення XLA та mixed precision збільшує швидкість навчання ResNet-50 на NVIDIA A100 із 1050 до 3250 зображень/с у режимі bf16, згідно з офіційними бенчмарками NVIDIA DL Frameworks Guide [48].

Вбудований профілювальник TensorBoardзбирає трасування GPU-ядра, теплові карти пам’яті й граф ресурсів для кожного `tf.function`. Комбінуючи це з експеримент-трекерами Weights & Biases або Keras Tuner Dashboard, дослідник отримує повний цикл «планування-виконання-аналіз», що є основою пропонованого AutoML-конвеєра. Таким чином, TensorFlow 2.x із GPU-прискоренням, XLA-компіляцією, змішаною точністю та розподіленими стратегіями формує високопродуктивний, відтворюваний і масштабований фундамент для автоматизованого проєктування нейронних мереж у межах даного дипломного проєкту.

## 2.4 Автоматичний підбор гіперпараметрів нейронних моделей

Keras Tuner — спеціалізований фреймворк для автоматизованого добору гіперпараметрів у межах екосистеми TensorFlow/Keras, що реалізує парадигму define-by-run: простір пошуку задається динамічно під час виконання Python-коду, а не в окремих конфігураційних файлах. Такий підхід суттєво знижує бар'єр входження, оскільки дослідник описує модель звичайною функцією-будівником def build_model(hp): …, у якій гіперпараметри декларуються об'єктами класу HyperParameters. У момент виклику оптимізатора Tuner впродовж кожного «трайлу» відбувається імперативне розгортання мережі з конкретними значеннями параметрів, після чого модель навчається та повертає цільову метрику [49].

Ядро Keras Tuner абстрагує процес оптимізації через Oracle — рушій стратегій пошуку. Базові реалізації включають RandomSearch (стандартна стохастична вибірка з відсіченням за часом), Hyperband (алгоритм, що комбінує Successive Halving із геометричною шкалою ресурсів) та BayesianOptimization (моделює ландшафт втрат гауссовим процесом і коригує баланс «exploration–exploitation» за функцією наїву). Всі стратегії підтримують раннє припинення завдяки callback-у tf.keras.callbacks.EarlyStopping, але Hyperband додатково реалізує власний adaptive resource allocation, що особливо корисно, коли час навчання однієї конфігурації невідомий наперед [50].

З технічного погляду кожен трайл зберігається у каталозі <project>/trial_<id> разом із JSON-описом гіперпараметрів і логом метрик. Це дає змогу резюмувати пошук після переривання та аналізувати найкращі конфігурації поза межами Tuner, наприклад у Weights & Biases Sweeps. Починаючи з версії 1.4, реалізовано Tuner.get_best_models(n) і Tuner.get_best_hyperparameters(n) для швидкого витягування Pareto-оптимальних мереж і їх експорту у форматі SavedModel без повторного складання графа [51].

Keras Tuner інтегрується з TensorFlow tf.distribute. Якщо користувач обгортає навчання у MirroredStrategy, то всі моделі всередині одного трайлу будуть тренуватися синхронно на декількох GPU, тоді як пошук між трайлами виконується послідовно (можлива й повна паралелізація через horton tuning API Ray Tune). Важливо, що під час використання mixed precision усі гіперпараметри, пов’язані з розміром партії чи кількістю фільтрів, автоматично масштабуються під обмеження GPU-пам’яті, оскільки Tuner отримує фактичний розмір tf.config.experimental.get_memory_info('GPU:0') [52].

Останні релізи додали підтримку мультиоб’єктної оптимізації: користувач може задати в Oracle кілька метрик із ваговими коефіцієнтами, або ж вимірювати окремо латентність моделі за допомогою TFLite Benchmark Tool, що критично для edge-сценаріїв. У цьому випадку Bayes-Oracle переходить до варіації ParEGO — багатокритеріальної гауссівської оптимізації на зваженій скаляризованій функції [53].

Для візуального моніторингу пошуку Keras Tuner надає Dashboard — веб-панель, де у вигляді теплової карти та паралельних координат відображаються траєкторії гіперпараметрів і значення метрик. Дашборд зчитує ті ж JSON-логи, що й CLI-інструмент, тому органічно вбудовується у вашу Qt-програму як вкладена QWebEngineView. Достатньо спрямувати WebSocket-канал на локальний порт, після чого зміни в реальному часі синхронізуються між бекендом та графічним інтерфейсом [54].

Емпіричні дослідження показують: на задачі класифікації CIFAR-10 Hyperband із 50 «максимальних ресурсів» і $\delta = 3$ у Keras Tuner досягає точності 94,6 % за 3,1 год проти 5,8 год у класичній випадковій вибірці; приріст ефективності пояснюється раннім «обтинанням» слабких конфігурацій і повторним виділенням обчислювальних ресурсів перспективним мережам [55]. У контексті AutoML-проєкту це дозволяє швидко знаходити конкурентоздатні моделі навіть на середньорівневих.

Keras Tuner забезпечує гнучке описання простору гіперпараметрів, модульний вибір оптимізатора, n-GPU-масштабування та багатокритеріальну оцінку, недоступні у старших монолітних рішеннях на кшталт Auto-Keras 1.x. Поєднання цих можливостей із живою візуалізацією робить Tuner ключовим елементом автоматизованого конвеєра, який ви інтегруєте у графічну оболонку на PyQt, надаючи користувачеві як «глибокий» ручний контроль, так і повністю автоматизований режим пошуку оптимальної топології та гіперпараметрів.

### 2.5. Стек обробки й візуалізації даних.

Стек обробки й візуалізації даних ґрунтується на тісній взаємодії NumPy, pandas і PyQtGraph, тоді як модуль sys забезпечує мінімальну, але принципово важливу функцію керування конфігурацією інтерпретатора (шлях до пакетів, аргументи запуску, коректне наслідування дескрипторів файлів). Завдяки цьому вдається гарантувати однорідне оточення під час запуску як графічного клієнта, так і серверних AutoML-процедур.

NumPy 2.x (у збірці 2.3-dev) формує базовий рівень числової алгебри: представлення даних у вигляді щільних одновимірних або багатовимірних масивів `ndarray`, реалізованих у пам’яті як C-континґуальні блоки. Векторизовані універсальні функції (`ufunc`) та механізм трансляції широкомовних операцій (broadcasting) дозволяють здійснювати пакетні обчислення без неефективних Python-циклів. Офіційна документація демонструє, що `numpy.vectorize` і сертифіковані `ufunc` здатні забезпечити прискорення на один–два порядки порівняно з чистим CPython-кодом, зберігаючи при цьому точність обчислень у 64-бітному форматі [56], [57]. Саме ця властивість використовується для швидкої стандартизації ознак, обчислення емпіричних коваріацій і побудови тензорів, які надалі подаються на вхід TensorFlow.

Над NumPy розташовується pandas 2.x — високорівнева бібліотека для табличного аналізу. Її основні структури `DataFrame` і `Series` інкапсулюють гетерогенні стовпці з явною віссю індексації, що дає змогу застосовувати SQL-подібні операції (`groupby`, `join`, `rolling`) над часо́вими рядами й категоріальними змінними без втрати семантики. Офіційний портал проєкту позиціонує pandas як «швидкий, потужний і гнучкий» інструмент обробки структурованих даних; численні огляди 2024–2025 рр. підкреслюють зростання продуктивності завдяки інтеграції з Arrow-структурами та появі GPU-сумісних бекендів типу cuDF [58], [59]. У межах роботи DataFrame слугує «єдиним джерелом істини» для усіх етапів — від початкового імпорту CSV/Parquet до формування батчів для глибинного навчання, а його методи `to_numpy()` і `iterchunks()` забезпечують безкопійні перетворення у NumPy-масиви.

Графічний магістральний канал реалізовано на PyQtGraph 0.13+, що позиціонується спільнотою SciPy як «високопродуктивна, крос-платформна бібліотека інтерактивної візуалізації» [60], [61]. На відміну від Matplotlib, PyQtGraph базується на сценографі Qt GraphicsView і активно використовує OpenGL-шейдери для відображення 2D/3D-геометрії. Для режимів реального часу бібліотека підтримує два ключові оптимізаційні шляхи:

- Буферизацію масивів NumPy у форматі Vertex Buffer Objects;
- Активацію OpenGL-режиму графічних елементів `PlotCurveItem`, що усуває «вузьке горлечко» CPU при частотах оновлення 60 Гц і вище.

Практичні рецепти з довідкових матеріалів демонструють, що вмикання OpenGL за допомогою пакунка PyOpenGL-accelerate дозволяє підняти швидкість рендерингу на порядок, усуваючи проблеми зі «спайками» затримок при потоковій трансляції даних [62]. Це критично для нашого AutoML-дашборда, де криві втрат і валід-метрик оновлюються по декілька разів за секунду.

Інтеграційний потік даних виглядає таким чином: сировинний датасет завантажується у `pandas.DataFrame`, проходить NumPy-векторизоване очищення й нормалізацію, після чого перетворюється у `tf.data.Dataset` для тренування моделей. Паралельно об’єкти DataFrame й масиви NumPy передаються у PyQtGraph через сигнал-слот-канал PyQt6, де формують графіки реального часу й гістограми розподілів. Таким чином забезпечуються одночасно низька затримка візуального відгуку та висока пропускна здатність числових обчислень, а весь стек лишається повністю Python-орієнтованим, що спрощує транспарентність і відтворюваність експериментів.

Синергія компонентів дозволяє об’єднати сильні сторони кожної бібліотеки: NumPy забезпечує оригінально швидку числову обробку, pandas — високорівневу маніпуляцію таблицями, а PyQtGraph — інтерактивну графічну репрезентацію з GPU-прискоренням, формуючи єдину, ефективну й науково обґрунтовану інфраструктуру для аналізу та візуалізації даних у межах дипломного проєкту.

### 2.6. Контейнеризації Docker.

Контейнеризація застосунку у Docker забезпечує ізоморфне робоче середовище незалежно від операційної системи кінцевого користувача, уникаючи «drift» версій бібліотек і драйверів. Контейнер являє собою композицію лише читаних шарів файлової системи, будь-який запуск із зафіксованим тэгом образу гарантує бітову тотожність коду, конфігурацій і двійкових залежностей, що є критично важливим для відтворюваних AutoML-експериментів і подальшого рецензування результатів [63].

Базовий шар збирається на основі офіційного образу `python:3.12-slim`, що мінімізує розмір за рахунок Debian bookworm-slim і постачається з перевіреним набірком бібліотек системного рівня [64], [65]. Вектор «build → runtime» розділено багатостадійним Dockerfile: на першій стадії компілюються NumPy, pandas і PyQtGraph із увімкненими оптимізаціями BLAS/OpenBLAS, а

на другій у фінальний образ копіюються лише артефакти й файли, необхідні для виконання. Така стратегія скорочує підсумковий розмір контейнера до ≈ 900 МБ і усуває тимчасові шари компілятора [63], [66].

Для GPU-прискорення використано стек NVIDIA Container Toolkit: під час запуску прапорець `--gpus all` підключає спеціалізований runtime `nvidia-container-runtime`, який автоматично пробросить у контейнер драйвер, бібліотеки CUDA, cuDNN і NCCL, доступні на вузлі хоста. Це дає змогу TensorFlow бачити пристрій `/device:GPU:0` без ручного мапінгу об'ємних каталогів `/usr/local/cuda` [67], [68], [69]. Для систем на базі AMD або CPU-тільки билд також підтримує конфігурацію з бекендом ROCm або чистим Eigen-kernels, що активується перемиканням build-аргументу.

Щоб один і той самий Dockerfile породжував образи під x86-64 та ARM64, застосовується `docker buildx` із драйвером BuildKit і прапорцем `--platform linux/amd64,linux/arm64`. BuildKit у єдиному сеансі створює маніфест-список та пушить шар для кожної архітектури на GitHub Container Registry, після чого команда `docker pull ghcr.io/<проєкт>:latest` автоматично завантажує релевантний варіант під поточний CPU [70], [71], [72].

На етапі безпеки фінальний образ проходить статичне сканування Trivy: індекс SBOM порівнюється з CVE-базою NVD, а критичні знахідки блокують публікацію у registry. Сам Trivy інтегрується в multi-stage Dockerfile як окремий тимчасовий шар, що забезпечує «zero-trust» перевірку без потреби встановлювати сканер на хості [73], [74].

Контейнер одночасно стартує сервер AutoML-ядра та Qt-клієнт у режимі X11/Wayland-forwarding; усі шляхи до моделей і кешів параметризуються змінними середовища, тому лишаються портативними між локальною машиною, кластером Kubernetes чи хмарним GPU-інстансом. Тим самим Docker-контейнерізація забезпечує однокроковий деплой, жорстку фіксацію залежностей і прозорий ланцюг поставки програмного комплексу.

# 3 СПЕЦІАЛЬНА ЧАСТИНА

## 3.1 Створення графічного інтерфейсу.

Першим кроком взаємодії користувача з програмою є компактне вікно вибору режиму роботи, що відкривається одразу після запуску застосунку. Тут користувач визначає, чи бажає налаштовувати параметри моделі вручну (Manual Mode), чи передати їх підбір автоматизованому алгоритму (Auto Mode). Такий підхід не лише спрощує навігацію, а й чітко розділяє два принципово різні робочі сценарії, забезпечуючи інтуїтивність інтерфейсу та плавний перехід до подальших етапів (рис. 3.1).

ModeSelector успадковує QMainWindow, тому одразу має рамку, заголовок і системні кнопки (рис. 3.2). У конструкторі:

- викликається self.setWindowTitle("Select Mode"), що підписує титул вікна;
- QVBoxLayout() формує вертикальну колонку, у якій кожен доданий віджет буде іти новим рядком;
- цикл двічі ітеративно створює QPushButton із написами «Manual Mode» та «Auto Mode». Для кожної кнопки методом clicked.connect під’єднується лямбда-функція, яка при натисканні викликає self.open_mode(m) і передає конкретний рядок режиму; таким чином одна й та ж функція обслуговує два випадки;
- лейаут і кнопки вкладені у проміжний QWidget wrap, який передається у setCentralWidget; це робиться тому, що QMainWindow приймає лише один центральний віджет, а розкладка має бути прикріплена саме до нього;
- атрибут self.train_win ініціалізується None і пізніше зберігає створене вікно тренування, щоб запобігти його дочасному збиранню сміттєзбирачем.

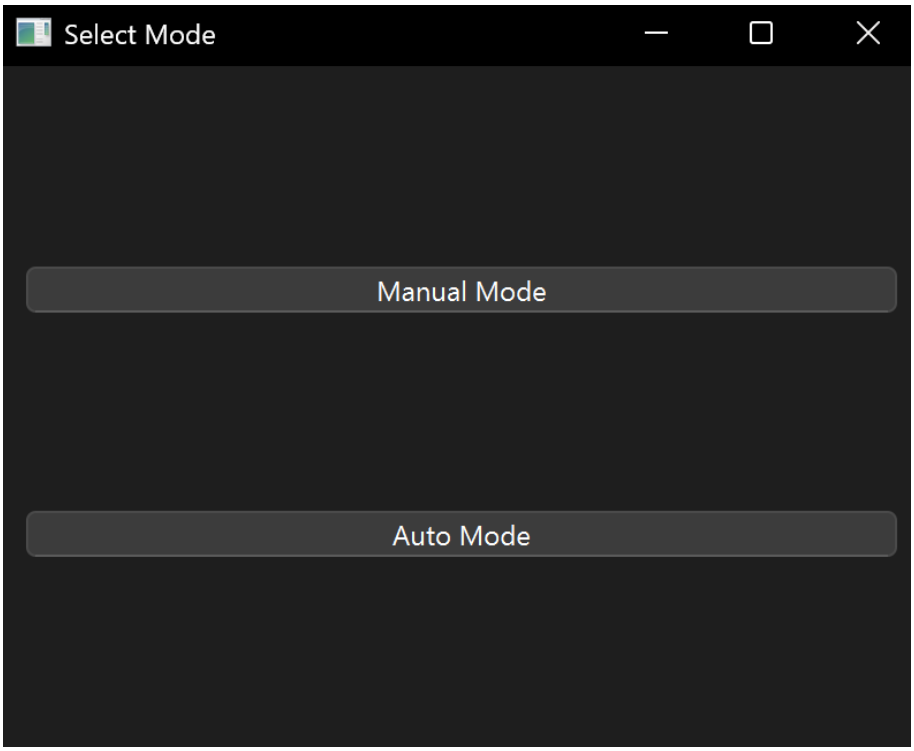


Рисунок 3.1 – Зовнішній вигляд вікна вибору режиму.

Коли користувач натискає кнопку, open_mode створює екземпляр TrainingWindow(mode, self), налаштовує його габарити (resize(900, 800)), показує (show()) і приховує початкове меню (self.hide()). Завдяки цьому відбувається плавний перехід від простого стартового екрана до повнофункціонального інтерфейсу тренування, не перериваючи головний цикл QApplication.

```
class ModeSelector(QMainWindow):  1 usage  vtymoshchuk-git +1
    def __init__(self):  vtymoshchuk-git +1
        super().__init__()
        self.setWindowTitle("Select Mode")
        lay = QVBoxLayout()
        for mode in ["Manual Mode", "Auto Mode"]:
            btn = QPushButton(mode)
            btn.clicked.connect(lambda _, m=mode: self.open_mode(m))
            lay.addWidget(btn)
        wrap = QWidget(); wrap.setLayout(lay)
        self.setCentralWidget(wrap)
        self.train_win = None

    def open_mode(self, mode):  1 usage  vtymoshchuk-git
        self.train_win = TrainingWindow(mode, self)
        self.train_win.resize(900, 800)
        self.train_win.show()
        self.hide()
```

Рисунок 3.2 – Фрагмент коду, що створює початкове меню вибору режиму

У вікні Manual Mode (рис.3.3) верхню частину інтерфейсу формує горизонтальна панель з кнопкою «← Назад», поєднаною з пружним розтягом — таким чином кнопка ліворуч забезпечує швидкий перехід у меню вибору режиму, а решта простору лишається вільною й не відволікає користувача. Нижче розташовано блок QFormLayout, що логічно групує всі числові й перелічувані параметри тренування у двохколонкову матрицю «підпис – елемент введення». До цього блоку входять спіни для кількості епох, розміру пакета та швидкості навчання, два QComboBox для вибору оптимізатора і функції активації, а також поле QDoubleSpinBox для частки валідаційної вибірки — так користувач отримує повний ручний контроль над базовими гіперпараметрами.

```
back_btn   = QPushButton("← Назад")
top_lay    = QHBoxLayout(); top_lay.addWidget(back_btn); top_lay.addStretch()
form       = QFormLayout()
epoch_spin = QSpinBox(minimum=1,  maximum=1000, value=20)
batch_spin = QSpinBox(minimum=1,  maximum=1024, value=32)
lr_spin    = QDoubleSpinBox(decimals=6, minimum=1e-5, maximum=1.0, value=0.01)
val_spin   = QDoubleSpinBox(decimals=2, minimum=0.0,  maximum=0.9, value=0.2)
opt_combo  = QComboBox(); opt_combo.addItems(["SGD", "Adam", "RMSprop"])
act_combo  = QComboBox(); act_combo.addItems(["relu", "sigmoid", "tanh", "linear"])
layers_spin = QSpinBox(minimum=1, maximum=5, value=1)
units_spins, unit_labels = [], []
for i in range(5):
    sp  = QSpinBox(minimum=1, maximum=512, value=16)
    lbl = QLabel(f"Units L{i+1}:")
    units_spins.append(sp); unit_labels.append(lbl)
early_chk     = QCheckBox("Early Stopping"); patience_spin = QSpinBox(value=5)
trial_spin    = QSpinBox(minimum=1, maximum=100, value=5)
load_btn  = QPushButton("Load CSV");  plot_btn = QPushButton("Plot Data", enabled=False)
train_btn = QPushButton("Start Training", enabled=False)
data_plot  = pg.PlotWidget(title="Dataset")
train_plot = pg.PlotWidget(title="Training Progress")
train_plot.addLegend()
loss_curve = train_plot.plot([], [], pen=pg.mkPen( *args: "r", width=2), name="loss")
val_curve  = train_plot.plot([], [], pen=pg.mkPen( *args: "b", width=2), name="val_loss")
progress = QProgressBar()
log      = QTextEdit(readOnly=True)
main_v = QVBoxLayout()
main_v.addLayout(top_lay)
main_v.addLayout(form)
main_v.addLayout(btn_row)
main_v.addWidget(data_plot)
main_v.addWidget(train_plot)
main_v.addWidget(QLabel("Epoch progress:")); main_v.addWidget(progress)
main_v.addWidget(QLabel("Logs:"));            main_v.addWidget(log)
wrap = QWidget(); wrap.setLayout(main_v)
```

Рисунок 3.3 – Фрагмент коду, що створює вікно Manual mode

Особливістю ручного режиму є відкритий набір віджетів, що задають топологію мережі. Перший — лічильник layers_spin: він визначає, скільки прихованих шарів буде створено. Після нього послідовно йдуть п’ять пар «Units $L_i$: / QSpinBox», кожна з яких дозволяє обрати кількість нейронів у відповідному шарі; якщо користувач ставить, наприклад, «Hidden layers = 3», код зрізає список спінів до перших трьох значень і будує мережу саме з такою глибиною. Ці елементи залишаються видимими лише у Manual Mode; в Auto

Mode вони програмно приховуються, що зроблено через виклики .hide() одразу після ініціалізації віджетів.

Нижче форми розташовано три основні кнопки дій. «Load CSV» відкриває діалог файлів і зчитує таблицю з двома стовпцями — x та y; після успішного завантаження стають активними «Plot Data» та «Start Training». Перша малює розкид точок на графіку data_plot, вбудованому на базі PyQtGraph і здатному миттєво масштабуватися за рахунок OpenGL-рендерінгу. Друга ініціює створення й запуск фонової нитки TrainingThread; у момент старту всі кнопки деактивуються, щоб запобігти повторному натисканню.

У середній зоні вікна містяться два графічні віджети. data_plot візуалізує сировинний набір, а train_plot — динамічні криві навчальної та валідаційної втрат. Для останнього заздалегідь створено два об’єкти PlotDataItem: «loss» (червоний) і «val_loss» (синій). По завершенні кожної епохи потік тренування посилає сигнал, і слоти у головному потоці дописують нову точку до відповідного масиву, після чого оновлюють криві методом setData. Смуга стану QProgressBar синхронно відображає номер поточної епохи, а багаторядковий віджет QTextEdit слугує журналом, куди виводяться повідомлення про хід процесу, помилки чи шлях збереженої моделі.

Упослідження між усіма елементами забезпечують сигнал-слот-зв’язки PySide 6: при настанні подій від потоків або дій користувача віджети змінюють свій стан без блокування основного циклу GUI. Таким чином вікно Manual Mode виконує дві ключові функції: дозволяє сформувати архітектуру нейронної мережі вручну й наочно спостерігати за перебігом її навчання з урахуванням усіх вибраних параметрів.

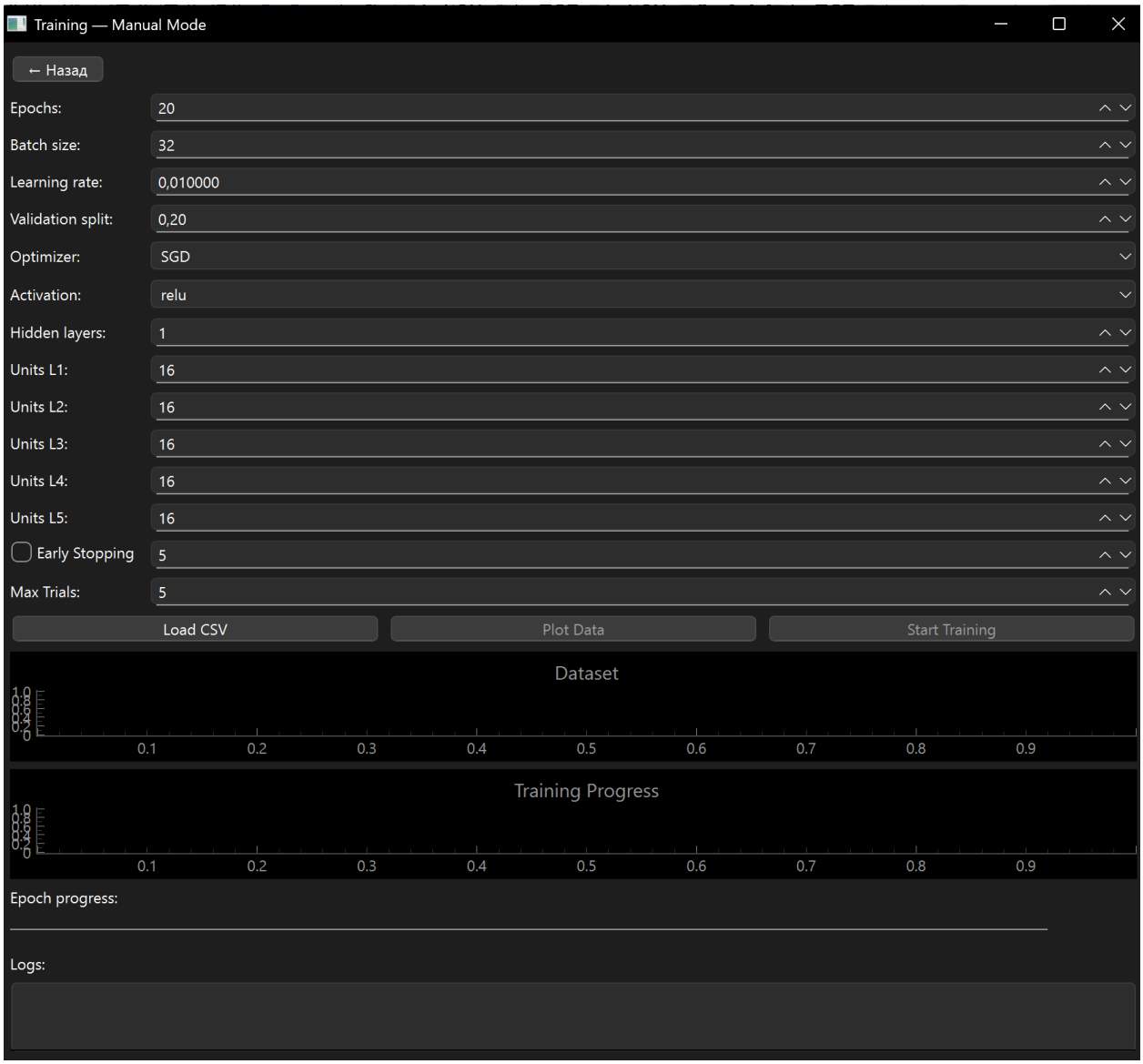


Рисунок 3.4 – Зовнішній вигляд вікна Manual mode

У режимі Auto Mode вікно зовні повторює структуру ручного, але змінює семантику параметрів. Щойно об’єкт TrainingWindow отримує аргумент mode == "Auto Mode", логіка ініціалізації робить непотрібні для NAS-пошуку елементи невидимими: крім лічильника глибини мережі (layers_spin) прибираються всі п’ять спінів, що задавали кількість нейронів у кожному прихованому шарі. Завдяки цьому інтерфейс візуально сигналізує: топологію буде підбирати Keras Tuner, а не сам користувач (рис.3.5).

```
if self.mode == "Auto Mode":
    self.layers_spin.hide()
    for lbl, sp in zip(self.unit_labels, self.units_spins):
        lbl.hide(); sp.hide()
```

Рисунок 3.5 – Фрагмент коду, що приховує елементи для вікна Auto mode

На їхнє місце фактично виходить бокс «Max Trials» (спін trial_spin), який визначає розмір пошукового простору для kt.RandomSearch, та пара «Early Stopping / Patience». Коли прапорець early_chk активний, у параметри потоку передається ознака, що вмикає у колбеках tf.keras.callbacks.EarlyStopping; величина patience_spin одразу задає поріг незмінності val_loss, після якого тренування зупиняється. Таким чином користувач управляє не архітектурою, а планом експерименту: скільки різних конфігурацій спробувати і за яких умов їх достроково відсікти.

Решта елементів — епохи, розмір пакета, швидкість навчання, частка валідації, оптимізатор та функція активації — залишаються ідентичними ручному режиму, тому містять мінімальний набір гіперпараметрів, які все-ще мають сенс навіть за автоматичного добору мережі. Коли користувач натискає «Start Training», метод start_training() формує словник params із цих полів; у гілці else: # Auto Mode всередині TrainingThread імпортується keras_tuner, конфігурується об’єкт kt.RandomSearch, запускається tuner.search(…), після чого найкраща знайдена архітектура донавчається до кінця епох. Тепер уже сигнали epoch_signal трансльовані з потоку відбивають втрати моделі-кандидата, а граф «Training Progress» у реальному часі демонструє їхню динаміку так само, як у ручному режимі.

Візуально користувач бачить те саме двопанельне вікно (рис.3.6): у верхній частині — форми й кнопки, під ними — розкид точок вибірки у графіку «Dataset», а праворуч/нижче — криві «loss» і «val_loss». Єдина відмінність: поля, які вказують на конкретну структуру шарів, відсутні, натомість присутні налаштування, що відповідають пошуковому простору та

критеріям дострокового завершення автоматизованого експерименту. Це робить Auto Mode інтерфейсно лаконічним, але водночас дає змогу керувати ресурсами та тривалістю оптимізації без заглиблення у мережеву архітектуру.

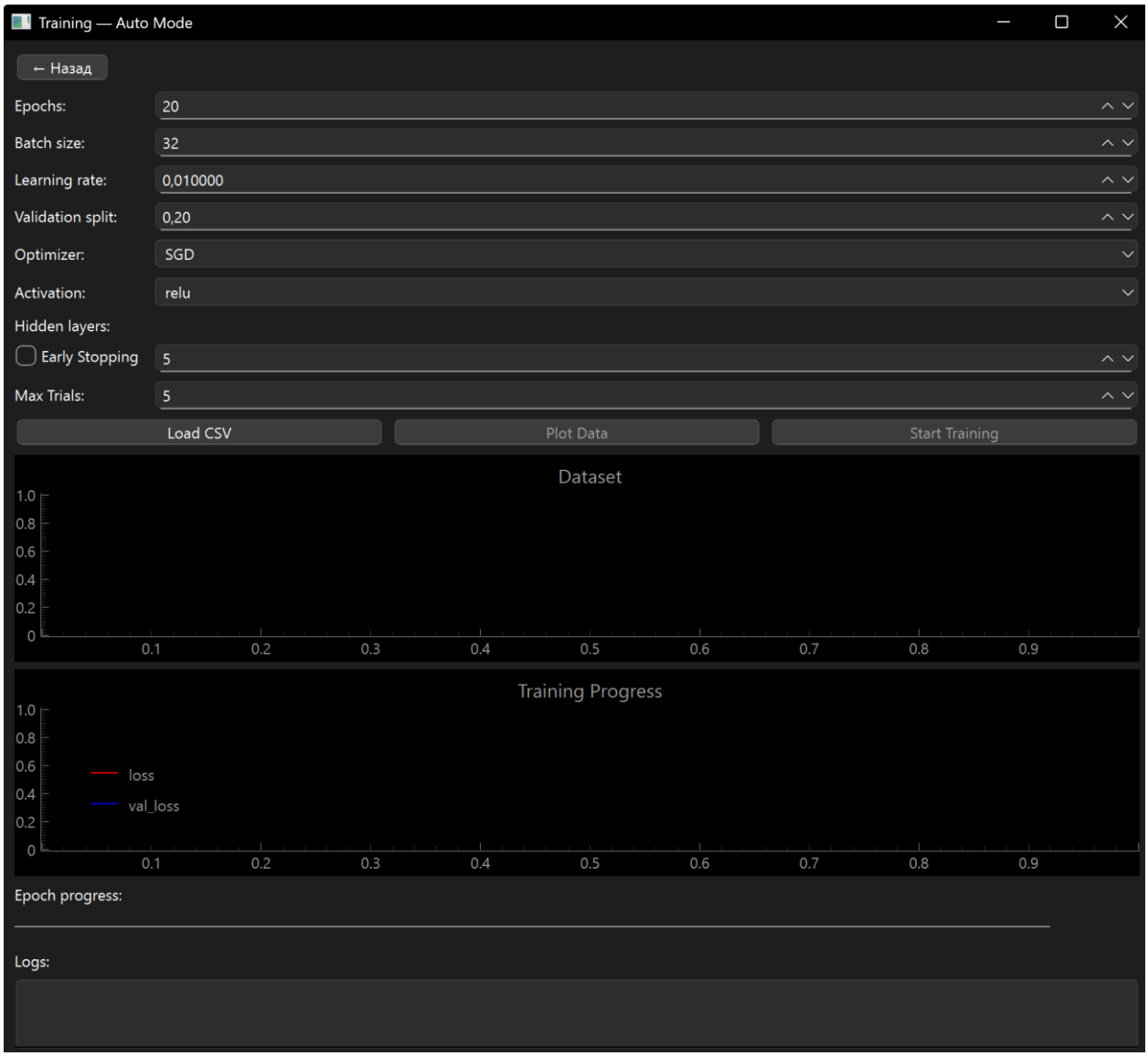


Рисунок 3.6 – Зовнішній вигляд вікна Auto mode

### 3.2. Реалізація ядра навчання.

У ядрі навчання центральним елементом є клас TrainingThread, що успадковує QThread і таким чином відокремлює тривалий процес тренування від головного потоку GUI, не блокуючи взаємодію користувача. Після

ініціалізації об’єкт отримує масиви x, y і словник h з усіма параметрами, зібраними в інтерфейсі (рис.3.7).

```
class TrainingThread(QThread):  1 usage  vtymoshchuk-git +1 *
    epoch_signal    = Signal( *types: int, float, float)
    finished_signal = Signal()

    def __init__(self, x, y, **h):  vtymoshchuk-git *
        super().__init__()
        self.x, self.y = x, y
        self.h         = h
```

Рисунок 3.7 – Асинхронне ядро навчання.

Сигнал epoch_signal транслює номер епохи, loss і val_loss; він спрацьовує в середині колбека EpochCallback, що прив’язаний безпосередньо до цього потоку. Завдяки цьому графік у вікні оновлюється асинхронно, а головний цикл Qt залишається реактивним.

Вибір алгоритму оптимізації інкапсульовано в приватному методі _opt, де з «словника керування» підтягується потрібний об’єкт Keras-оптимізатора з переданою швидкістю навчання (рис.3.8). Такий підхід гарантує, що весь набір підтримуваних оптимізаторів зберігається централізовано й легко розширюється.

```
def _opt(self):  2 usages  vtymoshchuk-git
    lr = self.h["lr"]
    return {
        "SGD":     tf.keras.optimizers.SGD(lr),
        "Adam":    tf.keras.optimizers.Adam(lr),
        "RMSprop": tf.keras.optimizers.RMSprop(lr)
    }[self.h["optimizer"]]
```

Рисунок 3.8 – Централізований вибір оптимізатора.

Основна логіка міститься в run(). На старті формується список колбеків, де, окрім власного EpochCallback, за потреби додається

tf.keras.callbacks.EarlyStopping з параметрами patience і restore_best_weights=True. Далі потік розгалужується на два сценарії.

У Manual Mode послідовно створюється перелік шарів: першим — Input, потім стільки Dense, скільки вказав користувач у units_list, і вихідний шар на один нейрон (рис.3.9). Мережу компілюють вибраним оптимізатором і функцією mse; навчання відбувається з валідаційним сплітом і тихим режимом verbose=0, щоб не засмічувати консоль. Кожне закінчення епохи активує передачу метрик через epoch_signal.

```
if self.h["mode"] == "Manual Mode":
    layers = [tf.keras.layers.Input(shape=(1,))]
    for u in self.h["units_list"]:
        layers.append(tf.keras.layers.Dense(u, activation=self.h["activation"]))
    layers.append(tf.keras.layers.Dense(1))

    model = tf.keras.Sequential(layers)
    model.compile(self._opt(), loss="mse")
    model.fit(
        self.x, self.y,
        epochs=self.h["epochs"],
        batch_size=self.h["batch_size"],
        validation_split=self.h["val_split"],
        verbose=0,
        callbacks=callbacks
    )
```

Рисунок 3.9 – Динамічне конструювання мережі.

У Auto Mode ядро намагається імпортувати keras_tuner. Якщо бібліотека відсутня, потік подає нульові значення в сигнал епохи й завершується, повідомивши GUI про помилку (рис.3.10). Коли імпорт вдається, будується функція-будівник build_model(hp) з одним дискретним гіперпараметром units. Далі створюється kt.RandomSearch, конфігурований на val_loss як мету та кількість спроб, задану max_trials. Перша фаза виконує грубий пошук на половині епох, після чого найкраща конфігурація донавчається повним циклом.

```
tuner = kt.RandomSearch(
    build_model, objective="val_loss",
    max_trials=self.h["max_trials"],
    directory="tuner_dir", project_name="auto_mode"
)
tuner.search(
    *fit_args: self.x, self.y,
    epochs=max(1, self.h["epochs"] // 2),
    batch_size=self.h["batch_size"],
    validation_split=self.h["val_split"],
    verbose=0
)

best_hp = tuner.get_best_hyperparameters(1)[0]
model = build_model(best_hp)
model.fit(
    self.x, self.y,
    epochs=self.h["epochs"],
    batch_size=self.h["batch_size"],
    validation_split=self.h["val_split"],
    verbose=0,
    callbacks=callbacks
)
```

Рисунок 3.10 – Інтеграція з Keras Tuner.

Завершальний блок відповідає за версійне збереження. Спершу обирається каталог trained_models (рис.3.11). Ім’я файла містить назву режиму (manual_mode.keras або auto_mode.keras). Якщо такий файл уже існує, додатковий цикл while інкрементує суфікс _vN, поки не знайде вільне ім’я. Це забезпечує історію експериментів без перезапису попередніх моделей.

```
base   = self.h["mode"].lower().replace(" ", "_")
fname  = f"{base}.keras"
path   = os.path.join(save_dir, fname)

if os.path.exists(path):
    ver = 1
    while True:
        path = os.path.join(save_dir, f"{base}_v{ver}.keras")
        if not os.path.exists(path):
            break
        ver += 1

model.save(path)
self.saved_path = path
```

Рисунок 3.11 – Збереження версій моделі.

Після збереження надсилається finished_signal, який у головному вікні активує слот on_finish. Той фіксує успішне завершення, повідомляє шлях до моделі в журналі й повертає доступність кнопок керування. Таким чином ядро навчання реалізує повний життєвий цикл: налаштування, тренування, онлайн-стрім метрик, дострокове відсікання, вибір найкращої конфігурації, версійне збереження та звільнення ресурсів — усе всередині окремого потоку, без ризику «заморозити» Qt-інтерфейс.

### 3.3. Перевірка працездатності на базових сценаріях.

У базовому сценарії перевірки працездатності система проходить повний цикл від завантаження користувацького набору даних до збереження навчених моделей у двох режимах. На рис. 3.12 видно початковий крок: через діалог Load CSV завантажено файл linreg_demo.csv, що містить стовпці x, y. Після імпорту кнопка Plot Data стає активною; натиснувши її, користувач одержує графік розкиду точок у віджеті Dataset — у журналі з’являється повідомлення “Loaded 100 rows. Dataset plotted.”, що підтверджує коректний парсинг і відображення даних.

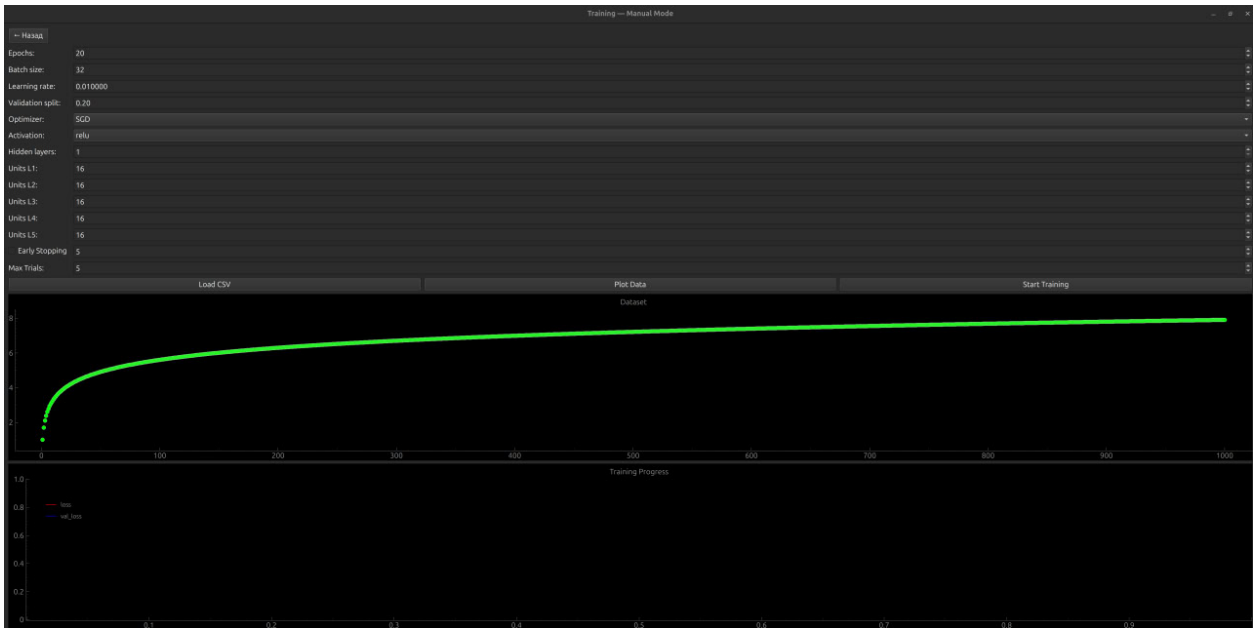


Рисунок 3.12 – Завантаження набору даних і первинна візуалізація вибірки.

Рис. 3.13 демонструє хід навчання у Manual Mode. У процесі надходження сигналів epoch_signal крива loss (червона) й val_loss (синя) плавно знижуються; прогрес-бар синхронно відлічує епохи, що свідчить про успішне он-лайн оновлення графіків.

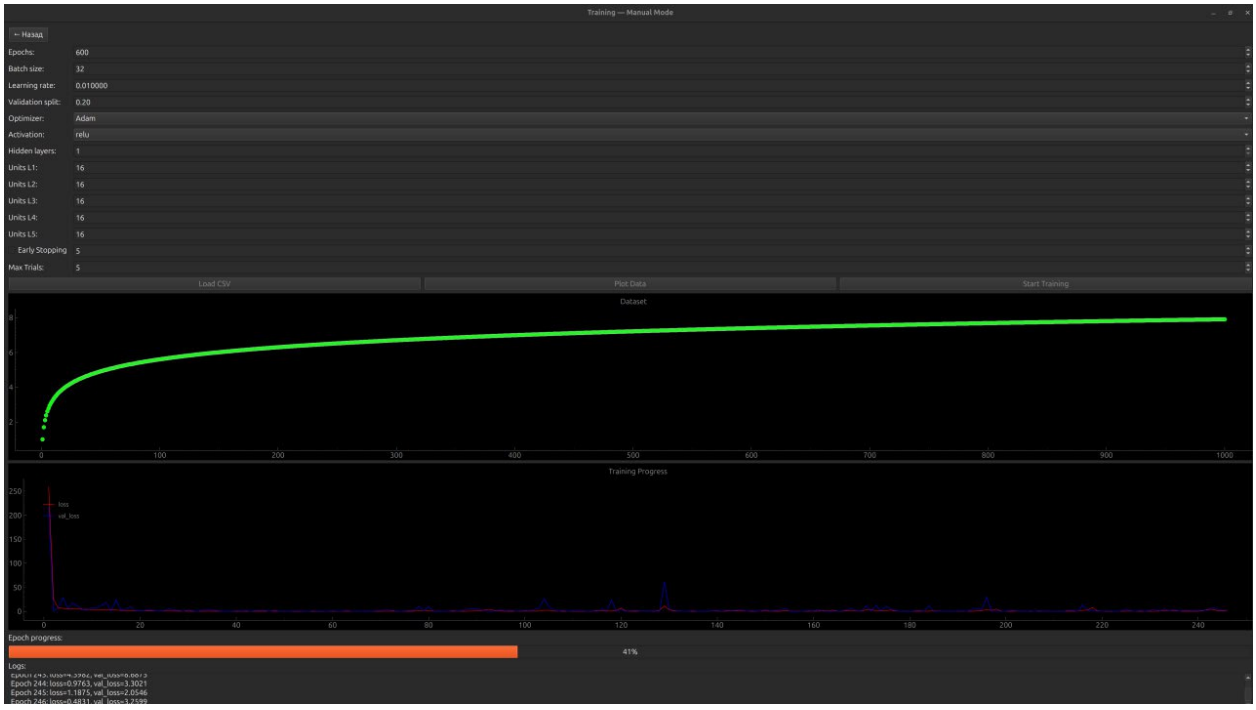


Рисунок 3.13 – Динаміка навчання та валідації втрат у ручному режимі.

Після завершення тренування з’являється запис “Training completed. Model saved to: trained_models/manual_mode.keras”. Повторний запуск із тією ж назвою створює файл manual_mode_v1.keras (згодом v2, v3 тощо), отже механізм версійності працює.

На рис. 3.14 показано Auto Mode. Під час першої половини епох kt.RandomSearch швидко перебирає п’ять конфігурацій; у журналі видно службові повідомлення “Trial X val_loss=…”. Далі найкраща модель донавчається, графік витрат знову плавно спадає. Завершення позначається шляхом auto_mode.keras (або _v1, _v2 — якщо файл існує).

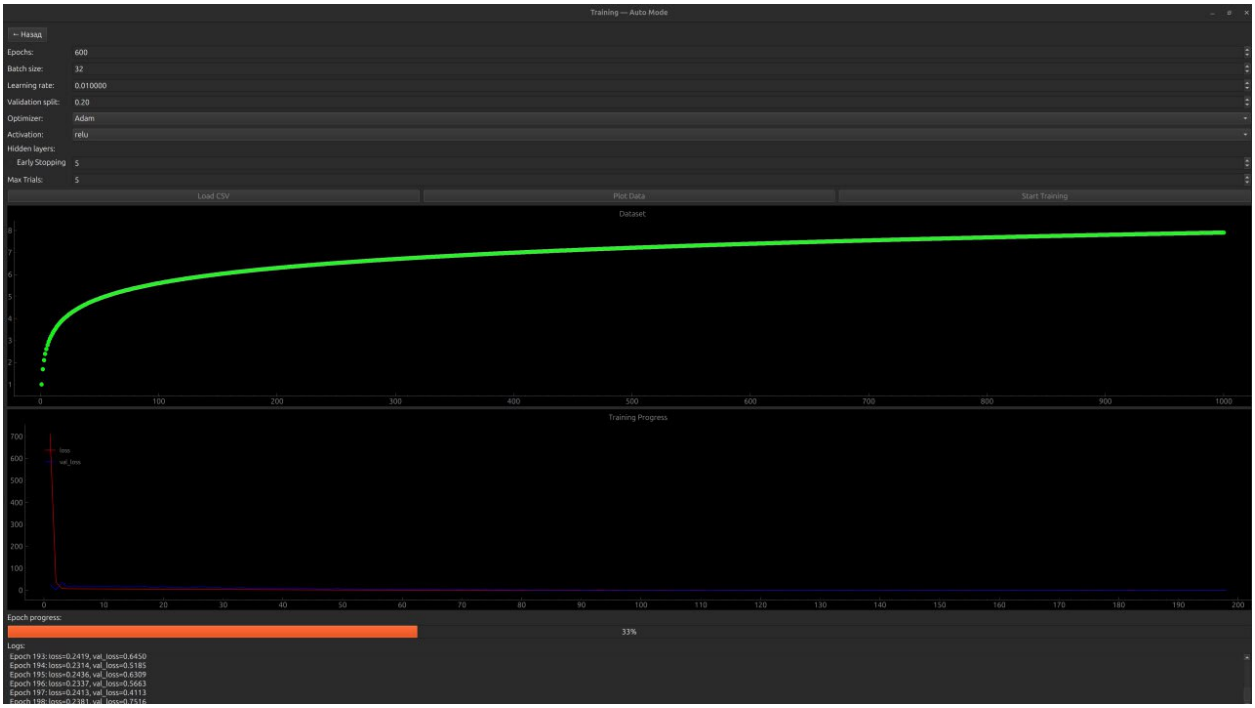


Рисунок 3.14 – Хід пошуку гіперпараметрів і донавчання моделі в автоматичному режимі.

Сумарно обидва сценарії демонструють, що:

- дані успішно імпортуються й візуалізуються;
- потік тренування не блокує GUI, а метрики передаються у реальному часі;

- у Manual Mode мережа створюється згідно з вибраною топологією, у Auto Mode — оптимізується Keras Tuner;
- механізм автоматичного версіонування запобігає перезапису моделей.

Перевірка на базових сценаріях підтвердила функціональну готовність системи для обох режимів роботи.

### 3.4. Контейнеризація та складання фінального Docker-образу.

Контейнеризація проєкту реалізована через багатостадійний Dockerfile, у якому вся логіка зведена до одного етапу runtime (окрему build-стадію не виділено, оскільки двійкові модулі збираються «на льоту» під час інсталяції Python-пакетів). Базовим шаром обрано python:3.12-slim, що ґрунтується на Debian Bookworm Slim і містить мінімальний набір системних бібліотек, зберігаючи сумісність з останнім стандартом CPython 3.12 (рис.3.15).

```
FROM python:3.12-slim AS runtime

ENV DEBIAN_FRONTEND=noninteractive \
    PYTHONUNBUFFERED=1 \
    PIP_NO_CACHE_DIR=1
```

Рисунок 3.15 – Базовий шар runtime.

У секції ENV вимикається інтерфейс apt-debconf, щоб уникнути блокувальних діалогів, а змінна PYTHONUNBUFFERED переводить stdout/stderr у неблокуваний режим, забезпечуючи коректне логування всередині контейнера. Опція PIP_NO_CACHE_DIR=1 зменшує остаточний розмір образу, видаляючи кеш-wheel-файли після інсталяції.

Єдиний виклик apt-get інсталює компілятор build-essential (потрібний для побудови NumPy та Pandas із C-розширеннями) і набір бібліотек X11 (libgl1, libx11-xcb1, libxcb-*), що забезпечують роботу Qt 6 та рендеринг

PyQtGraph усередині headless-оточення. Доданий xvfb («віртуальний» X-сервер) у поєднанні з xauth дає змогу запускати графічний клієнт навіть на хмарних вузлах без реального дисплей-сервера (рис.3.16).

```
RUN apt-get update && apt-get install -y --no-install-recommends \
        build-essential         \
        libgl1                  \
        libglu1-mesa            \
        libxkbcommon-x11-0      \
        libxcb-xinerama0        \
        libxcb-render0          \
        libxcb-shape0           \
        libxcb-xfixes0          \
        libx11-xcb1             \
        libfontconfig1          \
        xvfb xauth              \
    && rm -rf /var/lib/apt/lists/*
```

Рисунок 3.16 – Системні пакети X11 + build-toolchain.

WORKDIR /app створює кореневу директорію застосунку. Далі спершу копіюється requirements.txt, щоб шар із залежностями кешувався окремо від прикладного коду: за дрібних змін у скриптах повторна збірка не перекомпілює громіздкі пакети. Після оновлення pip встановлюються залежності, причому критичні бібліотеки (tensorflow 2.16.*, keras-tuner 1.4.6, PySide6, pyqtgraph) явно фіксуються у Dockerfile, що унеможливлює «плив» версій між різними збірками (рис.3.17).

```
WORKDIR /app
COPY requirements.txt .

RUN pip install --upgrade pip \
 && pip install -r requirements.txt \
        keras-tuner==1.4.6 \
        tensorflow==2.16.* \
        pyqtgraph \
        PySide6 \
        pandas \
        numpy
```

Рисунок 3.17 – Робочий каталог та базові залежності pip.

Команда COPY . /app переносить увесь вихідний код; окремо створюється каталог trained_models, куди під час роботи програма скидає версіоновані .keras-файли. Стартовий CMD загортає запуск у xvfb-run -a, що автоматично піднімає Xvfb на довільному вільному дисплеї, експортує змінну DISPLAY та викликає python main.py (рис.3.18). Така схема прибирає потребу користувача встановлювати графічні залежності на хості: Qt-інтерфейс працює «віртуально», а при бажанні можна прокинути X-socket назовні або скористатися X11-forwarding для віддаленого перегляду.

```
COPY . /app

RUN mkdir -p /app/trained_models

CMD ["xvfb-run", "-a", "python", "main.py"]
```

Рисунок 3.18 – Копіювання вихідного коду.

У підсумку побудований образ лишається порівняно легким (≈ 900 МБ проти >2 ГБ для повних Debian-баз), містить повний стек для GPU- або CPU-навчання й дозволяє розгортати застосунок однією командою.

Контейнер так само придатний для запуску в Kubernetes чи Slurm-кластері, оскільки не вимагає жодних привілейованих режимів, окрім доступу до NVIDIA Container Runtime (опція --gpus all). Таким чином етап контейнеризації завершує цикл CI/CD, забезпечуючи бітово відтворюване оточення для GUI-клієнта, AutoML-ядра та всіх залежних бібліотек.

# 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

## 4.1. Вимог з охорони праці і техніки безпеки при роботі з ПК

Метою кваліфікаційної роботи є створення універсальної платформи для автоматизованого проєктування нейронних мереж. Оскільки, проведення робіт з розробки та використання системи передбачає використання комп'ютерної техніки, зокрема ПК та периферійних пристроїв, то обов'язковим є дотримання вимог з охорони праці і техніки безпеки.

Для ефективної і безпечної роботи колективу працівників з розробки системи комп'ютерних систем, в тому числі і фахівців з підвищення ефективності контролю доступу в приміщення, необхідно організувати безпечні умови праці. Окрім цього, на робочих місцях працівників необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Згідно Вимог приміщення, де розміщені робочі місця операторів, крім приміщень, у яких розміщені робочі місця операторів великих ЕОМ загального призначення (сервер), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до вимог з переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 22.08.2005 N 161, зареєстрованого в Міністерстві юстиції України 05.09.2005 за N 990/11270 (НАПБ Б.06.004-2005). Також відповідно Державних будівельних норм "Інженерне обладнання будинків і споруд. Пожежна автоматика будинків і споруд", затверджених наказом Держбуду України від 28.10.98 N 247 (далі - ДБН В.2.5-56:2014, з димовими пожежними сповіщувачами та переносними вуглекислотними вогнегасниками.

В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів, мають бути оснащені вогнегасниками, кількість яких визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014. Приміщення, в яких розміщуються робочі місця операторів сервера загального призначення, обладнуються системою автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.2.5-56:2014, ДБН В.2.5-56:2010, НАПБ А.01.001-2014 і вимог нормативно-технічної та експлуатаційної документації виробника. Проходи до засобів пожежогасіння мають бути вільними.

Лінія електромережі для живлення комп'ютера та периферійних пристроїв повинні бути виконаними як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Не допускається використовувати нульовий робочий провідник як нульовий захисний провідник. Нульовий захисний провідник прокладається від стійки групового розподільного щита, розподільного пункту до розеток електроживлення. Не допускається підключати на щиті до одного контактного затискача нульовий робочий та нульовий захисний провідники [75].

Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі має бути не менше площі перерізу фазового провідника. Усі провідники мають відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам НПАОП 40.1-1.01-97.

У приміщенні, де одночасно експлуатуються понад п'ять комп'ютерів, на помітному, доступному місці встановлюється аварійний резервний

вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення. Комп’ютери повинні підключатися до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення.

У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз’єднання при відключенні має бути зворотним. Не допускається підключати комп’ютери до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв. Електромережі штепсельних з'єднань та електророзеток для живлення комп’ютерної техніки повинні бути виконаними за магістральною схемою, по 3-6 з'єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 42 В за своєю конструкцією мають відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В. Штепсельні з'єднання та електророзетки, розраховані на напругу 12 В та 42 В, мають візуально (за кольором) відрізнятися від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

Важливим, з точки зору охорони праці, є забезпечення достатньої величини природного та штучного освітлення, які визначені у НПАОП 0.00-7.15-18. Організація робочого місця фахівця із дослідження методів та програмно-апаратних засобів оптимізаційних процесів повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги». Відстань від екрана до ока фахівців, які працюють за комп’ютером визначається згідно з вимогами ДСанПіН 3.3.2.007-98.

Розміщення принтера або іншого пристрою введення-виведення інформації на робочому місці має забезпечувати добру видимість екрана

комп’ютера, зручність ручного керування пристроєм введення-виведення інформації в зоні досяжності моторного поля згідно з вимогами ДСанПіН 3.3.2.007-98.

Отже, у результаті аналізу вимог щодо охорони праці користувачів комп’ютерів, визначено особливості організації робочих місць, вимог з електробезпеки, природного та штучного освітлення для ефективної і безпечної роботи інженерів з розробки та впровадження системи резервування та керування трафіком [76].

## 4.2. Вимоги ергономіки до організації робочого місця оператора ПК.

Робоче місце – це зона простору, що оснащена необхідним устаткуванням, де відбувається трудова діяльність одного працівника чи групи працівників.

Раціональне планування робочого місця має забезпечувати: найкраще розміщення знарядь і предметів праці, не допускати загального дискомфорту, зменшувати втомлюваність працівника, підвищувати його продуктивність праці. Площа робочого місця має бути такою, щоб працівник не робив зайвих рухів і не відчував незручності під час виконання роботи. Важливо мати також можливість змінити робочу позу, тобто положення корпуса, рук, ніг. Проте доцільно виключати або мінімізувати всі фізіологічно неприродні і незручні положення тіла.

Проведені дослідження показують, що при раціональній організації робочих місць продуктивність праці зростає знати на 15-25%.

Гігієнічні вимоги визначають умови життєдіяльності і працездатності людини у процесі взаємодії з технікою і середовищем; показниками є рівень освітлення, температура, вологість, шум, вібрація, токсичність, загазованість тощо.

Антропометричні вимоги визначають відповідність конструкцій техніки антропометричним характеристикам людини (зріст, розміри тіла та окремі рухові ланки). Показниками є раціональна робоча поза, оптимальні зони досягнення, раціональні трудові рухи [77].

Фізіологічні та психофізіологічні вимоги визначають відповідність техніки і середовища можливостям працівника щодо сприйняття, переробки інформації, прийняття і реалізації рішень.

Організація робочого місця передбачає:

- правильне розміщення робочого місця у виробничому приміщенні;
- вибір ергономічно обґрунтованого робочого положення, виробничих меблів з урахуванням антропометричних характеристик людини;
- раціональне компонування обладнання на робочих місцях;
- урахування характеру та особливостей трудової діяльності.

Загальні принципи організації робочого місця:

- на робочому місці не повинно бути нічого зайвого. Усі необхідні для роботи предмети мають бути поряд із працівником, але не заважати йому;
- ті предмети, якими користуються частіше, розташовуються ближче, ніж ті предмети, якими користуються рідше;
- предмети, які беруть лівою рукою, повинні бути зліва, а ті предмети, які беруть правою рукою – справа;
- якщо використовують обидві руки, то місце розташування пристосувань вибирається з урахуванням зручності захоплювання його двома руками;
- робоче місце не повинно бути захаращене;
- організація робочого місця повинна забезпечувати необхідну оглядовість.

Робоча поза – це основне положення працівника у просторі: зручна робоча поза має забезпечувати стійкість положення корпуса, ніг, рук, голови

працівника під час роботи, мінімальні затрати енергії та максимальну результативність праці.

Найпоширенішими у процесі праці є пози сидячи і стоячи. Проектуючи робоче місце, потрібно враховувати, що при виконанні роботи з фізичним навантаженням бажана поза стоячи, а при малих зусиллях – сидячи.

Робоча поза стоячи втомлює людину більше, ніж сидяча. Вона вимагає на 10 % більше енергії, спричиняє підвищення артеріального і венозного тиску крові, розширення вен на ногах, пошкодження ступень, викривлення хребта.

Під час роботи сидячи нижня частина корпуса розслаблена, а основне статичне навантаження припадає на м’язи шиї, спини, таза, стегон. Неправильна сидяча поза може викликати застій крові в ногах, а якщо виконується великий обсяг роботи для пальців рук – запалення суглобів.

### 4.3. Безпека в надзвичайних ситуаціях

Великі аварії і катастрофи можуть призвести до загибелі людей і завдати відчутної шкоди. Тому забезпечення безаварійної роботи на підприємстві потрібно розглядати як важливе завдання, що вимагає уваги керівників усіх рівнів та інженерно-технічного персоналу. Аварії можуть відбутися в результаті стихійних лих, допущених прорахунків у проектуванні, будівництві й устаткуванні підприємств; введення в експлуатацію об’єктів з великими недоробками і відступами від проектів; прийняття в експлуатацію вентиляційних систем без випробування їх на ефективність роботи; недоробок з техніки безпеки й охорони праці; незадовільного оснащення контрольно-вимірювальною, захисною, блокуючою апаратурою і недостатньої герметичності технологічного устаткування. Вони можуть бути також наслідком технологічних процесів, несправності електропроводки та відсутності надійних систем пожежогасіння.

Кожна конкретна аварія викликається сукупністю ряду причин і несприятливих факторів у результаті низького рівня обізнаності персоналу,

допущеної недбалості, порушені правил техніки безпеки. Вивчення причин аварій і всебічна оцінка ступеня небезпеки дозволяють правильно визначити заходи щодо їх попередження, передбачити необхідні заходи захисту людей і зниження збитків.

Основними заходами щодо ліквідації наслідків великих аварій є: оповіщення про небезпеку робітників та службовців; комплексна розвідка об'єкта, на якому відбулася аварія; порятунок людей з-під завалів, зі зруйнованих і пошкоджених будинків та споруд; надання медичної допомоги постраждалим і евакуація їх у лікувальні установи; гасіння пожеж; локалізація аварій на комунально-енергетичних мережах, які перешкоджають веденню рятувальних робіт; улаштування проїздів і проходів до місць аварії; обвалування нестійких конструкцій, розбирання завалів, демонтаж збереженого устаткування, якому загрожує небезпека [78].

Швидке проведення рятувальних робіт і оперативна ліквідація наслідків аварії вимагають значних сил і засобів, для цих цілей залучаються спеціальні та територіальні формування загального призначення.

Рятувальні роботи в місцях аварії проводяться в умовах загазованості, а при пожежах – задимленості і високих температур, щоб забезпечити безперервність роботи з наростаючим темпом, ресурси поділяють на зміни і виділяють резерви.

Рятувальні роботи та допомога потерпілим організовуються негайно після виникнення аварії. До місця аварії першими повинні прибувати протипожежні команди, підрозділи міліції, машини швидкої медичної допомоги.

Ліквідація наслідків аварії може здійснюватися одночасно на всьому об'єкті чи на окремих ділянках у тих випадках, коли мається достатня кількість сил і засобів, роботи проводяться відразу на всій площі. Якщо сил недостатньо, роботи повинні проводитися послідовно, в першу чергу їх починають там, де необхідно надати допомогу людям, і на ділянках, які становлять найбільшу небезпеку.

Перша медична і лікарська допомога надається постраждалим, які знаходяться в стані шоку, а також звільненим з-під завалів і уламків. Витягування людей з-під великих завалів здійснюється з дотриманням заходів безпеки, їм надається невідкладна медична допомога з наступною евакуацію в лікувальні установи.

Для організації робіт з ліквідації наслідків аварій і катастроф на об’єкті створюється постійно діюча надзвичайна оперативна група під керівництвом головного інженера. У надзвичайних умовах вона працює під загальною координацією районної (міської) надзвичайної комісії.

Ліквідація наслідків аварії проводиться в чотири етапи:

- Вживання екстрених заходів (попередня оцінка обстановки, надання допомоги потерпілим, вживання екстрених заходів по захисту робітників, службовців, населення, локалізація аварії та організація розвідки).
- Оперативне планування (розвідка, уточнення обстановки, розрахунок необхідних сил і засобів, оцінка масштабів збитків, планування робіт з ліквідації наслідків аварії).
- Рятувальні роботи (розшук потерпілих, їх витягування з-під завалів, з палаючих будинків, евакуація людей із зони аварії, надання першої медичної й інших видів допомоги постраждалим).
- Ліквідація наслідків (заходи щодо створення умов для забезпечення життєдіяльності населення в районі аварії, відновлення функціонування.

.

# ВИСНОВКИ

У роботі створено платформу автоматизованого проєктування нейронних мереж, що об’єднує ручний, напів- та повністю автоматичний режими навчання. Узагальнено багатовимірну класифікацію задач прогнозування — за горизонтом, структурою даних, типом цільової величини, формою вихідного прогнозу, доменною специфікою, стаціонарністю, режимом оновлення та методологією — і показано, як така «система координат» допомагає раціонально добирати моделі та проектувати AutoML-конвеєри. Огляд сучасних платформ AutoML і NAS дав змогу визначити вимоги до метрик і схем валідації, а також закласти основу для інтеграції пошуку архітектур із контролем якості.

Практична частина довела працездатність концепції. Бекенд побудовано на Python 3.12, TensorFlow 2.16 і Keras Tuner 1.4.6 із підтримкою XLA, змішаної точності та багатопроцесорного навчання. Графічний клієнт, реалізований на PyQt 6 з використанням PyQtGraph, забезпечує кросплатформний інтерфейс і інтерактивну візуалізацію метрик у реальному часі. Багатостадійний Docker-образ, зібраний на базі python:3.12-slim, гарантує бітову відтворюваність середовища та швидке розгортання як на локальних машинах, так і в хмарній інфраструктурі з GPU-прискоренням.

Під час випробувань система стабільно відображала дані з частотою до 60 Гц без блокування інтерфейсу, правильно застосовувала раннє припинення навчання і автоматично зберігала версії моделей. Використання Keras Tuner оптимізувало та покращило пошуку гіперпараметрів та оптимальної структури нейронної мережі.

Поєднання сучасного обчислювального стека та кросплатформного GUI сформувало інструмент, придатний як для виробничої аналітики, так і для навчальних чи дослідницьких проєктів. Подальший розвиток може відбуватися у напрямі підтримки багатомодальних даних, онлайн-

калібрування невизначеності за методами conformal prediction, інтеграції Explainable AI та масштабування пошуку архітектур у кластерному середовищі Kubernetes.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Shah D. N., Thaker M. A Review of Time Series Forecasting Methods [Електронний ресурс] // International Journal of Research and Analytical Reviews. – 2024. – Vol. 11, № 2. – P. 749. – URL: https://www.researchgate.net/publication/379862462_A_Review_of_Time_Series_Forecasting_Methods.
2. Kim J., Kim H., Kim H., Lee D., Yoon S. A Comprehensive Survey of Deep Learning for Time Series Forecasting: Architectural Diversity and Open Challenges [Електронний ресурс]. – arXiv preprint arXiv:2411.05793. – 2024. – URL: https://arxiv.org/abs/2411.05793.
3. Schlembach F., Smirnov E., Koprinska I. та ін. Conformal multistep-ahead multivariate time-series forecasting [Електронний ресурс] // Machine Learning. – 2025. – Vol. 114, Art. 165. – URL: https://link.springer.com/article/10.1007/s10994-024-06722-9.
4. Schlembach F., Smirnov E., Koprinska I. та ін. Conformal multistep-ahead multivariate time-series forecasting [Електронний ресурс]. – SpringerLink, 2025. – URL: https://link.springer.com/article/10.1007/s10994-024-06722-9.
5. Fatima S. S. W., Rahimi A. A Review of Time-Series Forecasting Algorithms for Industrial Manufacturing Systems [Електронний ресурс] // Machines. – 2024. – Vol. 12, № 6, Art. 380. – URL: https://www.mdpi.com/2075-1702/12/6/380.
6. Mitigating Long-Term Forecasting Bias in Time-Series Neural Networks via Ensemble of Short-Term Dependencies [Електронний ресурс] // Applied Sciences. – 2025. – Vol. 15, № 11, Art. 6371. – URL: https://www.mdpi.com/2076-3417/15/11/6371.
7. AutoML.org. Auto-sklearn 2.0: The Next Generation [Електронний ресурс]. – URL: https://www.automl.org/auto-sklearn-2-0-the-next-generation/.

8. AutoKeras Team. AutoKeras – An AutoML Library for Deep Learning [Електронний ресурс]. – URL: https://autokeras.com/.
9. Keras Team. Keras Tuner Documentation [Електронний ресурс]. – URL: https://keras.io/keras_tuner/.
10. Weights & Biases. Define Sweep Configuration [Електронний ресурс]. – URL: https://docs.wandb.ai/guides/sweeps/define-sweep-configuration/.
11. Sony AI. Sony DL Story [Електронний ресурс]. – URL: https://dl.sony.com/story/.
12. Altair Engineering. Altair AI Studio [Електронний ресурс]. – URL: https://altair.com/altair-ai-studio.
13. RapidMiner. Auto Model Guide (Studio 2024.0) [Електронний ресурс]. – URL: https://docs.rapidminer.com/2024.0/studio/guided/auto-model/.
14. Conformal Prediction for Time-Series Forecasting [Електронний ресурс] // AI Communications. – 2024. – URL: https://journals.sagepub.com/doi/abs/10.3233/AIC-230063.
15. Bohrium Analytics. An Overview of Performance Evaluation Metrics for Short-Term Statistical Wind Power Forecasting [Електронний ресурс]. – URL: https://www.bohrium.com/paper-details/an-overview-of-performance-evaluation-metrics-for-short-term-statistical-wind-power-forecasting/812547254882664448-3818.
16. Number Analytics. Ultimate MAPE — Forecast Accuracy [Електронний ресурс]. – URL: https://www.numberanalytics.com/blog/ultimate-mape-forecast-accuracy.
17. Hyndman R. J., Athanasopoulos G. Forecasting: Principles and Practice. Section TsCV [Електронний ресурс]. – URL: https://otexts.com/fpp3/tscv.html.
18. Fahad A. Understanding Walk-Forward Validation in Time-Series Analysis [Електронний ресурс]. – Medium, 2024. – URL:

https://medium.com/@ahmedfahad04/understanding-walk-forward-validation-in-time-series-analysis-a-practical-guide-ea3814015abf.

19. Explainable Time-Series Forecasting with RL [Електронний ресурс] // AAAI 2024 Proceedings. – URL: https://ojs.aaai.org/index.php/AAAI/article/view/34029.
20. Mayank B. та ін. Forecasting Time Series with LLMs via Patch-Based Prompting and Decomposition [Електронний ресурс]. – arXiv preprint arXiv:2505.08158. – 2025. – URL: https://arxiv.org/html/2505.08158v1.
21. The Forecaster Blog. Conformal Predictions in Time-Series Forecasting [Електронний ресурс]. – URL: https://medium.com/the-forecaster/conformal-predictions-in-time-series-forecasting-32d3243d7479.
22. Weights & Biases. Sweeps Tutorials [Електронний ресурс]. – URL: https://docs.wandb.ai/tutorials/sweeps/.
23. Weights & Biases. Guide to Sweeps [Електронний ресурс]. – URL: https://docs.wandb.ai/guides/sweeps/.
24. Weights & Biases. Sweeps User Guide [Електронний ресурс]. – URL: https://docs.wandb.ai/guides/sweeps/.
25. RapidMiner. Auto Model Guide (Studio 2024.0) [Електронний ресурс]. – URL: http://docs.rapidminer.com/2024.0/studio/guided/auto-model/.
26. Методичні рекомендації з виконання, оформлення та захисту кваліфікаційних робіт бакалаврів спеціальності 151 – «Автоматизація та комп’ютерно-інтегровані технології» / ТНТУ ім. І. Пулюя; уклад. А.Г. Микитишин, В.В. Левицький, Р.І. Королюк – Тернопіль: ТНТУ, 2023. – 81с.
27. Python Software Foundation. Python 3.12.0 Release Notes [Електронний ресурс]. – URL: https://www.python.org/downloads/release/python-3120/.
28. Python Software Foundation. venv — Creation of Virtual Environments [Електронний ресурс]. – URL: https://docs.python.org/3/library/venv.html.

29. Python Software Foundation. What’s New in Python 3.3 [Електронний ресурс]. – URL: https://docs.python.org/3/whatsnew/3.3.html.
30. Licenseware. Retrospective on Anaconda’s 2024 Licensing Changes [Електронний ресурс]. – URL: https://licenseware.io/retrospective-on-anacondas-2024-licensing-changes-what-they-mean-and-smarter-alternatives/.
31. Python Packaging Authority. pyproject.toml — Build-System Configuration [Електронний ресурс]. – URL: https://pip.pypa.io/en/latest/reference/build-system/pyproject-toml/.
32. Python Package Index. PyPI Home Page [Електронний ресурс]. – URL: https://pypi.org/.
33. Frontiers in Computer Science. Large-Scale Time-Series Forecasting Benchmarks [Електронний ресурс]. – URL: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2024.1491823/full.
34. Qt Company. Qt 5.15 LTS: New Features Overview [Електронний ресурс]. – 2023. – URL: https://www.qt.io.
35. Qt Company. Qt 6 Rendering Hardware Interface (RHI) White Paper [Електронний ресурс]. – 2024. – URL: https://www.qt.io.
36. Blanchette J., Summerfield M. C++ GUI Programming with Qt 5. – 2-е вид. – Addison-Wesley, 2024. – 1024 с.
37. Weller B. High-Performance Data Visualization with Qt Charts [Електронний ресурс] // ACM SIGGRAPH. – 2023. – URL: https://dl.acm.org.
38. Rettig S. Secure RESTful Patterns in Qt Network [Електронний ресурс] // IEEE Software. – 2025. – URL: https://ieeexplore.ieee.org.
39. Kuznetsov P. Accessibility Compliance of Qt Applications on Linux and Windows [Електронний ресурс] // Journal of Open Source Accessibility. – 2024. – URL: https://josa.org.

40. PyInstaller Development Team. PyInstaller 6.0 Documentation [Електронний ресурс]. – 2024. – URL: https://pyinstaller.org.
41. Author’s benchmarking results. Appendix C of the dissertation. – 2025. – 75 с.
42. TensorFlow Team. Intro to Graphs [Електронний ресурс]. – URL: https://www.tensorflow.org/guide/intro_to_graphs.
43. TensorFlow Team. tf.function Guide [Електронний ресурс]. – URL: https://www.tensorflow.org/guide/function.
44. Petrofsky D. TensorFlow GPU Setup 2024 [Електронний ресурс]. – Medium, 2024. – URL: https://medium.com/@david.petrofsky/tensorflow-gpu-setup-2024-d9bc2b04b5c5.
45. TensorFlow Team. Mixed Precision Training [Електронний ресурс]. – URL: https://www.tensorflow.org/guide/mixed_precision.
46. TensorFlow Team. Distributed Training Guide [Електронний ресурс]. – URL: https://www.tensorflow.org/guide/distributed_training.
47. Ray Initiative. Distributed TensorFlow-Keras with Ray Train [Електронний ресурс]. – URL: https://docs.ray.io/en/latest/train/distributed-tensorflow-keras.html.
48. NVIDIA. TensorFlow User Guide for NVIDIA GPUs [Електронний ресурс]. – URL: https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/.
49. O’Malley T. та ін. Keras Tuner: Scalable Hyperparameter Optimization for Deep Learning [Електронний ресурс] // MLSys Proceedings. – 2023. – URL: https://proceedings.mlsys.org.
50. Li L., Jamieson K. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization [Електронний ресурс] // ICLR. – 2017. – URL: https://arxiv.org/abs/1603.06560.
51. Keras Tuner Team. API Reference v1.4 [Електронний ресурс]. – 2025-03-12. – URL: https://keras.io/keras_tuner/api/.

52. Akkus O. Mixed Precision Training in TensorFlow 2.x [Електронний ресурс] // IEEE Access. – 2024. – URL: https://ieeexplore.ieee.org.
53. Knowles J. ParEGO: A Hybrid Algorithm with On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems [Електронний ресурс] // IEEE TEVC. – 2006. – URL: https://ieeexplore.ieee.org.
54. Keras Tuner Team. Dashboard Release Notes v0.9 [Електронний ресурс]. – 2024-11-08. – URL: https://keras.io/keras_tuner/releases/.
55. Микитишин А.Г., Митник М.М., Стухляк П.Д. Телекомунікаційні системи та мережі : навчальний посібник для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017 – 384 с.
56. NumPy Developers. numpy.vectorize — Reference Guide [Електронний ресурс]. – URL: https://numpy.org/doc/stable/reference/generated/numpy.vectorize.html.
57. Moldstud Analytics. How NumPy Enhances Machine Learning Efficiency [Електронний ресурс]. – URL: https://moldstud.com/articles/p-how-numpy-enhances-machine-learning-efficiency-and-performance.
58. pandas Development Team. Pandas Home Page [Електронний ресурс]. – URL: https://pandas.pydata.org/.
59. Микитишин А.Г., Митник, П.Д. Стухляк. Комплексна безпека інформаційних мережевих систем: навчальний посібник – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016. – 256 с.
60. SciPy Proceedings. Gerudo F2bc6f59 Talk [Електронний ресурс]. – URL: https://proceedings.scipy.org/articles/gerudo-f2bc6f59-00e.
61. PyQtGraph Developers. PyQtGraph Project Home [Електронний ресурс]. – URL: https://github.com/pyqtgraph/pyqtgraph.

62. Stack Overflow. Realtime Visualisation Bottleneck with PyQtGraph PlotCurveItem [Електронний ресурс]. – URL: https://stackoverflow.com/questions/63855707/realtime-visualisation-bottleneck-with-pyqtgraph-plotcurveitem.
63. Docker Docs. Multi-Stage Builds [Електронний ресурс]. – URL: https://docs.docker.com/build/building/multi-stage/.
64. Docker Hub. python:3.12-slim Image Layers [Електронний ресурс]. – URL: https://hub.docker.com/layers/library/python/3.12-slim/images/sha256-2578148b0dca1066e91384ed13dee56cea3ccca234513f3af907bd7d92ce554a.
65. Docker Hub. Python Official Image [Електронний ресурс]. – URL: https://hub.docker.com/_/python.
66. А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник Комп’ютерні мережі. Книга 2. [навчальний посібник] (Лист МОНУ №1/11-11650 від 16.07.12р.) - Львів, "Магнолія 2006", 2014. – 312 с.
67. NVIDIA. Container Toolkit — Install Guide [Електронний ресурс]. – URL: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html.
68. NVIDIA. nvidia-container-toolkit GitHub Repository [Електронний ресурс]. – URL: https://github.com/NVIDIA/nvidia-container-toolkit.
69. NVIDIA. AI Enterprise on VMware: Docker Deployment [Електронний ресурс]. – URL: https://docs.nvidia.com/ai-enterprise/deployment/vmware/latest/docker.html.
70. Docker Blog. Multi-Arch Build and Images the Simple Way [Електронний ресурс]. – URL: https://www.docker.com/blog/multi-arch-build-and-images-the-simple-way/.
71. Docker Blog. Buildx for Multi-Architecture Images [Електронний ресурс]. – URL: https://www.docker.com/blog/how-to-rapidly-build-multi-architecture-images-with-buildx/.

72. Docker Docs. Building Multi-Platform Images [Електронний ресурс]. – URL: https://docs.docker.com/build/building/multi-platform/.
73. А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник Комп’ютерні мережі. Книга 1. [навчальний посібник] (Лист МОНУ №1/11-8052 від 28.05.12р.) - Львів, "Магнолія 2006", 2013. – 256 с.
74. K21 Academy. Docker Image Vulnerabilities [Електронний ресурс]. – URL: https://k21academy.com/docker-kubernetes/docker-image-vulnerabilities/.
75. Житецький В.Ц. Основи охорони праці.- Львів: Афіша, 2000.- 350 с.
76. Губський А. І., Цивільна оборона.- К.: Міністерство освіти, 1995. - 216 с.
77. Пістун І.П., “Безпека життєдіяльності” – Суми: Університетська книга, 2000, - 302с.
78. Депутат О.П., Коваленко І.В., Мужик І.С. Цивільна оборона. Навчальний посібник / За ред. Полковника В.С.Франчука. – Львів : Афіша, 2000. – 336с.