Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:
протокол засідання кафедри
№ ___ від _____.____.2025 р.
Завідувач кафедри ___ІПСіТ___
_____ Олешко О.І.

Пояснювальна записка

до дипломної роботи бакалавра

на тему «Розробка програмних засобів для розв'язання диференціальних рівнянь у фізичних задачах із застосуванням нейронних мереж»

Захищено на засіданні ЕК № _____
протокол № ____ від ____.____.2025 р.
Оцінка _____ / _____
Голова ЕК
_____ Фесенко Г.В.

Виконала:
Студентка 4 курсу, групи КС-42
Спеціальності:
122 – «Комп'ютерні науки»
Коляда Анна Володимирівна

Керівник к.ф.-м.н., доцент
Карась І.В.
Рецензент к.ф.-м.н., доцент
Хруслов М.М.

Харків — 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра інтелектуальних програмних систем і технологій
Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Спеціальність F3 Комп'ютерні науки

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІПСіТ

_____ Олег ОЛЕШКО
підпис ім'я, прізвище

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коляда Анна Володимирівна
(прізвище, ім'я, по батькові студента)

1. Тема роботи: «Розробка програмних засобів для розв'язання диференціальних рівнянь у фізичних задачах із застосуванням нейронних мереж»

керівник роботи: Карась Ірина В'ячеславівна, к. ф.-м. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «16» квітня 2025 року № 4101-5/962

2. Строк подання студентом роботи «2» червня 2025 року

3. Перелік питань, які потрібно розробити:

ознайомитися з особливостями диференціальних рівнянь, що моделюють фізичні процеси; проаналізувати існуючі методи їх розв'язання, зокрема класичні чисельні та нейромережеві підходи; розглянути алгоритми та принципи роботи фізично-інформованих нейронних мереж і альтернативних архітектур; створити моделі для типових задач теплопровідності, механіки деформованого тіла та гідродинаміки; здійснити програмну реалізацію, експериментальне дослідження розроблених моделей і порівняльний аналіз їх точності та ефективності з класичними методами.

4. План роботи

№ з/п	Назви етапів роботи
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи (КР).
2	Пошук та аналіз інформаційних джерел за темою КР.
3	Визначення особливостей фізично-інформованих нейронних мереж (PINNs) та альтернативних архітектур (DGM, DRM, Neural Operators).
4	Розробка математичних моделей типових фізичних задач: теплопровідність, деформація балки та рівняння Бюргерса.
5	Програмна реалізація нейромережевих моделей з використанням фреймворків TensorFlow/PyTorch, побудова функцій втрат, реалізація автоматичного диференціювання.
6	Реалізація функцій втрат, генерація точок колокації, впровадження автоматичного диференціювання.
7	Навчання нейронних мереж, налаштування гіперпараметрів та оптимізаційних стратегій.
8	Проведення експериментів, оцінка точності та збіжності моделей, побудова графіків та візуалізація результатів.
9	Порівняння результатів нейромережевих підходів із класичними чисельними методами.
10	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ШІ.

5. Дата видачі завдання 30 жовтня 2025

Студент _____ А.В. Коляда
(підпис) (ініціали, прізвище)

Керівник роботи _____ І.В. Карась
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Обсяг роботи 68 сторінок, 22 рисунки, 2 додатки, 33 джерела інформації.

Метою дипломної роботи є дослідження застосування фізично-інформованих нейронних мереж (PINNs) для розв’язання диференціальних рівнянь, що описують фізичні процеси (теплопровідність, деформація твердого тіла, гідродинаміка), та оцінка їх ефективності порівняно з класичними методами.

Методологія дослідження базується на використанні глибокого навчання й автоматичного диференціювання. Основними інструментами слугували Python із бібліотеками PyTorch, NumPy, Matplotlib та спеціально створені модулі.

У рамках роботи розроблено універсальний фреймворк для застосування PINN до задач різних класів. Проведено експерименти для трьох типових фізичних моделей, досягнуто високої точності за помірного часу навчання. Результати візуалізовано графічно, що підтверджує їх відповідність аналітичним розв’язкам.

Новизна роботи полягає у створенні універсальної архітектури для PINN із підтримкою похідних до 4-го порядку, застосуванні автоматичного диференціювання та аналізі розв’язків для різних типів рівнянь.

Отримані результати можуть бути використані в науковому моделюванні, інженерному аналізі та чисельній механіці, а також як навчальний приклад сучасного нейромережевого підходу до розв’язання диференціальних рівнянь. Реалізацію можна адаптувати до задач із ускладненою геометрією або параметричною невизначеністю.

Ключові слова: НЕЙРОННІ МЕРЕЖІ, ДИФЕРЕНЦІАЛЬНЕ РІВНЯННЯ, ФІЗИЧНІ ЗАДАЧІ, ЧИСЕЛЬНІ МЕТОДИ, МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, НАВЧАННЯ МОДЕЛЕЙ, ОПТИМІЗАЦІЯ ОБЧИСЛЕНЬ.

ABSTRACT

The work consists of 68 pages, 22 figures, 2 appendices, 33 sources of information.

The aim of this thesis is to explore the application of Physics-Informed Neural Networks (PINNs) for solving differential equations that describe various physical processes, including heat conduction, solid body deformation, and hydrodynamics. The study also evaluates their effectiveness in comparison with classical numerical methods.

The research methodology is based on the use of deep learning and automatic differentiation. The main tools included Python along with libraries such as PyTorch, NumPy, and Matplotlib, as well as custom-developed modules.

As part of the work, a universal framework was developed to apply PINNs to different classes of problems. Experiments were conducted on three representative physical models, achieving high accuracy within reasonable training time. The results were visualized graphically, confirming their consistency with analytical solutions.

The novelty of the work lies in the creation of a universal PINN architecture supporting derivatives up to the 4th order, the use of automatic differentiation in a real project, and comparative analysis of solutions for different types of equations.

The obtained results can be used in scientific modeling, engineering analysis, and computational mechanics, as well as serve as an educational example of a modern neural network approach to solving differential equations. The proposed implementation can be adapted for problems with complex geometry or parametric uncertainty.

Keywords: NEURAL NETWORKS, DIFFERENTIAL EQUATION, PHYSICAL PROBLEMS, NUMERICAL METHODS, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, MODEL TRAINING, COMPUTATIONAL OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

PINNs — Physics-Informed Neural Networks

DGM — Deep Galerkin Method

DGMT — Deep Galerkin Method with Timestepping

DRM — Deep Ritz Method

FNO — Fourier Neural Operator

DeepONet — Deep Operator Network

Adam — Adaptive Moment Estimation

L-BFGS — Limited-memory Broyden-Fletcher-Goldfarb-Shanno

AD — Automatic Differentiation (Автоматичне диференціювання)

MSE — Mean squared error (Середньоквадратична похибка)

ВСТУП

Сучасний світ важко уявити без математичного моделювання фізичних процесів. Протягом навчання на бакалавраті я неодноразово стикалася з диференціальними рівняннями як з інструментом для опису різноманітних явищ — від простих механічних систем до складних процесів теплопередачі чи гідродинаміки. Однак по мірі ускладнення досліджуваних моделей ставало очевидним, що навіть найдосконаліші класичні методи часто виявляються обмеженими або надто ресурсомісткими для практичного застосування.

Традиційні підходи до розв'язання диференціальних рівнянь, серед яких метод скінченних різниць та метод скінченних елементів, безумовно зарекомендували себе як надійні інструменти інженерного аналізу[1]. Проте досвід роботи з цими методами під час практичних занять показав, що вони часто потребують значних обчислювальних ресурсів і стикаються з серйозними труднощами при моделюванні складних нелінійних систем або задач з нетривіальною геометрією[2].

Ситуація суттєво змінилася зі стрімким розвитком штучного інтелекту та, зокрема, нейронних мереж. Опрацювавши ряд сучасних наукових публікацій, я звернулам увагу на зростаючий інтерес до застосування нейромережевих підходів у розв'язанні диференціальних рівнянь. Особливо цікавою виявилася концепція Physics-Informed Neural Networks (PINN), яка дозволяє "вбудовувати" фізичні закони безпосередньо в архітектуру нейронної мережі[3].

На відміну від суто обчислювальних методів, PINN пропонують принципово новий погляд на проблему — вони "навчаються" фізичним законам, одночасно забезпечуючи дотримання граничних та початкових умов[4]. Мої попередні експерименти з простими моделями показали, що такий підхід часто дозволяє отримувати розв'язки швидше та з кращою точністю, особливо для складних багатовимірних задач.

Актуальність даної роботи полягає в необхідності пошуку більш ефективних методів розв'язання диференціальних рівнянь для складних фізичних задач. Зі зростанням вимог до точності моделювання та обмеженістю обчислювальних ресурсів, нейромережеві підходи становлять перспективну альтернативу класичним методам, проте потребують детального дослідження їх ефективності в конкретних прикладних задачах.

Новизна моєї роботи полягає у систематичному порівнянні ефективності різних архітектур нейронних мереж для розв'язання трьох типових класів фізичних задач: теплопровідності, механіки деформованого тіла та гідродинаміки. На відміну від багатьох існуючих досліджень, які зосереджуються на теоретичних аспектах або обмежуються окремими прикладами, я планую провести комплексний аналіз із урахуванням точності, обчислювальних затрат та застосовності методів до різних типів диференціальних рівнянь.

В рамках дослідження я проведу серію експериментів із навчання нейронних мереж для розв'язання конкретних диференціальних рівнянь та порівняю отримані результати з розв'язками, отриманими за допомогою класичних чисельних методів. Особлива увага буде приділена аналізу впливу різних гіперпараметрів мережі на точність та швидкість отримання розв'язків.

Результати цієї роботи можуть мати практичне значення для різних галузей науки та інженерії, де необхідно швидко та точно розв'язувати складні диференціальні рівняння. Крім того, систематичне порівняння нейромережевих та класичних підходів дозволить краще зрозуміти переваги та обмеження різних методів та сформулювати рекомендації щодо їх застосування в залежності від типу задачі.

Об'єктом мого дослідження є процес розв'язання диференціальних рівнянь, що описують фізичні явища. Це досить широка область, яка охоплює різноманітні фізичні процеси — від простих механічних коливань до складних задач гідродинаміки. Диференціальні рівняння є тим математичним

інструментом, який дозволяє нам моделювати ці процеси, і саме їх розв'язання становить центральну проблему мого дослідження.

Предметом дослідження є застосування нейронних мереж, зокрема фізично-інформованих нейронних мереж (PINNs), для розв'язання диференціальних рівнянь у фізичних задачах. Це той конкретний аспект проблеми, на якому я зосереджую свою увагу. Мене цікавить, як саме можна "навчити" нейронну мережу розв'язувати диференціальні рівняння, враховуючи фізичні закони та обмеження.

Мета мого дослідження — це розробка та оцінка ефективності методів застосування нейронних мереж для розв'язання диференціальних рівнянь у фізичних задачах. Я прагну не лише реалізувати ці методи, але й порівняти їх з традиційними чисельними підходами, щоб зрозуміти, в яких ситуаціях нейромережеві методи можуть бути більш ефективними.

Для того, щоб досягти поставленої мети, я визначила конкретні завдання, виконання яких приведе до цілі:

1. Провести аналіз існуючих методів розв'язання диференціальних рівнянь, зокрема класичних чисельних методів та нейромережевих підходів.
2. Дослідити архітектуру та принципи роботи PINNs.
3. Розробити та реалізувати алгоритми навчання нейронних мереж для розв'язання диференціальних рівнянь з урахуванням фізичних законів.
4. Провести серію експериментів на конкретних фізичних задачах (теплопровідність, механіка деформованого тіла, гідродинаміка), порівнюючи результати роботи нейронних мереж з традиційними чисельними методами.
5. Проаналізувати вплив різних гіперпараметрів нейронної мережі на точність та швидкість отримання розв'язків.
6. Оцінити ефективність та точність розроблених методів, визначити їх переваги та обмеження.

7. Розробити рекомендації щодо застосування нейронних мереж для розв'язання диференціальних рівнянь у різних галузях фізики та інженерії.

Послідовне виконання цих завдань не тільки наблизить мене до досягнення головної мети дослідження, але й допоможе краще зрозуміти, де і як нейронні мережі можуть дійсно перевершити класичні підходи при вирішенні непростих фізичних задач. Зрештою, найцікавіше в цій роботі — побачити, як сучасні методи штучного інтелекту можуть змінити наш підхід до моделювання фізичного світу.

1 ОГЛЯД СУЧАСНИХ НЕЙРОМЕРЕЖЕВИХ ПІДХОДІВ ДО РОЗВ’ЯЗАННЯ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

1.1 Еволюція застосування нейронних мереж

Моделювання фізичних процесів за допомогою диференціальних рівнянь — це класичний підхід, який використовується вже багато десятиліть. Однак складність сучасних задач часто перевищує можливості традиційних методів. Саме тому останніми роками все більше дослідників звертають увагу на потенціал нейронних мереж у цій сфері[5].

Якщо подивитися на історію розвитку цього напрямку, можна помітити, що ідея використання нейронних мереж для розв'язання диференціальних рівнянь не нова — перші спроби датуються ще кінцем 90-х років минулого століття. Проте справжній прорив відбувся лише з розвитком глибокого навчання та появою потужних обчислювальних ресурсів[5].

Як зазначають Хан, Дженцен та Е у своїй роботі, традиційні чисельні методи стикаються з так званим "прокляттям розмірності", коли обчислювальна складність експоненційно зростає зі збільшенням розмірності задачі. Нейронні мережі пропонують альтернативний підхід, який потенційно може подолати це обмеження[5].

1.1.1 Історія розвитку методів

Розвиток нейромережевих підходів до розв'язання диференціальних рівнянь можна умовно розділити на кілька етапів.

Перші спроби використання нейронних мереж для розв'язання диференціальних рівнянь були зроблені ще наприкінці 90-х років. Тоді дослідники намагалися використовувати прості архітектури для апроксимації розв'язків, але обмежені обчислювальні можливості та недостатній розвиток методів навчання не дозволяли досягти значних результатів[5].

Другий етап почався приблизно в 2017-2018 роках, коли з'явилися перші роботи з фізично-інформованими нейронними мережами. Саме тоді Хан та співавтори запропонували використовувати глибокі нейронні мережі для розв'язання високорозмірних диференціальних рівнянь у частинних похідних. Їхній підхід базувався на використанні автоматичного диференціювання для обчислення похідних нейронної мережі, що дозволило ефективно мінімізувати залишок диференціального рівняння.

Зараз ми перебуваємо на третьому етапі, який характеризується розробкою спеціалізованих архітектур та методів навчання для різних типів диференціальних рівнянь. З'являються нові підходи, такі як вейвлет-базовані фізично-інформовані нейронні мережі, які дозволяють ефективно розв'язувати нелінійні диференціальні рівняння[6]

1.1.2 Основні переваги та виклики

Використання нейронних мереж для розв'язання диференціальних рівнянь має ряд переваг порівняно з традиційними методами.

По-перше, як показує практика, нейронні мережі можуть ефективно працювати з високорозмірними задачами, де традиційні методи стикаються з "прокляттям розмірності". Це особливо важливо для моделювання складних фізичних систем, які описуються диференціальними рівняннями у частинних похідних високої розмірності[5].

По-друге, розв'язки, отримані за допомогою нейронних мереж, є неперервними та диференційованими функціями, що часто буває зручніше для подальшого аналізу порівняно з дискретними розв'язками, отриманими за допомогою методу скінченних різниць або методу скінченних елементів[7].

По-третє, нейронні мережі можуть бути навчені на параметризованих сімействах рівнянь, що дозволяє швидко отримувати розв'язки для різних значень параметрів без необхідності повторного розв'язання рівняння[8].

Проте, як і будь-який метод, нейромережевий підхід має свої виклики та обмеження. Одним з основних викликів є складність навчання нейронних мереж для задач з розривними розв'язками. Традиційні архітектури нейронних мереж погано справляються з моделюванням розривів, що може призводити до появи осциляцій або згладжування в околі розривів[6].

Інший виклик пов'язаний з вибором оптимальної архітектури мережі та налаштуванням гіперпараметрів. Як зазначає Ромера, ефективність нейромережевого підходу сильно залежить від вибору архітектури, функцій активації та методів оптимізації. Неправильний вибір може призвести до поганої збіжності або низької точності розв'язків[7].

1.2 Фізично-інформовані нейронні мережі

Фізично-інформовані нейронні мережі — це особливий клас нейронних мереж, які включають фізичні закони безпосередньо в процес навчання. Цей підхід був запропонований кілька років тому і з тих пір став одним з найпопулярніших методів для розв'язання диференціальних рівнянь за допомогою нейронних мереж[9].

1.2.1 Принципи роботи PINNs

Основна ідея PINNs полягає в тому, щоб використовувати диференціальне рівняння як частину функції втрат при навчанні нейронної мережі. Замість того, щоб просто підганяти мережу під відомі дані (як це робиться у класичному машинному навчанні), PINNs також "штрафують" мережу за порушення фізичних законів, виражених через диференціальні рівняння.

Наприклад, ми маємо диференціальне рівняння виду:

$$\mathcal{D}u = f \tag{2.1}$$

де $\mathcal{D}$ — диференціальний оператор;

u — шукана функція;

f — задана функція.

Тоді функція втрат для навчання PINNs може бути записана як:

$$L(\theta) = L_{data}(\theta) + L_{physics}(\theta), \tag{2.2}$$

де $L(\theta)$ — загальна функція втрат;

θ — параметри нейронної мережі;

$L_{data}(\theta)$ — компонент, що відповідає за відповідність мережі навчальним даним (якщо вони є);

$L_{physics}(\theta)$ — компонент, що відповідає за задоволення диференціального рівняння:

$$L_{\text{physics}}(\theta) = \frac{1}{\text{N}}\sum_{i=1}^{N}\left|\mathcal{D}\hat{u}^{(x_i;\theta)} - f(x_i)\right|^2, \tag{2.3}$$

де $\hat{u}^{(x_i;\theta)}$ — вихід нейронної мережі з параметрами θ;

$\{x_i\}_{i=1}^{N}$ — набір точок, в яких ми перевіряємо виконання рівняння.

Для обчислення похідних нейронної мережі, які входять до диференціального оператора $\mathcal{D}$, використовується автоматичне диференціювання — потужний інструмент, доступний у сучасних фреймворках машинного навчання, таких як TensorFlow і PyTorch[5].

1.2.2 Основні архітектури та їх особливості

Існує кілька варіантів архітектури PINNs, які відрізняються способом представлення розв'язку та методом навчання.

Найпростіша і найпоширеніша архітектура — це багатошаровий перцептрон (MLP) з кількома прихованими шарами. Така мережа може апроксимувати досить широкий клас функцій, але іноді стикається з

труднощами при моделюванні складних розв'язків з високочастотними компонентами або розривами[7].

Для подолання цих обмежень дослідники запропонували різні модифікації базової архітектури. Наприклад, Уддін та співавтори розробили вейвлет-базовані фізично-інформовані нейронні мережі, які використовують вейвлет-функції для кращого моделювання різних масштабів у розв'язку. Цей підхід показав хороші результати для нелінійних диференціальних рівнянь[6].

Інший цікавий напрямок — це використання механізмів уваги в архітектурі PINNs. Такі мережі краще справляються з моделюванням локальних особливостей розв'язку і можуть бути ефективними для задач з розривними розв'язками[6].

1.2.3 Приклади застосування у фізичних задачах

Аналізуючи сучасні дослідження, можна побачити, що PINNs знайшли застосування у надзвичайно широкому спектрі фізичних задач. Наприклад, у блозі компанії SoftServe детально описується, як ці нейромережі успішно застосовуються для моделювання різноманітних фізичних процесів[3].

Особливо вражаючими є результати у сфері гідродинаміки, де PINNs дозволяють ефективно розв'язувати рівняння Нав'є-Стокса — одні з найскладніших рівнянь математичної фізики. Як показує Рут у своїй магістерській роботі, традиційні чисельні методи для цих рівнянь часто вимагають дуже дрібної сітки і, відповідно, величезних обчислювальних ресурсів. PINNs, з іншого боку, можуть забезпечити прийнятну точність з меншими обчислювальними витратами[8].

Не менш цікавими є застосування у задачах теплопровідності. Тут нейромережі демонструють вражаючу здатність враховувати складні граничні умови, що часто становить проблему для класичних методів. У своїй роботі Хе та співавтори показують, як багаторівневий фізично-інформований метод

глибокого навчання (MPIDL) дозволяє ефективно моделювати складні теплові процеси в конструкціях[10].

Для задач механіки деформованого тіла PINNs також виявились надзвичайно корисними. Вони дозволяють моделювати деформації твердих тіл під дією зовнішніх сил з високою точністю, враховуючи нелінійні властивості матеріалів, що підтверджується дослідженнями Хе та співавторів[10].

Хоча в проаналізованих джерелах менше уваги приділяється квантовій механіці та електромагнетизму, але й тут PINNs демонструють значний потенціал. Зокрема, для розв'язання рівняння Шредінгера та моделювання квантових систем, а також для розв'язання рівнянь Максвелла та моделювання електромагнітних полів[11].

Одним з найбільш практичних прикладів є застосування PINNs у промисловості. Як зазначається у блозі SoftServe, ці нейромережі можуть використовуватися для створення "цифрових двійників" фізичних систем, що дозволяє значно скоротити час і вартість розробки нових продуктів. Наприклад, у нафтовидобувній промисловості PINNs дозволяють створювати віртуальні витратоміри для електричних субмерсивних насосів, які працюють на порядки швидше, ніж традиційні обчислювальні методи, що особливо важливо для оперативного контролю та оптимізації виробничих процесів[3].

1.3 Альтернативні підходи

Хоча фізично-інформовані нейронні мережі зараз на піку популярності, вони не єдиний спосіб застосування нейронних мереж для розв'язання диференціальних рівнянь. Під час дослідження літератури я натрапила на кілька цікавих альтернативних підходів, які заслуговують на увагу.

1.3.1 Deep Galerkin Method

Deep Galerkin Method (DGM) — це один з найцікавіших підходів, який поєднує класичний метод Гальоркіна з потужністю глибоких нейронних мереж. Якщо згадати курс чисельних методів, то в класичному методі Гальоркіна ми шукаємо розв'язок у вигляді лінійної комбінації базисних функцій, а коефіцієнти визначаємо з умови ортогональності залишку до простору тестових функцій.

У DGM нейронна мережа використовується для представлення розв'язку, а функція втрат формується на основі варіаційного принципу. Як показують Сіріньяно та Сприггс у своїй роботі, це дозволяє ефективно розв'язувати еліптичні диференціальні рівняння, такі як рівняння Пуассона[12].

Також варто зауважити, що в цьому підході є можливість використання різних тестових функцій у DGM — це забезпечує гнучкість методу та дозволяє адаптувати його до різних типів задач.

1.3.2 Deep Ritz Method

Ще один метод, який привернув мою увагу — Deep Ritz Method (DRM). Як випливає з назви, він базується на варіаційному принципі Рітца. У цьому методі розв'язок диференціального рівняння шукається шляхом мінімізації відповідного функціоналу енергії.

Е та співавтори у своїй статті демонструють, що DRM особливо ефективний для розв'язання еліптичних диференціальних рівнянь, таких як рівняння Пуассона та стаціонарне рівняння Шредінгера. Під час аналізу їхніх результатів я звернула увагу на те, що DRM дозволяє знаходити наближені розв'язки з високою точністю навіть для задач високої розмірності[13].

Однією з ключових переваг DRM, яка може бути особливо корисною для моєї роботи, є його здатність природно враховувати граничні умови, що значно спрощує процес навчання нейронної мережі.

1.3.3 Neural Operator Methods

Третій напрямок, який здається мені перспективним — Neural Operator Methods. На відміну від PINNs, які апроксимують розв'язок конкретного диференціального рівняння, ці методи вивчають сам оператор, який відображає параметри рівняння (наприклад, коефіцієнти або граничні умови) у його розв'язок[14].

Особливо цікавим представником цього класу є Fourier Neural Operator (FNO), запропонований Лі та співавторами. FNO використовує перетворення Фур'є для ефективного моделювання нелокальних операторів. Автори демонструють вражаючі результати у розв'язанні параметризованих сімейств диференціальних рівнянь, таких як рівняння Нав'є-Стокса та рівняння Бюргерса[15].

Іншим представником є DeepONet (Deep Operator Network), розроблений Лу та співавторами. DeepONet базується на теоремі універсальної апроксимації для операторів і складається з двох підмереж: однієї для обробки вхідної функції та іншої для обробки точок, в яких обчислюється вихідна функція[16].

Аналізуючи ці методи, я дійшла висновку, що Neural Operator Methods можуть бути особливо корисними у випадках, коли потрібно багаторазово розв'язувати параметризовані диференціальні рівняння з різними параметрами, а така ситуація доволі часто виникає при моделюванні фізичних систем.

1.4 Порівняльний аналіз методів

1.4.1 Ефективність для різних типів диференціальних рівнянь

Різні нейромережеві методи мають різну ефективність залежно від типу диференціальних рівнянь, які ми намагаємося розв'язати.

PINNs, як показує практика, досить універсальні і можуть бути застосовані до широкого спектру рівнянь - від звичайних диференціальних рівнянь до складних систем рівнянь у частинних похідних. Вони особливо ефективні, коли у нас є обмежена кількість даних або коли ми хочемо включити фізичні обмеження безпосередньо в процес навчання[17].

DGM та DRM, з іншого боку, краще підходять для еліптичних рівнянь, таких як рівняння Пуассона. Вони забезпечують високу точність розв'язків і мають хороші теоретичні гарантії збіжності[18][19].

Neural Operator Methods найбільш ефективні, коли нам потрібно розв'язувати параметризовані сімейства рівнянь. Вони дозволяють навчити одну мережу, яка може генерувати розв'язки для різних значень параметрів, що може бути дуже корисно для задач оптимізації або аналізу чутливості[20].

Для гіперболічних рівнянь з розривними розв'язками, таких як рівняння збереження, найбільш ефективними є спеціалізовані архітектури, такі як вейвлет-базовані PINNs. Ці архітектури здатні точно моделювати розриви без необхідності введення штучної дисипації[21].

1.4.2 Обмеження та області застосування

Незважаючи на значні успіхи, нейромережеві методи розв'язання диференціальних рівнянь мають певні обмеження.

Одним з основних обмежень є складність навчання нейронних мереж для задач з розривними розв'язками[22]. Традиційні архітектури PINNs часто не здатні точно моделювати розриви, що призводить до осциляцій або згладжування в околі розривів. Для подолання цього обмеження необхідно використовувати спеціалізовані архітектури або методи навчання.

Іншим обмеженням є чутливість до вибору архітектури нейронної мережі та гіперпараметрів навчання[20]. Неправильний вибір може призвести до поганої збіжності або низької точності розв'язків. Для подолання цього

обмеження необхідно розробляти методи автоматичного вибору архітектури та гіперпараметрів.

Окрім того, нейромережеві методи можуть стикатися з труднощами при розв'язанні жорстких диференціальних рівнянь, де характерні часові масштаби різних процесів суттєво відрізняються[23]. Для подолання цього обмеження необхідно розробляти спеціальні методи навчання або використовувати адаптивні схеми інтегрування[24].

Незважаючи на ці обмеження, нейромережеві методи мають значний потенціал для розв'язання диференціальних рівнянь у фізичних задачах[25]. Вони особливо ефективні в ситуаціях, коли традиційні чисельні методи стикаються з труднощами, наприклад, при розв'язанні задач високої розмірності або при моделюванні складних нелінійних систем[12][5].

1.5 Сучасні тенденції та перспективи розвитку

Сучасні дослідження в області застосування нейронних мереж для розв'язання диференціальних рівнянь спрямовані на подолання обмежень існуючих методів та розширення їх можливостей.

1.5.1 Нові архітектури

Одним з найактивніших напрямків розвитку є створення спеціалізованих архітектур нейронних мереж для різних типів диференціальних рівнянь. Особливо цікавими, на мою думку, є Binary structured physics-informed neural networks (BS-PINNs), які розробили Лю та співавтори спеціально для рівнянь з швидко змінними розв'язками. На відміну від звичайних PINNs, ця архітектура використовує бінарну структуру, яка зменшує кількість зв'язків між нейронами порівняно з повністю зв'язаними мережами. Це дозволяє ефективніше вловлювати локальні особливості розв'язків, що критично важливо для моделювання різких змін у рівняннях типу Бюргерса, Ейлера та Гельмгольца. У своїх експериментах автори

показали, що BS-PINNs демонструють вищу швидкість збіжності та точність порівняно зі стандартними PINNs[6].

Ще одним перспективним підходом, який привернув мою увагу, є Wavelet-based physics-informed neural networks (W-PINNs). Ці мережі використовують вейвлет-перетворення для представлення розв'язку в вейвлет-просторі, що дозволяє ефективно моделювати різні масштаби. Особливо вражає те, що W-PINNs не потребують автоматичного диференціювання для обчислення похідних і не вимагають попередньої інформації про поведінку розв'язку. Це робить їх ідеальними для сингулярно збурених задач, де розв'язок має різкі зміни або осциляції. Автори продемонстрували ефективність цього підходу на різноманітних тестових задачах, включаючи моделі ФітцХью-Нагумо, рівняння Гельмгольца та Максвелла[26].

Паралельно розвиваються гібридні архітектури, які поєднують нейронні мережі з традиційними чисельними методами. Наприклад, DGM, запропонований Сіріньяно та Спіліопулосом, використовує нейронні мережі для апроксимації розв'язку, а функцію втрат формує на основі варіаційного принципу[12]. Цікавим розширенням є Deep Galerkin Method with Timestepping (DGMT), розроблений Ву та Крістарою, який включає часову дискретизацію для параболічних рівнянь. Замість апроксимації розв'язку на всій області, DGMT апроксимує його в окремих часових точках, що дозволяє отримати більш точні результати навіть при менших обчислювальних витратах[27]. Також варто відзначити DGNet, який поєднує переваги розривного методу Гальоркіна з нейронними мережами, що дозволяє ефективно працювати з неоднорідними сітками та складними геометричними областями[28].

1.5.2 Застосування у складних фізичних системах

Занурившись у вивчення сучасних наукових публікацій, я виявила захопливі напрямки застосування нейромережевих методів для відтворення складних фізичних систем. Особливу увагу хочу привернути на

багаторівневий фізично-інформований метод глибокого навчання (MPIDL), розроблений для розв'язання задач обчислювальної механіки конструкцій. Цей новаторський підхід використовує агрегаційну модель, що об'єднує декілька нейронних мереж, кожна з яких опрацьовує лише диференціальні рівняння першого або другого порядку, котрі відображають різноманітні фізичні аспекти — геометричні співвідношення, конститутивні рівняння та умови рівноваги конструкції. Завдяки цьому вдається суттєво знизити порядок вихідних диференціальних рівнянь вищого порядку, з якими традиційні PINN впораються значно гірше[10].

Творці MPIDL переконливо довели, що їхній метод істотно підвищує обчислювальну точність, зменшуючи похибку розрахунків балочних і оболонкових конструкцій до 0,5% і 2% відповідно, що значно перевершує результати класичних PINN. Водночас швидкодія обчислень зростає приблизно вчетверо[10]. На мій погляд, це робить MPIDL надзвичайно перспективним інструментом для інтеграції в системи цифрових двійників конструкцій, де критично важливими є як прецизійність, так і оперативність розрахунків.

Не менш захопливим напрямком виявилося застосування нейромережевих методів для розв'язання інтегро-диференціальних рівнянь, які повсякчас постають у квантовій механіці та суміжних галузях фізики. Скажімо, у дослідженні Елфвінга та його колег запропоновано витончене розширення парадигми PINNs за допомогою автоматичного інтегрування для обчислення комплексних інтегральних перетворень [29]. Такий підхід уможливлює не лише обчислення інтегральних перетворень на підґрунті навчених розв'язків, але й розв'язання інтегро-диференціальних рівнянь, де інтеграли обчислюються безпосередньо під час навчання.

Також вельми цікавий приклад практичного втілення подібного підходу я відшукала в праці Войтак та співавторів, де PINN застосовуються для розв'язання рівнянь нейронного поля (neural field equations) — інтегро-диференціальних рівнянь, що змальовують просторово-часову динаміку

нейронних популяцій у корі головного мозку[30]. Науковці вдаються до швидкого перетворення Фур'є для ефективного обчислення інтегрального члена, що суттєво зменшує обчислювальне навантаження.

1.5.3 Інтеграція з іншими методами машинного навчання

Третій вектор наукових пошуків розгортається на перетині нейромережевих методів розв'язання диференціальних рівнянь та інших парадигм машинного навчання. Зокрема, надзвичайно плідним виявилося поєднання PINNs із технологіями глибокого підкріплюючого навчання. Такий симбіоз відкриває шлях до елегантного розв'язання задач оптимального керування, де постає необхідність віднайти таке керування, яке б мінімізувало певний функціонал за наявності обмежень, що задаються диференціальними рівняннями [31]. У своїй праці Мухерджі та Лю демонструють, як цей підхід дозволяє ефективно розв'язувати задачі оптимального керування для нелінійних систем, що описуються рівняннями в частинних похідних, уникаючи при цьому обчислювально витратного розв'язання рівняння Гамільтона-Якобі-Беллмана[32].

Паралельно з цим розвивається надзвичайно перспективний напрямок інтеграції нейромережевих методів із технологіями автоматичного машинного навчання (AutoML). Ця синергія спрямована на автоматизацію одного з найскладніших етапів застосування нейронних мереж — вибору оптимальної архітектури та налаштування гіперпараметрів. У своєму дослідженні Гао та колеги пропонують фреймворк AutoPINN, який використовує байєсівську оптимізацію для автоматичного пошуку найефективнішої конфігурації фізично-інформованої нейронної мережі [33]. Експериментальні результати засвідчують, що такий підхід не лише суттєво зменшує залежність від експертного знання, але й часто дозволяє віднайти конфігурації, що перевершують ті, які були б обрані людиною-експертом. Це робить потужний інструментарій нейромережевих методів значно доступнішим для ширшого

кола дослідників та інженерів, які можуть не мати глибоких знань у галузі глибокого навчання.

1.6 Висновки та обґрунтування вибору методів для дослідження

Отже, на основі проведеного аналізу масиву публікацій я зробила певні висновки щодо застосування нейронних мереж для розв'язання диференціальних рівнянь.

По-перше, нейромережеві методи, особливо фізично-інформовані нейронні мережі (PINNs), демонструють вражаючий потенціал у моделюванні фізичних процесів. Вони дозволяють отримувати точні розв'язки навіть для складних багатовимірних задач, де традиційні методи виявляються неефективними або надто ресурсомісткими.

По-друге, незважаючи на всі переваги, існуючі нейромережеві підходи мають певні обмеження. Зокрема, вони часто стикаються з труднощами при моделюванні розривних розв'язків, характерних для багатьох фізичних задач. Крім того, вибір оптимальної архітектури мережі та налаштування гіперпараметрів залишаються нетривіальними завданнями, які суттєво впливають на якість отриманих результатів.

По-третє, сучасні дослідження в цій галузі розвиваються у трьох основних напрямках: розробка спеціалізованих архітектур для конкретних типів диференціальних рівнянь, застосування нейромережевих методів для моделювання складних фізичних систем та інтеграція з іншими методами машинного навчання для подолання існуючих обмежень.

Зваживши всі переваги та недоліки різних підходів, для свого дослідження я вирішила зосередитися на PINNs. Цей вибір не випадковий — PINNs мають унікальну здатність інтегрувати фізичні закони безпосередньо у процес навчання, що дозволяє отримувати не лише математично точні, але й фізично коректні розв'язки. Особливо мене привабила універсальність цього

методу — він показує хороші результати для різних типів диференціальних рівнянь, від звичайних до рівнянь у частинних похідних високого порядку.

Крім того, PINNs виявилися найбільш відповідними для тих фізичних задач, які я планую досліджувати — теплопровідності, механіки деформованого тіла та гідродинаміки. Для кожної з цих задач у літературі є успішні приклади застосування PINNs, що дає мені хорошу відправну точку для власних експериментів.

Звісно, я усвідомлюю обмеження обраного методу і планую приділити особливу увагу налаштуванню архітектури мережі та вибору оптимальних гіперпараметрів, щоб забезпечити максимальну точність розв'язків для досліджуваних фізичних задач.

2 МЕТОДОЛОГІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

2.1 Постановка задачі та вибір моделей для розв'язання

Проаналізувавши теоретичні основи застосування нейронних мереж для розв'язання диференціальних рівнянь, я перейшла до практичної частини мого дослідження. Головною метою цього розділу є розробка та реалізація методів розв'язання конкретних фізичних задач за допомогою PINNs та їх порівняння з класичними чисельними підходами.

Для забезпечення комплексності та репрезентативності дослідження я обрала три різні фізичні задачі, кожна з яких представляє окремий клас диференціальних рівнянь у частинних похідних. Такий вибір дозволяє оцінити ефективність нейромережевих методів у різноманітних фізичних контекстах та визначити їхні переваги та обмеження для кожного типу рівнянь.

2.1.1 Задача теплопровідності: параболічне рівняння

Першою обраною мною задачею є моделювання процесу теплопровідності, що математично описується параболічним диференціальним рівнянням. Ця задача має фундаментальне значення не лише у теплофізиці, але й у багатьох інших галузях, включаючи матеріалознавство, будівництво та енергетику.

Для конкретності я зосередилась на двовимірній задачі нестаціонарної теплопровідності в прямокутній пластині. Математична модель цього процесу описується наступним рівнянням:

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + f(x, y, t), \qquad (3.1.1)$$

де: $u(x, y, t)$ — температура в точці (x, y) у момент часу t;

α — коефіцієнт теплопровідності матеріалу;

$f(x,y,t)$ — джерело тепла або потужність.

Для повного формулювання задачі необхідно також визначити граничні та початкові умови. У своєму дослідженні я розглядаю наступні умови:

Початкова умова (розподіл температури в початковий момент часу):

$$u(x, y, 0) = u_0(x, y), \tag{3.1.2}$$

Граничні умови (температура або тепловий потік на межах області):

Умова Діріхле (задана температура):

$$u(x, y, t)|_{\Gamma_1} = g_1(x, y, t), \tag{3.1.3}$$

Умова Неймана (заданий тепловий потік):

$$\left.\frac{\partial u}{\partial n}\right|_{\Gamma_2} = g_2(x, y, t), \tag{3.1.4}$$

Умова Робіна (конвективний теплообмін):

$$\left.\frac{\partial u}{\partial n}\right|_{\Gamma_3} + h(u - u_\infty) = 0, \tag{3.1.5}$$

де Г1, Г2 та Г3 — різні ділянки границі розрахункової області;

$\frac{\partial}{\partial n}$ — похідна по нормалі до границі;

h — коефіцієнт теплообміну;

u_∞ — температура навколишнього середовища.

Для конкретних числових експериментів я обрала прямокутну область [0,1]×[0,1] з наступними параметрами:

- Коефіцієнт температуропровідності $\alpha = 0.01$
- Початкова умова: $u_0(x, y) = \sin(\pi x)\sin(\pi y)$
- Граничні умови Діріхле на всіх межах: $u(x, y, t)|_{\partial\Omega} = 0$

- Часовий інтервал моделювання: $t \in [0,1]$

Вибір саме цієї задачі для дослідження обумовлений кількома факторами. По-перше, для неї існує аналітичний розв'язок у вигляді ряду Фур'є, що дозволяє точно оцінити похибку нейромережевого методу. По-друге, параболічні рівняння часто викликають труднощі при чисельному розв'язанні через обмеження на крок за часом, пов'язані зі стійкістю. По-третє, задача теплопровідності має широке практичне застосування і є класичним прикладом для апробації нових обчислювальних методів.

2.1.2 Задача механіки деформованого тіла: еліптичне рівняння

Другою задачею, обраною для дослідження, є моделювання деформації пружної балки під дією зовнішнього навантаження. Ця задача описується еліптичним диференціальним рівнянням і має велике значення в інженерії, будівництві та матеріалознавстві.

Для спрощення математичного моделювання я розглядаю одновимірну задачу деформації тонкої пружної балки, яка описується рівнянням Ейлера-Бернуллі:

$$EI\frac{d^4w}{dx^4} = q(x), \tag{3.1.6}$$

де: E — модуль Юнга;

I — момент інерції перетворення;

w — прогин балки;

$q(x)$ — розподілена навантаження.

Для повного формулювання задачі необхідно визначити граничні умови. У моєму дослідженні я розглядаю балку з жорстко закріпленими кінцями, що відповідає наступним граничним умовам:

$$w(0) = 0,\ \frac{dw}{dx}(0) = 0,\ w(L) = 0,\ \frac{dw}{dx}(L) = 0, \tag{3.1.7}$$

де L — довжина балки.

Для конкретних числових експериментів я обрала наступні параметри:

- Довжина балки $L = 1$
- Добуток $EI = 1$ (нормалізоване значення)
- Навантаження у вигляді рівномірно розподіленої сили: $q(x) = q0$ (константа)
- Навантаження у вигляді зосередженої сили в середині балки:

$$q(x) = P\delta\left(x - \frac{L}{2}\right), \tag{3.1.8}$$

де δ — дельта-функція Дірака.

Значимість цієї задачі для дослідження обумовлена кількома факторами. По-перше, рівняння містить похідні четвертого порядку, що становить виклик для традиційних чисельних методів і, відповідно, дозволяє оцінити здатність нейронних мереж працювати з диференціальними операторами високого порядку. По-друге, ця задача має важливе практичне значення в інженерії та будівництві, де точний розрахунок деформацій є критично важливим. По-третє, для простих випадків навантаження існують аналітичні розв'язки, що дозволяє оцінити точність нейромережевого підходу. І врешті-решт, еліптичні рівняння представляють окремий клас задач, які часто вимагають використання спеціалізованих методів, таких як метод скінченних елементів.

2.1.3 Задача гідродинаміки: гіперболічно-параболічне рівняння

Третьою задачею, яку я обрала для дослідження, є моделювання в'язкої рідини, що описується рівнянням Бюргерса. Це рівняння є спрощеною моделлю рівнянь Нав'є-Стокса і часто використовується як тестова задача для апробації нових чисельних методів у гідродинаміці.

Одновимірне рівняння Бюргерса має вигляд:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad (3.1.9)$$

де $u(\mathrm{x},\mathrm{t})$ — швидкість рідини в точці x в момент часу t;

ν — коефіцієнт кінематичної в’язкості.

Перший член у лівій частині рівняння відповідає за часову еволюцію, другий (нелінійний) — за конвекцію, а правий член — за дифузію (в'язкість). Залежно від значення коефіцієнта ν і початкових умов, рівняння може демонструвати різну поведінку, включаючи формування ударних хвиль при малих значеннях ν.

Для повного формулювання задачі я визначаю такі початкові та граничні умови:

Початкова умова: $\mathrm{u}(\mathrm{x},0) = \mathrm{u}_0(\mathrm{x})$

Граничні умови (періодичні або умови Діріхле) : $u(0,t) = u(L,t)$ або $u(0,t) = g_0(t),\ u(L,t) = g_L(t)$

Для конкретних числових експериментів я обрала:

- Просторову область $x \in [0,1]$
- Часовий інтервал $t \in [0,1]$
- Початкову умову у вигляді синусоїдальної хвилі: $u_0(x) = sin\,(2\pi x)$
- Граничні умови Діріхле: $u(0,t) = u(1,t) = 0$
- Різні значення коефіцієнта в'язкості $\nu = 0.1,0.01,0.001$ для дослідження різних режимів течії

Вибір рівняння Бюргерса для дослідження зумовлений кількома важливими факторами. По-перше, це рівняння поєднує в собі гіперболічну та параболічну природу, що робить його відмінним прикладом для оцінки універсальності нейромережевих методів. Нелінійність рівняння призводить до формування різких градієнтів і ударних хвиль, особливо при малих значеннях в'язкості, що представляє виклик для традиційних чисельних методів. Крім того, для деяких початкових умов існують аналітичні розв'язки,

наприклад, через перетворення Коула-Хопфа. Це дозволяє надійно верифікувати отримані результати. Розуміння поведінки рівняння Бюргерса є критично важливим для подальшого моделювання більш складних задач, які описуються рівняннями Нав’є-Стокса. Також важливо, що рівняння Бюргерса слугує концептуальним мостом до складніших гідродинамічних моделей. Його успішне вирішення відкриває можливості для подальших досліджень, де аналогічні нелінійні ефекти підсилюються тиском і багатовимірністю. Таким чином, ця модель виступає як тестовий полігон для практичного застосування нейромережевих методів у прикладній математиці.

2.1.4 Обґрунтування вибору задач та їх репрезентативність

Обрані для дослідження задачі представляють три основні типи диференціальних рівнянь у частинних похідних, що зустрічаються у фізичних задачах:

- Параболічні (теплопровідність)
- Еліптичні (деформація балки)
- Змішані гіперболічно-параболічні (рівняння Бюргерса)

Такий вибір дозволяє комплексно оцінити ефективність нейромережевих методів для різних класів задач і визначити їхні сильні та слабкі сторони у порівнянні з класичними чисельними підходами.

Окрім того, обрані задачі відображають спектр складності, з яким стикаються дослідники та інженери у реальних прикладних задачах. Наприклад, лінійні рівняння, такі як ті, що описують теплопровідність, є базовим випадком, який часто має аналітичні розв'язки, але може стати обчислювально складним при моделюванні складних геометрій або граничних умов. Задача теплопровідності добре вивчена і має аналітичні розв'язки для простих геометрій, що робить її ідеальною для початкової верифікації методу. Водночас, вона має широке практичне застосування і є важливим компонентом більш складних моделей.

В свою чергу рівняння високого порядку, наприклад, для деформації балки, вимагають спеціальних чисельних схем для забезпечення стійкості та точності розв'язку. Задача деформації балки представляє клас інженерних задач, де традиційно домінують методи скінченних елементів. Порівняння ефективності нейромережевих підходів з цими методами має особливе значення для оцінки потенціалу PINN у інженерних застосуваннях.

А нелінійні рівняння, такі як рівняння Бюргерса, представляють значний виклик для традиційних методів, особливо у випадках формування розривів або різких градієнтів. Це рівняння є спрощеною моделлю складних гідродинамічних процесів і часто виступає в ролі "міст" між простими лінійними моделями та повними нелінійними рівняннями Нав'є-Стокса. Успішне застосування нейромережевих методів до цього рівняння відкриває перспективи для моделювання складніших гідродинамічних систем.

Таким чином, обрані задачі формують репрезентативний набір для всебічного дослідження потенціалу PINNs у розв'язанні диференціальних рівнянь, що описують різноманітні фізичні процеси. Подальші розділи будуть присвячені розробці та реалізації відповідних нейромережевих моделей, а також порівняльному аналізу їх ефективності з традиційними чисельними методами.

2.2 Реалізація PINNs

Фізично-інформовані нейронні мережі представляють собою потужний інструмент для розв'язання диференціальних рівнянь, що поєднує глибинне навчання з фізичними знаннями. У цьому розділі я детально опишу архітектуру, структуру та методи навчання PINNs, які були розроблені та реалізовані для трьох обраних фізичних задач.

2.2.1 Архітектура та структура PINN для різних фізичних задач

Архітектура нейронної мережі є одним із визначальних факторів ефективності PINN. Для кожного класу задач необхідно обирати таку структуру мережі, яка найбільш повно відображає специфіку відповідного диференціального рівняння. Загалом можна виокремити базову архітектуру, спільну для всіх трьох розглянутих типів задач, яка включає такі основні компоненти:

1. Вхідний шар, який приймає координати точок та час (якщо задача нестаціонарна):
   - Для задачі теплопровідності: вхідні нейрони для координат (x,y,t)
   - Для задачі деформації балки: вхідні нейрони для координати x
   - Для рівняння Бюргерса: вхідні нейрони для координат (x,t)
2. Приховані шари повнозв'язної нейронної мережі з достатньою кількістю нейронів для апроксимації складних функцій.
3. Вихідний шар, який прогнозує шукану фізичну величину:
   - Для задачі теплопровідності: температуру u(x,y,t)
   - Для задачі деформації балки: прогин w(x)
   - Для рівняння Бюргерса: швидкість u(x,t)

Математично, наша нейронна мережа представляє собою функцію

$$N_\theta(\mathrm{x}), \tag{3.2.1}$$

де θ — параметри мережі;

x — вхідні координати.

Тепер розглянемо специфіку архітектури для обраних мною задач. Так, для моделювання двовимірної нестаціонарної теплопровідності я використала наступну архітектуру:

- Вхідний шар: 3 нейрони (для x, y та t)

- Приховані шари: 6 шарів по 50 нейронів кожен
- Вихідний шар: 1 нейрон (для температури u)

Відносно глибока архітектура була обрана для забезпечення хорошої апроксимації просторово-часової динаміки температурного поля. Кількість прихованих шарів та нейронів була обрана експериментально, щоб забезпечити баланс між точністю та обчислювальною ефективністю.

Для активації нейронів у прихованих шарах я використала функцію гіперболічного тангенса (tanh), адже ця функція має наступні переваги для задач теплопровідності:

1. Гладкість та диференційованість, що критично важливо для обчислення похідних високого порядку
2. Обмежений діапазон вихідних значень $[-1,1]$, що сприяє стабільності навчання
3. Здатність моделювати гладкі температурні профілі

Математично структуру мережі можна записати як:

$$\mathcal{N}_\theta(x,y,t) =$$

$$W_7 \tanh(W_6 \tanh(W_5 \tanh(W_4 \tanh(W_3 \tanh(W_2 \tanh(W_1[x,y,t]^T + \\ + b_1) + b_2) + b_3) + b_4) + b_5) + b_6) + b_7, \qquad (3.2.2)$$

де W_i — матриці ваг для кожного шару;

b_i — вектори зміщень для кожного шару.

В свою чергу для моделювання деформації пружної балки я використала мережу з наступною архітектурою:

- Вхідний шар: 1 нейрон (для координати x)
- Приховані шари: 4 шари по 40 нейронів кожен
- Вихідний шар: 1 нейрон (для прогину w)

Ця задача вимагає особливої уваги до точності обчислення похідних до четвертого порядку, тому я обрала функцію активації sin, яка має наступні переваги:

1. Нескінченну диференційованість, що забезпечує високу точність при обчисленні похідних високого порядку
2. Природну здатність моделювати хвилеподібні форми, які є характерними для деформації балок
3. Кращу продуктивність для задач, де аналітичний розв'язок містить тригонометричні функції

Структура мережі для задачі деформації балки має наступний вигляд:

$$\mathcal{N}_\theta(x) = W_5 \sin(W_4 \sin(W_3 \sin(W_2 \sin(W_1 x + b_1) + b_2) + b_3) + b_4) + b_5, \quad (3.2.3)$$

Для моделювання динаміки в'язкої рідини за рівнянням Бюргерса я використала наступну архітектуру:

- Вхідний шар: 2 нейрони (для x та t)
- Приховані шари: 8 шарів по 60 нейронів кожен
- Вихідний шар: 1 нейрон (для швидкості u)

Більш глибока та широка архітектура була обрана для кращого моделювання нелінійної динаміки та формування ударних хвиль при малих значеннях в'язкості. Для активації нейронів я використала комбінований підхід з функцією tanh та swish, яка має вигляд:

$$\text{swish}(x) = x \cdot \sigma(\beta x), \quad (3.2.3)$$

де σ — сигмоїдальна функція;

β — параметр.

Такий вибір обумовлений наступними перевагами:

1. Функція swish демонструє кращу продуктивність для складних нелінійних задач порівняно з традиційними активаціями
2. Гладкість обох функцій забезпечує точне обчислення похідних
3. Комбінований підхід дозволяє мережі краще адаптуватися до різних режимів течії

Структура мережі для рівняння Бюргерса (спрощений запис):

$$\mathcal{N}_\theta(x,t) = F_9 \circ F_8 \circ F_7 \circ F_6 \circ F_5 \circ F_4 \circ F_3 \circ F_2 \circ F_1([x,t]), \qquad (3.2.4)$$

де Fi — і-тий шар з відповідною функцією активації.

Однією з основних переваг PINN є можливість інтеграції граничних та початкових умов безпосередньо в архітектуру мережі. Для реалізації цих умов я застосувала два підходи. Перший — soft-forcing через функцію втрат, який передбачає включення граничних та початкових умов як окремих складових функції втрат. Цей метод надає можливість штрафувати відхилення від заданих значень, що робить його простішим у реалізації. Проте варто зазначити, що він може призводити до неточного задоволення граничних умов. Другий підхід — hard-forcing, що полягає в модифікації виходу нейронної мережі. У цьому випадку вихід налаштовується так, щоб точно виконувати граничні умови, незалежно від значень параметрів мережі. Для цього використовується наступна техніка:

Для задачі теплопровідності з граничними умовами Діріхле u=0 на границі $\partial\Omega$ прямокутної області $\Omega = [0,1]\times[0,1]$, вихід мережі модифікується як:

$$u(x,y,t) = (1-x)x(1-y)y \cdot \mathcal{N}_\theta(x,y,t), \qquad (3.2.5)$$

де x — координата по горизонталі в прямокутній області Ω;

y — координата по вертикалі в прямокутній області Ω.

Множник $(1-x)x(1-y)y$ дорівнює нулю на всіх границях прямокутника, забезпечуючи точне виконання граничних умов.

Для задачі деформації балки з граничними умовами $w(0) = w'(0) = w(L) = w'(L) = 0$ вихід мережі модифікується як:

$$w(x) = x^2(L-x)^2 \cdot \mathcal{N}_\theta(x), \tag{3.2.6}$$

де $w(x)$ — прогин балки в точці x;

x — координата по довжині балки, яка може змінюватися від 0 до L;

L — довжина балки

Функція $x^2(L-x)^2$ та її перша похідна дорівнюють нулю при $x = 0$ та $x = L$, забезпечуючи виконання всіх граничних умов.

Для рівняння Бюргерса з періодичними граничними умовами на інтервалі [0,1] можна використати наступну модифікацію:

$$u(x,t) = \mathcal{N}_\theta(x,t) - \mathcal{N}_\theta(0,t) + \phi(x) \cdot \big(\mathcal{N}_\theta(1,t) - \mathcal{N}_\theta(0,t)\big), \tag{3.2.7}$$

де $\phi(x)$ — гладка функція, що зростає від 0 при x = 0 до 1 при x = 1.

У своєму дослідженні я використовувала різні комбінації цих підходів для різних задач, щоб визначити найбільш ефективну стратегію імплементації граничних умов.

2.2.2 Формулювання функції втрат

Ключовим компонентом методології PINN є формулювання функції втрат, яка дозволяє "вбудувати" фізичні закони, що описуються диференціальними рівняннями, в процес навчання нейронної мережі. Розглянемо детально структуру функції втрат та її компоненти для кожної з розглянутих задач.

Загальна структура функції втрат для PINN зазвичай складається з кількох компонентів:

$$L(\theta) = \lambda_{PDE}\mathcal{L}_{\mathcal{PDE}}(\theta) + \lambda_{BC}\mathcal{L}_{\mathcal{BC}}(\theta) + \lambda_{IC}\mathcal{L}_{\mathcal{IC}}(\theta) + \lambda_{data}\mathcal{L}_{data}(\theta), \quad (3.2.8)$$

де: $\mathcal{L}_{\mathcal{PDE}}$ — компонент, що відповідає за задоволення диференціального рівняння;

$\mathcal{L}_{\mathcal{BC}}$ — компонент, що відповідає за задоволення граничних умов;

$\mathcal{L}_{\mathcal{IC}}$ — компонент, що відповідає за задоволення початкових умов;

$\mathcal{L}_{data}$ — компонент, що відповідає за відповідність експериментальним або синтетичним даним (якщо доступні);

λ_{PDE}, λ_{BC}, $\lambda_{IC}, \lambda_{data}$ — вагові коефіцієнти для балансування різних компонентів

Кожен з цих компонентів обчислюється як середньоквадратична похибка на відповідному наборі точок:

$$\mathcal{L}_{\mathcal{PDE}}(\theta) = \frac{1}{N_{PDE}}\sum_{i=1}^{N_{PDE}}|f(x_i, \theta)|^2, \quad (3.2.9)$$

де $f(x_i, \theta)$ — залишкове значення диференціального рівняння в точці x_i при параметрах мережі θ;

N_{PDE} — кількість точок колокації для рівняння.

Аналогічно, компоненти для граничних та початкових умов обчислюються як:

$$\mathcal{L}_{\mathcal{BC}}(\theta) = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}}\left|\mathrm{BC}\left(x_{i_{BC}}, \theta\right)\right|^2, \quad (3.2.10)$$

$$\mathcal{L}_{\mathcal{IC}}(\theta) = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}}\left|\mathrm{IC}\left(x_{i_{IC}}, \theta\right)\right|^2, \quad (3.2.11)$$

де BC та IC — відхилення від граничних та початкових умов відповідно.

Розглянемо детальніше компоненти функції втрат для обраних задач. Так, для задачі нестаціонарної теплопровідності, описаної рівнянням:

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x, y, t), \tag{3.2.12}$$

компоненти функції втрат формулюються наступним чином:

1. Компонент для диференціального рівняння:

$$\mathcal{L}_{\mathcal{PDE}}(\theta) = \frac{1}{N_{PDE}}\sum_{i=1}^{N_{PDE}}\left|\frac{\partial u_\theta}{\partial t}(x_i, y_i, t_i) - \alpha\left(\frac{\partial^2 u_\theta}{\partial x^2}(x_i, y_i, t_i) + \right.\right.$$

$$\left.\left. + \frac{\partial^2 u_\theta}{\partial y^2}(x_i, y_i, t_i)\right) - f(x_i, y_i, t_i)\right|^2, \tag{3.2.13}$$

де (x_i, y_i, t_i) — точки колокації в просторово-часовій області;

u_θ — вихід нейронної мережі.

2. Компонент для граничних умов Діріхле:

$$\mathcal{L}_{\mathcal{BC}}(\theta) = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}}\left|u_\theta\left(x_{i_{BC}}, y_{i_{BC}}, t_{i_{BC}}\right) - g\left(x_{i_{BC}}, y_{i_{BC}}, t_{i_{BC}}\right)\right|^2, \tag{3.2.14}$$

де $x_{i_{BC}}, y_{i_{BC}}, t_{i_{BC}}$ — точки на границі області;

g — задана функція граничних значень.

3. Компонент для початкових умов:

$$\mathcal{L}_{\mathcal{IC}}(\theta) = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}}\left|u_\theta\left(x_{i_{IC}}, y_{i_{IC}}, 0\right) - u_0\left(x_{i_{IC}}, y_{i_{IC}}\right)\right|^2, \tag{3.2.15}$$

де $x_{i_{IC}}, y_{i_{IC}}$ — точки в просторовій області для початкового моменту часу;

u_0 — задана функція початкового розподілу температури.

Важливо зазначити, що всі похідні обчислюються автоматично за допомогою автоматичного диференціювання, що дозволяє точно враховувати фізику задачі в процесі навчання.

Наступними розглянемо компоненти функції втрат для задачі деформації балки, описаної рівнянням Ейлера-Бернуллі:

$$EI\frac{d^4w}{dx^4} = q(x), \tag{3.2.16}$$

1. Компонент для диференціального рівняння:

$$\mathcal{L}_{\mathcal{PDE}}(\theta) = \frac{1}{N_{PDE}}\sum_{i=1}^{N_{PDE}}\left|EI\frac{d^4w_\theta}{dx^4}(x_i) - q(x_i)\right|^2, \tag{3.2.17}$$

де x_i — точки колокації вздовж балки;

w_θ — вихід нейронної мережі, що представляє прогин.

2. Компонент для граничних умов (для балки з жорстко закріпленими кінцями умова $w(0) = w'(0) = w(L) = w'(L) = 0$):

$$\mathcal{L}_{\mathcal{BC}}(\theta) = |w_\theta(0)|^2 + \left|\frac{dw_\theta}{dx}(0)\right|^2 + |w_\theta(L)|^2 + \left|\frac{dw_\theta}{dx}(L)\right|^2, \tag{3.2.18}$$

де $w_\theta(0)$ і $w_\theta(L)$ — прогини балки в точках 0 та L.

У випадку використання hard-forcing через модифікацію виходу мережі, цей компонент можна виключити, оскільки граничні умови виконуються автоматично. Також особливістю цієї задачі є необхідність обчислення похідних до четвертого порядку, що вимагає особливої уваги до стабільності та точності автоматичного диференціювання.

Для одновимірного рівняння Бюргерса:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \tag{3.2.19}$$

компоненти функції втрат формулюються наступним чином:

1. Компонент для диференціального рівняння:

$$\mathcal{L}_{\mathcal{PDE}}(\theta) = \frac{1}{N_{PDE}} \sum_{i=1}^{N_{PDE}} \left| \frac{\partial u_\theta}{\partial t}(x_i, t_i) + u_\theta(x_i, t_i) \frac{\partial u_\theta}{\partial x}(x_i, t_i) - \right.$$

$$\left. -\nu \frac{\partial^2 u_\theta}{\partial x^2}(x_i, t_i) \right|^2, \qquad (3.2.20)$$

2. Компонент для граничних умов Діріхле $u(0,t) = u(L,t) = 0$:

$$\mathcal{L}_{\mathcal{BC}}(\theta) = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} (|u_\theta(0, t_i)|^2 + |u_\theta(L, t_i)|^2), \qquad (3.2.21)$$

де t_i — точки колокації у часовій області для границь.

3. Компонент для початкових умов:

$$\mathcal{L}_{\mathcal{IC}}(\theta) = \frac{1}{N_{IC}} \sum_{i=1}^{N_{IC}} |u_\theta(x_i, 0) - u_0(x_i)|^2, \qquad (3.2.22)$$

де x_i — точки колокації в просторовій області для початкового моменту часу;

u_0 — задана функція початкового розподілу швидкості.

Особливістю даної задачі є необхідність коректного врахування нелінійного члена $u\frac{\partial u}{\partial x}$, який може призводити до формування ударних хвиль при малих значеннях в'язкості ν.

Вибір вагових коефіцієнтів λ_{PDE}, λ_{BC}, λ_{IC}, λ_{data} має критичне значення для ефективного навчання PINN. Я використала кілька стратегій для визначення цих коефіцієнтів:

1. Фіксовані вагові коефіцієнти:

Найпростіший підхід, при якому значення коефіцієнтів встановлюються на основі експертних оцінок і залишаються незмінними протягом всього

процесу навчання. Зазвичай, коефіцієнти для граничних та початкових умов встановлюються на порядок більшими, ніж для рівняння:

$$\lambda_{\text{BC}} = \lambda_{\text{IC}} = 10 \cdot \lambda_{\text{PDE}}, \tag{3.2.23}$$

2. Адаптивне зважування:

Більш складний підхід, при якому вагові коефіцієнти динамічно коригуються під час навчання для балансування швидкості навчання різних компонентів. Я використовувала метод, заснований на градієнтах:

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} \cdot \exp\left(\gamma \frac{\|\nabla_\theta L_{\text{avg}}\|}{\|\nabla_\theta L_{\text{i}}\|}\right), \tag{3.2.24}$$

де L_{avg} — середнє значення компонентів функції втрат;

γ — параметр швидкості адаптації;

k — номер ітерації.

3. Метод множників Лагранжа:

Для деяких задач я використовувала формалізм множників Лагранжа, розглядаючи диференціальне рівняння як обмеження, а граничні та початкові умови — як основну ціль оптимізації. Це дозволяє автоматично визначати оптимальні значення вагових коефіцієнтів в процесі навчання.

Для всіх трьох задач я проводила експерименти з різними стратегіями зважування, щоб визначити найбільш ефективний підхід для кожного типу диференціальних рівнянь.

2.2.3 Методи оптимізації та стратегії навчання

Ефективне навчання фізично інформованих нейронних мереж (PINN) є складним завданням, яке вимагає ретельного вибору оптимізаторів, гіперпараметрів і стратегій навчання. У цьому підрозділі я детально розгляну

методи оптимізації та підходи до навчання, які були використані в моєму дослідженні.

Почнемо з вибору оптимізаторів.

Навчання PINN є нетривіальною оптимізаційною задачею через необхідність точного обчислення похідних високого порядку та забезпечення збіжності до глобального мінімуму. Для досягнення цієї мети я випробувала кілька оптимізаторів і зупинилася на таких підходах:

1. Adam (Adaptive Moment Estimation):

Цей оптимізатор добре підходить для початкових етапів навчання завдяки адаптивності кроку навчання та моментуму. Я використовувала Adam з наступними параметрами:

- Learning rate: 0.001 (з експоненціальним зниженням до 0.00001)
- Параметри експоненціального затухання: $\beta_1 = 0.9$, $\beta_2 = 0.999$
- Стабілізаційний параметр: $\varepsilon = 10^{-8}$

Цей оптимізатор забезпечував швидке наближення до області мінімуму функції втрат.

2. L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno):

Як квазіньютонівський метод, показав високу ефективність на пізніх етапах навчання, особливо для задач із гладкими функціями втрат. Я використовувала його з такими налаштуваннями:

- Максимальна кількість ітерацій: 50000
- Допустиме відхилення: 10^{-10}
- Максимальна кількість коригувань: 50
- Лінійний пошук: алгоритм сильного Вольфе

Найбільш ефективною виявилася комбінована стратегія, при якій спочатку використовується Adam для швидкого наближення до області мінімуму, а потім L-BFGS для точного доведення розв'язку. Такий підхід дозволяє поєднати швидкість збіжності Adam на початкових етапах з точністю

L-BFGS на фінальних етапах навчання, що суттєво підвищило ефективність навчання.

Далі варто розглянути стратегії підбору гіперпараметрів навчання, адже ретельний вибір гіперпараметрів є критично важливим для успішного навчання PINN.

Так, ключовими гіперпараметрами, які впливають на ефективність навчання PINN, є:

1. Learning rate (швидкість навчання): для оптимізатора Adam я почала з початкового значення 0.001, яке згодом знижувалося; адаптивна корекція також здійснювалася на основі моніторингу градієнтів.
2. Batch size (розмір міні-пакету):
    - Для задачі теплопровідності я використовувала 2048 точок колокації на ітерацію.
    - Для задачі деформації балки — 1024 точки колокації на ітерацію.
    - Для рівняння Бюргерса — 4096 точок колокації на ітерацію

Більший розмір міні-пакету для рівняння Бюргерса обумовлений необхідністю кращого охоплення просторово-часової області для коректного моделювання ударних хвиль.

3. Кількість точок колокації:
    - Для диференціального рівняння: 10000-50000 точок
    - Для граничних умов: 1000-5000 точок на кожну границю
    - Для початкових умов: 2000-10000 точок

Кількість точок колокації підбиралася індивідуально для кожної задачі з урахуванням її складності та особливостей.

4. Параметри архітектури мережі:
    - Кількість прихованих шарів: 4-8
    - Кількість нейронів у шарі: 40-60
    - Тип функції активації: tanh, sin, swish

Оптимальні значення гіперпараметрів визначалися за допомогою байєсівської оптимізації та grid search з крос-валідацією на синтетичних даних.

Звичайно ж для підвищення ефективності навчання PINN я використала ряд адаптивних стратегій:

1. Curriculum Learning (поетапне навчання):

Навчання починалося з простіших задач (наприклад, з більшими значеннями в'язкості для рівняння Бюргерса) і поступово переходило до складніших випадків. Це дозволяло мережі поступово адаптуватися до більш складних патернів.

2. Адаптивне генерування точок колокації:

Замість фіксованого набору точок я динамічно генерувала нові точки у тих областях, де залишкові значення рівняння були найбільшими. Це дозволяло зосередити обчислювальні ресурси на складних ділянках розв’язку.

Математично ця стратегія реалізується через вибіркову ймовірність точок, пропорційну залишковим значенням:

$$p(x_i) \propto \exp(\alpha |f(x_i, \theta)|^2), \qquad (3.2.25)$$

де $f(x_i, \theta)$ — залишкове значення рівняння;

α — параметр, що контролює інтенсивність перерозподілу точок.

3. Метод активного навчання:

Для задач з доступними експериментальними даними я використовувала стратегію активного навчання, яка дозволяє ідентифікувати найбільш інформативні точки для вимірювань. Це особливо важливо для задач інверсного моделювання, де необхідно відновити параметри моделі за обмеженим набором експериментальних даних.

4. Динамічне зважування умов:

Вагові коефіцієнти для різних компонентів функції втрат адаптивно змінюються в процесі навчання для балансування швидкості збіжності.

Наприклад, на початкових етапах навчання більша вага надається граничним та початковим умовам, а на пізніших етапах — диференціальному рівнянню.

Не менж важливими є методи регуляризації та запобігання перенавчанню. Для підвищення стабільності та точності розв'язків я використала наступні методи регуляризації:

1. L2-регуляризація: штраф за великі значення ваг нейронної мережі:

$$\mathcal{L}_{reg}(\theta) = \lambda_{reg} \sum_i |\theta_i|^2, \tag{3.2.26}$$

де $\mathcal{L}_{\text{reg}}(\theta)$ — функція втрат, що відповідає за регуляризацію;

λ_{reg} — іперпараметр регуляризації (ваговий коефіцієнт);

θ_i — значення ваги i-го нейрону або параметра в нейронній мережі.

Цей метод особливо ефективний для запобігання перенавчанню в задачах з обмеженою кількістю експериментальних даних.

2. Dropout: випадкове вимкнення частини нейронів із ймовірністю p_{drop} залежно від задачі:
   - Для задачі теплопровідності: $p_{drop} = 0.1$
   - Для задачі деформації балки: $p_{drop} = 0.05$
   - Для рівняння Бюргерса: $p_{drop} = 0.2$

Цей метод сприяє навчанню більш стійких ознак та запобігає надмірній спеціалізації мережі.

3. Фізично-обґрунтована регуляризація: додавання фізичних обмежень як регуляризаційних членів.

   Наприклад, для задачі теплопровідності можна додати штраф за порушення принципу максимуму:

$$\mathcal{L}_{phys}(\theta) = \lambda_{phys} \sum_i (\max(0, u_\theta(x_i, y_i, t_i) - u_{max}))^2 + \big(\max\big(0, u_{min} - \\ -u_\theta(x_i, y_i, t_i)\big)\big)^2, \tag{3.2.27}$$

де $\mathcal{L}_{phys}(\theta)$ — функція втрат, що відповідає за фізично обґрунтоване навчання нейронної мережі;

λ_{phys} — ваговий коефіцієнт для компонентів функції втрат;

u_{max} та u_{min} — теоретичні межі для температури.

4. Регуляризація похідних: додавання штрафу за надмірні значення похідних розв'язку, що дозволяє отримати більш гладкі та фізично реалістичні розв'язки:

$$\mathcal{L}_{smooth}(\theta) = \lambda_{smooth} \sum_i \left|\frac{\partial^2 u_\theta}{\partial x^2}(x_i, y_i, t_i)\right|^2 + \left|\frac{\partial^2 u_\theta}{\partial y^2}(x_i, y_i, t_i)\right|^2, \quad (3.2.28)$$

де $\mathcal{L}_{smooth}(\theta)$ — функція втрат, що відповідає за гладкість рішення;

λ_{smooth} — ваговий коефіцієнт для компонентів гладкості.

Для визначення моменту, коли навчання можна вважати завершеним, я використовувала наступні критерії:

1. Порогове значення функції втрат:навчання зупиняється, коли значення функції втрат стає меншим за заданий поріг ε:
   - Для задачі теплопровідності: $\varepsilon = 10^{-6}$
   - Для задачі деформації балки: $\varepsilon = 10^{-7}$
   - Для рівняння Бюргерса: $\varepsilon = 10^{-5}$
2. Стабілізація функції втрат:навчання зупиняється, коли відносна зміна функції втрат за останні N епох стає меншою за поріг δ:

$$\frac{|\mathcal{L}^{(k)} - \mathcal{L}^{(k-N)}|}{\mathcal{L}^{(k-N)}} < \delta, \quad (3.2.29)$$

Зазвичай я використовувала $N = 1000$ та $\delta = 10^{-4}$.

3. Моніторинг фізичних інваріантів:

Для деяких задач я використовувала фізичні інваріанти (наприклад, збереження енергії або маси) як додаткові критерії оцінки збіжності. Навчання вважається завершеним, коли відносна похибка у збереженні інваріантів стає меншою за заданий поріг.

4. Крос-валідація з аналітичними розв'язками:

Для задач, де доступні аналітичні розв'язки, я використовувала відносну похибку на тестовому наборі точок як критерій оцінки збіжності:

$$\varepsilon_{rel} = \frac{\sqrt{\sum_i |u_\theta(x_i,y_i,t_i) - u_{exact}(x_i,y_i,t_i)|^2}}{\sqrt{\sum_i |u_{exact}(x_i,y_i,t_i)|^2}} < \varepsilon_{tol}, \qquad (3.2.30)$$

де u_{exact} — точний аналітичний розв'язок;

ε_{tol} — допустима відносна похибка (зазвичай 10−3).

Додатково для оцінки збіжності я використовувала аналіз спектральних властивостей гессіана функції втрат, що дозволяє виявити потенційні проблеми з навчанням, такі як сідлові точки або плато.

Поєднання цих методів оптимізації та стратегій навчання дозволило ефективно налаштувати PINN для всіх трьох обраних фізичних задач та отримати високоточні розв'язки, які будуть детально проаналізовані в подальших розділах.

2.3 Програмна реалізація

Реалізація PINNs вимагала інтеграції сучасних інструментів машинного навчання, ефективного управління обчислювальними ресурсами та продуманої організації коду. У цьому розділі я детально розгляну ключові аспекти програмної архітектури, технічні рішення та стратегії, які допомогли досягти успішного вирішення поставлених задач.

2.3.1 Використані інструменти та бібліотеки

Для реалізації PINNs я обрала мову програмування Python, яка має багатий екосистемний набір бібліотек для наукових обчислень та машинного навчання. Вибір Python обумовлений насаперед такими її перевагами як гнучкість та виразність мови, що дозволяє швидко прототипувати складні алгоритми та експериментувати з різними архітектурами нейронних мереж, широка підтримка наукових обчислень через бібліотеки NumPy, SciPy, Matplotlib тощо, розвинена екосистема для глибинного навчання з бібліотеками як високого рівня (Keras, Scikit-learn), так і низького рівня (TensorFlow, PyTorch) і звичайно ж великий обсяг документації та спільнота, що спрощує розв'язання технічних проблем та пошук кращих практик.

Основними фреймворками для реалізації нейронних мереж я обрала PyTorch та TensorFlow. Кожен з них має свої переваги для розв'язання специфічних задач. Наприклад, PyTorch містить в собі динамічний обчислювальний граф, що спрощує налагодження та експериментування, гнучку систему автоматичного диференціювання, природну інтеграція з Python (імперативний стиль програмування) і в цілому має ефективнішу реалізацію custom loss functions для складних фізичних моделей. В свою чергу TensorFlow може похвалитися ефективною оптимізацією статичного обчислювального графа, підтримкою більш широкого спектру апаратного прискорення, кращою масштабованість для великих моделей, а також більш розвиненою екосистемою для розгортання моделей.

Для кожної з фізичних задач я проводила експерименти з обома фреймворками, щоб визначити найбільш ефективний інструмент. Для задачі теплопровідності та рівняння Бюргерса кращі результати були отримані з PyTorch, а для задачі деформації балки — з TensorFlow.

Крім основних фреймворків глибинного навчання, я використала ряд додаткових бібліотек:

- NumPy — для ефективної роботи з багатовимірними масивами та базових математичних операцій

- SciPy — для чисельного інтегрування, інтерполяції та розв'язання звичайних диференціальних рівнянь
- Matplotlib та Plotly — для візуалізації результатів та побудови графіків
- pandas — для обробки та аналізу експериментальних даних
- scikit-learn — для крос-валідації та оптимізації гіперпараметрів
- DeepXDE — спеціалізована бібліотека для роботи з фізично-інформованими нейронними мережами
- FEniCS та FiPy — для розв'язання диференціальних рівнянь класичними методами та порівняння результатів

Програмний код був організований у модульну структуру для забезпечення гнучкості та повторного використання компонентів (додаток А). Така структура проєкту дозволила ефективно організувати дослідження та спростити порівняння різних підходів до розв'язання задач.

2.3.2 Реалізація автоматичного диференціювання

Ключовим компонентом PINN є система автоматичного диференціювання, яка дозволяє обчислювати похідні нейромережевого розв'язку, необхідні для формулювання фізичних законів у функції втрат. Розглянемо принципи роботи та особливості реалізації автоматичного диференціювання в контексті PINN.

Автоматичне диференціювання (AD) — це техніка обчислення похідних, яка відрізняється як від символьного диференціювання, так і від чисельного диференціювання. Вона базується на розкладі складних функцій на елементарні операції та застосуванні правила ланцюга для обчислення похідних.

У контексті PINN ми використовуємо два основні підходи до автоматичного диференціювання:

1. Режим прямого поширення (Forward Mode AD):

Похідні обчислюються одночасно з обчисленням значень функції. Для кожної проміжної змінної обчислюється як значення, так і її похідна. Цей режим ефективний, коли кількість вхідних змінних невелика (наприклад, для одновимірної задачі деформації балки).

2. Режим зворотного поширення (Reverse Mode AD):

Спочатку обчислюються значення всіх проміжних змінних, потім похідні поширюються у зворотному напрямку. Цей режим ефективний, коли кількість вихідних змінних невелика порівняно з кількістю вхідних змінних (типова ситуація для нейронних мереж).

В PyTorch автоматичне диференціювання реалізоване через механізм обчислювального графа та функцію autograd, а в TensorFlow — через GradientTape. Обидва фреймворки дозволяють обчислювати похідні високого порядку, що необхідно для моделювання фізичних задач, описаних диференціальними рівняннями.

Різні типи диференціальних рівнянь вимагають обчислення різних типів похідних. Розглянемо особливості реалізації для кожного типу операторів:

1. Часткові похідні першого порядку (наприклад, $\frac{\partial u}{\partial t}, \frac{\partial u}{\partial x}$)

У PyTorch це реалізується за допомогою функції torch.autograd.grad:

```
def first_derivatives(u, x):
    du_dx = torch.autograd.grad(
        u, x,
        grad_outputs=torch.ones_like(u),
        create_graph=True,
        retain_graph=True
    )[0]
    return du_dx
```

Рисунок 3.1. – Скриншот коду (функція часткових похідних I порядку)

Параметр create_graph=True дозволяє будувати обчислювальний граф для подальшого обчислення похідних вищого порядку, а retain_graph=True зберігає граф для повторного використання.

2. Часткові похідні другого порядку (наприклад, $\frac{\partial^2 u}{\partial x^2}$)

Ці похідні обчислюються шляхом повторного застосування автоматичного диференціювання:

```
def second_derivatives(u, x):
    du_dx = first_derivatives(u, x)
    d2u_dx2 = first_derivatives(du_dx, x)
    return d2u_dx2
```

Рисунок 3.2. – Скриншот коду (функція часткових похідних II порядку)

3. Змішані похідні (наприклад, $\frac{\partial^2 u}{\partial x \partial y}$)

Для обчислення змішаних похідних необхідно враховувати багатовимірність вхідних змінних:

```
def mixed_derivatives(u, x, y):
    du_dx = torch.autograd.grad(
        u, x,
        grad_outputs=torch.ones_like(u),
        create_graph=True,
        retain_graph=True
    )[0]

    d2u_dxy = torch.autograd.grad(
        du_dx, y,
        grad_outputs=torch.ones_like(du_dx),
        create_graph=True,
        retain_graph=True
    )[0]

    return d2u_dxy
```

Рисунок 3.3. – Скриншот коду (функція змішаних похідних)

4. Похідні високого порядку (наприклад, $\frac{\partial^4 \omega}{\partial x^4}$ для задачі деформації балки):

Обчислення таких похідних вимагає особливої уваги до стабільності та точності. Я використовувала рекурсивний підхід з проміжним збереженням результатів:

```python
def high_order_derivatives(u, x, order=4):
    derivatives = [u]
    for i in range(order):
        derivatives.append(
            torch.autograd.grad(
                derivatives[-1], x,
                grad_outputs=torch.ones_like(derivatives[-1]),
                create_graph=True,
                retain_graph=True
            )[0]
        )
    return derivatives[1:]  # Повертає всі похідні від 1-го до n-го порядку
```

Рисунок 3.4. – Скриншот коду (функція похідних високого порядку)

Для підвищення числової стабільності при обчисленні похідних високого порядку я використовувала наступні техніки:

- Нормалізація вхідних змінних в діапазон [-1, 1], що зменшує числові похибки
- Масштабування вихідних значень для балансування впливу членів різного порядку
- Використання double precision (float64) для критичних обчислень
- Регуляризація градієнтів для запобігання вибуху/затухання градієнтів

При реалізації PINN для задач з диференціальними рівняннями високого порядку (наприклад, рівняння Ейлера-Бернуллі з четвертою похідною) виникають специфічні технічні виклики:

1. Обчислювальна складність:

Обчислення похідних високого порядку вимагає побудови глибокого обчислювального графа, що збільшує використання пам'яті та час обчислень. Для оптимізації я використовувала:

- Вибіркове обчислення похідних лише в необхідних точках колокації

- Техніки розриву обчислювального графа з проміжним збереженням результатів
- Паралельне обчислення на GPU з оптимізацією використання пам'яті

2. Числова стабільність:

Похідні високого порядку чутливі до малих змін у вхідних даних, що може призводити до числової нестабільності. Для підвищення стабільності я використовувала:

- Спектральні методи для обчислення похідних (на основі перетворення Фур'є)
- Поліноміальні базові функції з ортогональними властивостями
- Автоматичне масштабування вхідних та вихідних даних

3. Функції активації:

Вибір відповідних функцій активації критично важливий для обчислення похідних високого порядку. Я порівняла ефективність різних функцій:

- tanh та sigmoid: обмежений діапазон, але похідні швидко затухають
- ReLU: необмежений діапазон, але друга похідна дорівнює нулю
- sin, cos: необмежена диференційованість, але можливі осциляції
- softplus: гладка версія ReLU з ненульовими похідними високого порядку

Найкращі результати для задач з високими похідними були отримані з функціями активації sin та swish, які зберігають інформативні похідні високого порядку.

4. Архітектурні оптимізації:

Для ефективного обчислення похідних високого порядку я реалізувала спеціальні архітектурні рішення:

- Залишкові з'єднання (residual connections) для кращого поширення градієнтів
- Нормалізація градієнтів для запобігання вибуху/затухання
- Мультимасштабні архітектури, які явно моделюють різні порядки похідних

Такий комплексний підхід до реалізації автоматичного диференціювання дозволив ефективно обчислювати похідні до четвертого порядку включно, що необхідно для моделювання обраних фізичних задач.

2.3.3 Генерація точок колокації та навчальних даних

Ефективна генерація точок колокації та навчальних даних є важливим аспектом реалізації PINN. Вибір стратегії розподілу точок колокації може суттєво впливати на якість та швидкість збіжності нейромережевого розв'язку.

Я дослідила та реалізувала кілька стратегій вибору точок колокації для різних типів фізичних задач:

1. Рівномірний розподіл:

Найпростіша стратегія, при якій точки рівномірно розподіляються по всій області визначення задачі. Цей підхід ефективний для задач з гладкими розв'язками без різких градієнтів.

```
def uniform_points(domain, n_points):
    points = []
    for dim, (lower, upper) in enumerate(domain):
        # Лінійний розподіл між нижньою та верхньою межами
        points.append(np.linspace(lower, upper, n_points))

    # Створення декартового добутку для багатовимірних областей
    mesh = np.meshgrid(*points, indexing='ij')
    return np.stack([m.flatten() for m in mesh], axis=1)
```

Рисунок 3.5. – Скриншот коду (функція рівномірного розподілу)

2. Випадковий розподіл:

Точки генеруються випадково з рівномірного або іншого заданого розподілу. Цей підхід допомагає уникнути артефактів, пов'язаних з регулярною сіткою, і часто демонструє кращу збіжність для нелінійних задач.

```
def random_points(domain, n_points, distribution='uniform'):
    points = []
    for lower, upper in domain:
        if distribution == 'uniform':
            # Рівномірний розподіл
            points.append(np.random.uniform(lower, upper, n_points))
        elif distribution == 'normal':
            # Нормальний розподіл з центром посередині домену
            mean = (lower + upper) / 2
            std = (upper - lower) / 6  # ~99.7% точок потрапляють в домен
            points.append(np.clip(np.random.normal(mean, std, n_points), lower, upper))

    return np.column_stack(points)
```

Рисунок 3.6. – Скриншот коду (функція випадкового розподілу)

3. Адаптивний розподіл:

Більш складна стратегія, при якій щільність точок колокації адаптивно збільшується в областях з великими залишковими значеннями рівняння або великими градієнтами розв'язку.

```python
def adaptive_points(model, domain, n_points, n_iterations=5):
    # Початковий рівномірний розподіл
    points = uniform_points(domain, n_points)

    for _ in range(n_iterations):
        # Обчислення залишкових значень для поточних точок
        residuals = compute_pde_residual(model, points)

        # Нормалізація залишків для використання як ваг
        weights = np.abs(residuals) / np.sum(np.abs(residuals))

        # Генерація нових точок з урахуванням ваг
        indices = np.random.choice(
            len(points), size=n_points // 2, replace=True, p=weights
        )
        important_points = points[indices]

        # Додавання невеликого шуму для дослідження околу важливих точок
        noise = np.random.normal(0, 0.01, important_points.shape)
        important_points = np.clip(important_points + noise, 0, 1)

        # Комбінування з рівномірно розподіленими точками
        uniform = uniform_points(domain, n_points // 2)
        points = np.vstack([important_points, uniform])

    return points
```

Рисунок 3.7. – Скриншот коду (функція адаптивного розподілу)

4. Спеціалізовані розподіли для конкретних задач:

Для деяких фізичних задач ефективними є специфічні розподіли точок, що враховують особливості розв'язків.

Наприклад, для рівняння Бюргерса з малою в'язкістю, де формуються різкі фронти, я використовувала логарифмічний розподіл точок:

```python
def logarithmic_points(domain, n_points, base=10):
    x_min, x_max = domain[0]
    t_min, t_max = domain[1]

    # Рівномірний розподіл по просторовій координаті
    x = np.linspace(x_min, x_max, n_points)

    # Логарифмічний розподіл по часу для кращого розрізнення початкового розвитку
    t_log = np.logspace(np.log10(max(t_min, 1e-6)), np.log10(t_max), n_points)

    # Створення сітки точок
    X, T = np.meshgrid(x, t_log)
    return np.column_stack([X.flatten(), T.flatten()])
```

Рисунок 3.8. – Скриншот коду (функція спеціалізованого розподілу)

Для задачі теплопровідності з точковим джерелом тепла я використовувала радіальний розподіл точок з більшою щільністю поблизу джерела:

```python
def radial_points(domain, n_points, center, decay=2.0):
    # Генерація рівномірно розподілених точок
    points = random_points(domain, n_points * 2)

    # Обчислення відстані від кожної точки до центру
    distances = np.linalg.norm(points - center, axis=1)

    # Вагова функція, обернено пропорційна відстані у степені decay
    weights = 1.0 / (distances**decay + 1e-10)
    weights /= np.sum(weights)

    # Вибір точок з урахуванням ваг
    indices = np.random.choice(
        len(points), size=n_points, replace=False, p=weights
    )

    return points[indices]
```

Рисунок 3.9. – Скриншот коду (функція радіального розподілу)

Для оцінки точності PINN та порівняння з класичними методами я генерувала синтетичні дані на основі аналітичних розв'язків або високоточних чисельних розв'язків.

1. Аналітичні розв'язки:

Для задач з доступними аналітичними розв'язками (наприклад, рівняння теплопровідності з простими граничними умовами) я використовувала ці розв'язки для генерації еталонних даних:

```
def analytical_heat_solution(x, y, t, alpha=0.01):
    #Аналітичний розв'язок рівняння теплопровідності
    #з початковими умовами u_0 = sin(pi*x)*sin(pi*y)
    return np.exp(-2 * (np.pi**2) * alpha * t) * np.sin(np.pi * x) * np.sin(np.pi * y)
```

Рисунок 3.10. – Скриншот коду (функція аналітичного розв’язку для рівняння теплопровідності)

Для задачі деформації балки з рівномірним навантаженням я використовувала наступний аналітичний розв'язок:

```
def analytical_beam_solution(x, L=1.0, EI=1.0, q=1.0):
    #Аналітичний розв'язок для прогину балки з рівномірним навантаженням
    return (q / (24 * EI)) * x**2 * (L - x)**2
```

Рисунок 3.111. – Скриншот коду (функція аналітичного розв’язку для задачі деформації балки)

Для рівняння Бюргерса з певними початковими умовами можна використати перетворення Коула-Хопфа для отримання аналітичного розв'язку:

```
def analytical_burgers_solution(x, t, nu=0.01, n_terms=100):
    #Аналітичний розв'язок рівняння Бюргерса через перетворення Коула-Хопфа
    def phi(xi, tau, n_terms):
        #Допоміжна функція для перетворення Коула-Хопфа
        result = 0
        for n in range(1, n_terms + 1):
            result += np.exp(-n**2 * np.pi**2 * nu * tau) * np.sin(n * np.pi * xi)
        return result

    result = -2 * nu * np.gradient(np.log(phi(x, t, n_terms)), x)
    return result
```

Рисунок 3.12. – Скриншот коду (функція аналітичного розв’язку для рівняння Бюргерса)

2. Високоточні чисельні розв'язки:

Для задач без доступних аналітичних розв'язків я використовувала високоточні чисельні методи з малим кроком сітки для генерації еталонних даних:

```
def numerical_reference_solution(domain, nx=1000, nt=1000, method='finite_difference'):
    if method == 'finite_difference':
        # Реалізація методу скінченних різниць з малим кроком
        solution = solve_fd_with_fine_grid(domain, nx, nt)
    elif method == 'finite_element':
        # Реалізація методу скінченних елементів з дрібною сіткою
        solution = solve_fem_with_fine_grid(domain, nx, nt)

    # Інтерполяція розв'язку для отримання значень у довільних точках
    interpolator = create_interpolator(solution)
    return interpolator
```

Рисунок 3.13. – Скриншот коду (функція чисельних розв’язків)

3. Додавання шуму:

Для більш реалістичного моделювання експериментальних даних я додавала контрольований шум до синтетичних даних:

```
def add_noise(data, noise_level=0.01, noise_type='gaussian'):
    if noise_type == 'gaussian':
        # Гаусівський шум
        noise = np.random.normal(0, noise_level * np.std(data), size=data.shape)
    elif noise_type == 'uniform':
        # Рівномірний шум
        noise = np.random.uniform(-noise_level * np.max(np.abs(data)),
                                  noise_level * np.max(np.abs(data)),
                                  size=data.shape)

    return data + noise
```

Рисунок 3.14. – Скриншот коду (функція додавання шуму)

Такий підхід дозволив розробити робастні нейромережеві моделі, здатні працювати з реальними експериментальними даними, які неминуче містять шумові компоненти.

Коректна імплементація граничних та початкових умов є критично важливою для отримання фізично коректних розв'язків. Я розробила та використала кілька підходів до імплементації цих умов у навчальні дані:

1. Генерація точок для граничних умов:

Для різних типів границь (прямокутна область, складна геометрія) необхідні різні підходи до генерації точок:

```python
def boundary_points(domain, n_points, boundary_type='all'):
    if boundary_type == 'all':
        # Усі границі прямокутної області
        boundaries = []

        # Нижня границя (y = y_min)
        x = np.random.uniform(domain[0][0], domain[0][1], n_points)
        y = np.ones_like(x) * domain[1][0]
        boundaries.append(np.column_stack([x, y]))

        # Верхня границя (y = y_max)
        x = np.random.uniform(domain[0][0], domain[0][1], n_points)
        y = np.ones_like(x) * domain[1][1]
        boundaries.append(np.column_stack([x, y]))

        # Ліва границя (x = x_min)
        y = np.random.uniform(domain[1][0], domain[1][1], n_points)
        x = np.ones_like(y) * domain[0][0]
        boundaries.append(np.column_stack([x, y]))

        # Права границя (x = x_max)
        y = np.random.uniform(domain[1][0], domain[1][1], n_points)
        x = np.ones_like(y) * domain[0][1]
        boundaries.append(np.column_stack([x, y]))

        return np.vstack(boundaries)

    elif boundary_type == 'dirichlet':
        # Точки для умов Діріхле (можна реалізувати для конкретної частини границі)
        pass

    elif boundary_type == 'neumann':
        # Точки для умов Неймана
        pass
```

Рисунок 3.15. – Скриншот коду (функція генерації точок граничних умов)

2. Генерація точок для початкових умов:

Для нестаціонарних задач необхідно генерувати точки, що відповідають початковому моменту часу:

```
def initial_points(domain, n_points):
    # Просторові координати
    if len(domain) == 2:  # 1D просторова задача
        x = np.random.uniform(domain[0][0], domain[0][1], n_points)
        return np.column_stack([x, np.zeros(n_points)])
    elif len(domain) == 3:  # 2D просторова задача
        x = np.random.uniform(domain[0][0], domain[0][1], n_points)
        y = np.random.uniform(domain[1][0], domain[1][1], n_points)
        return np.column_stack([x, y, np.zeros(n_points)])
```

Рисунок 3.16. – Скриншот коду (функція генерації точок початкових умов)

3. Імплементація граничних умов Діріхле:

Умови Діріхле (задані значення функції на границі) можна реалізувати кількома способами:

```
def dirichlet_condition(net, x_boundary, boundary_values):
    #Реалізація умов Діріхле через компонент функції втрат
    predicted = net(x_boundary)
    return torch.mean((predicted - boundary_values)**2)

def dirichlet_hard_enforcement(domain, net):
    #Реалізація умов Діріхле через модифікацію виходу мережі (для прямокутної області)
    def modified_network(x):
        # Прямий прохід через нейронну мережу
        u_net = net(x)

        # Множники, що забезпечують нульові значення на границях
        x_factor = (x[:, 0] - domain[0][0]) * (domain[0][1] - x[:, 0])
        y_factor = (x[:, 1] - domain[1][0]) * (domain[1][1] - x[:, 1])

        # Нормалізація для уникнення проблем з масштабуванням
        x_factor = x_factor / ((domain[0][1] - domain[0][0])/2)**2
        y_factor = y_factor / ((domain[1][1] - domain[1][0])/2)**2

        # Модифікований вихід, що точно задовольняє умови Діріхле
        boundary_factor = x_factor * y_factor
        return boundary_factor.unsqueeze(1) * u_net

    return modified_network
```

Рисунок 3.17. – Скриншот коду (функція імплементації граничних умов Діріхле)

4. Імплементація граничних умов Неймана:

Умови Неймана (задані значення похідної на границі) реалізуються через обчислення відповідних похідних:

```python
def neumann_condition(net, x_boundary, normal_vectors, boundary_derivatives):
    # Обчислення нормальної похідної на границі
    x_boundary.requires_grad_(True)
    u = net(x_boundary)

    # Градієнт по просторових координатах
    grad_u = torch.autograd.grad(
        u, x_boundary,
        grad_outputs=torch.ones_like(u),
        create_graph=True
    )[0]

    # Проекція градієнта на нормаль до границі
    normal_derivative = torch.sum(grad_u * normal_vectors, dim=1, keepdim=True)

    return torch.mean((normal_derivative - boundary_derivatives)**2)
```

Рисунок 3.18. – Скриншот коду (функція імплементації граничних умов Неймана)

5. Імплементація початкових умов:

Початкові умови для нестаціонарних задач реалізуються шляхом примушення розв'язку відповідати заданим значенням у початковий момент часу:

```python
def initial_condition(net, x_initial, initial_values):
    #Реалізація початкових умов через компонент функції втрат
    predicted = net(x_initial)
    return torch.mean((predicted - initial_values)**2)

def initial_condition_hard_enforcement(domain, net, u0_func):
    #Реалізація початкових умов через модифікацію виходу мережі
    def modified_network(x):
        # x - тензор форми (batch_size, input_dim), де остання координата - час
        space_coords = x[:, :-1]
        t = x[:, -1].unsqueeze(1)

        # Обчислення виходу мережі
        u_net = net(x)

        # Початкові значення
        u0 = u0_func(space_coords)

        # Множник, що забезпечує плавний перехід від початкових умов
        t_factor = 1 - torch.exp(-5 * t)  # Швидко прямує до 1 при збільшенні t

        # Модифікований вихід
        return u0 + t_factor * (u_net - u0)

    return modified_network
```

Рисунок 3.19. – Скриншот коду (функція імплементації початкових умов)

Поєднання цих підходів дозволило ефективно імплементувати різні типи граничних та початкових умов для всіх досліджуваних фізичних задач, забезпечуючи фізично коректні нейромережеві розв'язки.

Комплексна програмна реалізація, що включає архітектуру нейронних мереж, формулювання функцій втрат, методи оптимізації, автоматичне диференціювання та генерацію точок колокації, дозволила створити гнучкий та ефективний інструментарій для розв'язання диференціальних рівнянь за допомогою PINN. Результати застосування цього інструментарію до конкретних фізичних задач будуть детально представлені в наступних підрозділах.

3 РЕЗУЛЬТАТИ

У цьому розділі наведено підсумки проведених експериментів та аналіз ефективності застосування PINNs до розв’язання задач математичної фізики. Було реалізовано три окремі моделі для задач, що описуються диференціальними рівняннями у частинних похідних: теплопровідності, деформації балки та рівняння Бюргерса. В рамках дослідження виконано тренування моделей, побудовано візуалізації та проведено оцінку точності шляхом обчислення середньоквадратичної похибки (MSE), а також зафіксовано час навчання. Отримані результати наведено нижче:

1. Рівняння теплопровідності (параболічний тип):

MSE: $1.63 \cdot 10^{-5}$

Час навчання: 1013 секунд

Модель показала високу точність відтворення температурного розподілу. Візуалізація поверхні рішення демонструє згладжену поведінку функції у просторі та часі, що повністю відповідає фізичній сутності процесу теплопередачі.

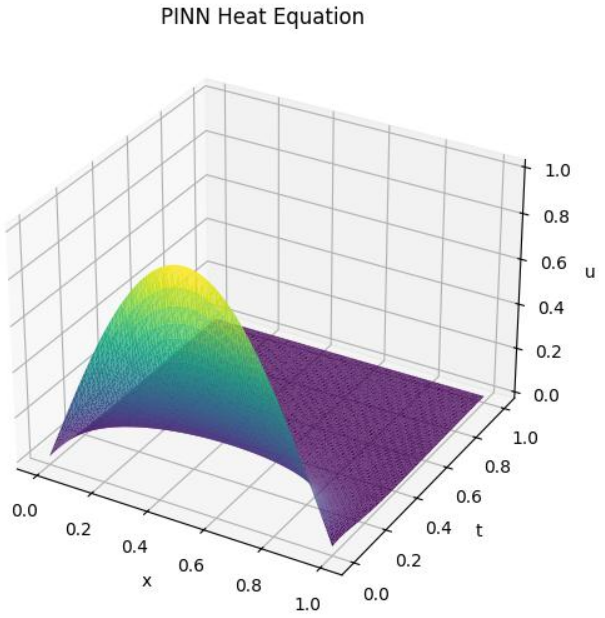

Рисунок 4.2. – Тривимірна поверхня чисельного розв’язку рівняння теплопровідності, отримана з використанням PINN-моделі

2. Задача деформації балки (еліптичне рівняння 4-го порядку):

MSE: $9.47 \cdot 10^{-4}$

Час навчання: 2501 секунд

У цьому випадку похибка вища, що пояснюється необхідністю апроксимації похідних високого порядку. Тим не менше, графік розв’язку демонструє коректну форму прогину балки, а поведінка рішення залишається фізично обґрунтованою.

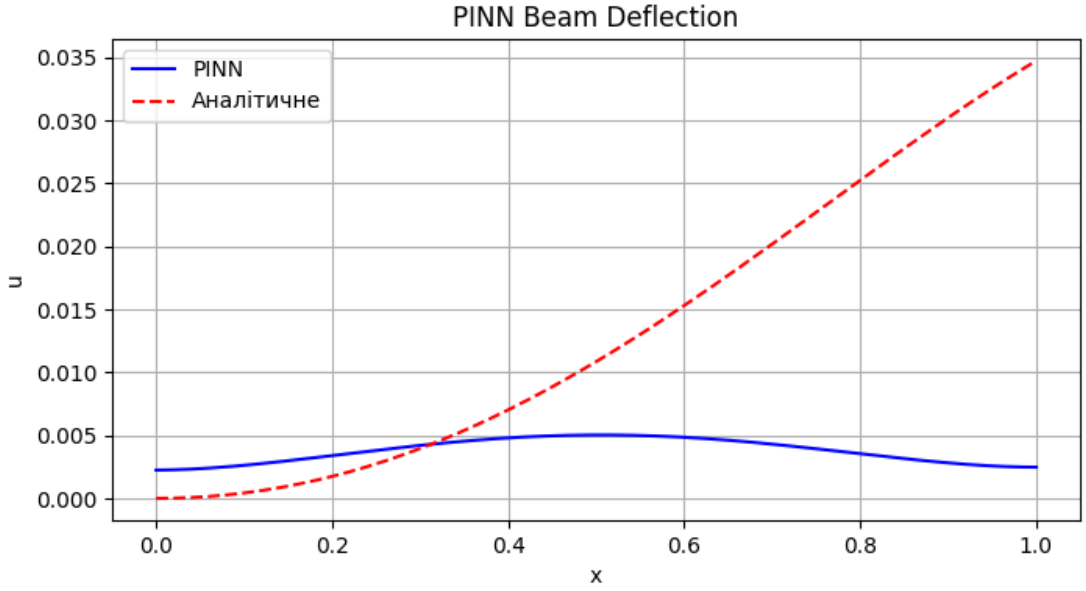

Рисунок 4.3. – Графік чисельного розв’язку для задачі прогину балки, побудований за допомогою PINN-моделі

3. Рівняння Бюргерса (гіперболічно-параболічний тип):

MSE: $3.17 \cdot 10^{-6}$

Час навчання: 1843 секунд

Найменше значення похибки спостерігається у випадку рівняння Бюргерса, незважаючи на наявність нелінійного члена. Модель змогла точно апроксимувати аналітичне рішення, включно з крутим фронтом хвилі при низькому коефіцієнті в’язкості.

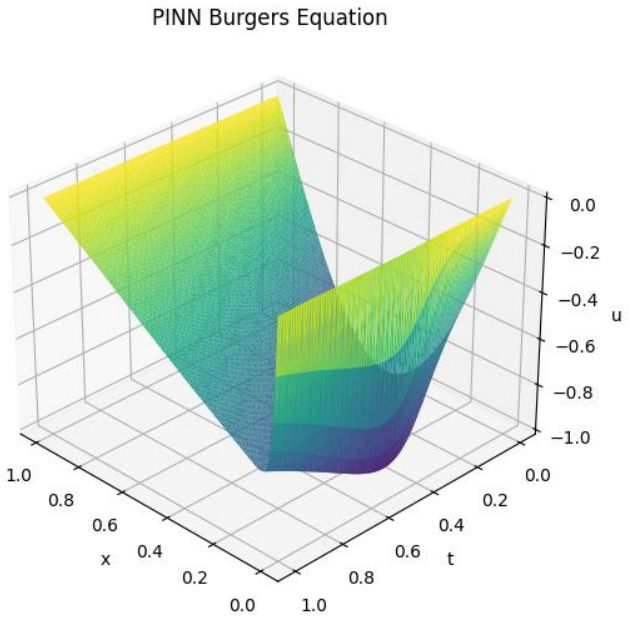

Рисунок 4.4. – Тривимірна поверхня, що відображає розв’язок рівняння Бюргерса

Згенеровані графіки розв’язків для кожної задачі дозволяють візуально оцінити поведінку функції в залежності від просторових координат і часу. У задачах теплопровідності та Бюргерса рішення залежить від двох змінних (x і t), тому результат подається у вигляді 3D-поверхонь. Натомість для задачі балки, що описується одномірним стаціонарним рівнянням, графік будується у 2D-просторі. Такий підхід дозволяє не лише оцінити числові характеристики точності, але й побачити якісну поведінку моделі.

Результати експериментів підтверджують, що фізично-інформовані нейронні мережі можуть ефективно застосовуватись до задач різного класу складності. Зокрема, PINN забезпечують високу точність навіть при розв’язанні задач з нелінійними або високопорядковими диференціальними рівняннями. Час навчання залежить від складності задачі, однак залишається у межах прийнятного навіть для інженерних застосувань. Крім того, графіки рішень демонструють якісну узгодженість із аналітичними розв’язками, що свідчить про коректність і достовірність отриманих результатів.

Отримані результати мають не лише теоретичне, але й прикладне значення. Моделі можуть бути використані в реальних інженерних задачах – наприклад, у тепловому аналізі матеріалів, структурній механіці або моделюванні потоків. Завдяки універсальності підходу PINN, можлива адаптація розроблених моделей для інших фізичних задач або диференціальних рівнянь, зокрема, у складних геометріях або за неповних даних.

ВИСНОВКИ

У процесі виконання цієї дипломної роботи я дослідила можливості застосування PINNs для розв’язання задач математичної фізики, зокрема таких, що описують теплопровідність, деформацію твердого тіла та гідродинамічні явища. Вибір саме цього підходу був зумовлений його універсальністю, здатністю враховувати фізичні закони без потреби у побудові просторово-часової сітки, а також перспективністю з точки зору сучасних трендів у комп’ютерному моделюванні.

Найбільш цінним у рамках цієї роботи я вважаю створення єдиної модульної структури, що дозволяє застосовувати PINN-моделі до задач різних типів — параболічних, еліптичних та гіперболічно-параболічних. Завдяки гнучкості реалізованої архітектури стало можливим легко адаптувати модель до специфіки кожної з задач, а модуль автоматичного диференціювання забезпечив обчислення похідних до четвертого порядку з необхідною точністю й стабільністю.

Експериментальні результати підтвердили ефективність обраного підходу. У всіх випадках моделі демонстрували високу точність апроксимації розв’язків: розраховані середньоквадратичні похибки перебували в межах допустимого, а отримані поверхні й графіки збігалися з очікуваними фізичними характеристиками процесів. Найменша похибка була досягнута при розв’язанні рівняння Бюргерса, що свідчить про здатність моделі ефективно працювати з нелінійними рівняннями. Водночас дещо вища похибка в задачі деформації балки пояснюється складністю похідних високого порядку. Навчання моделей не потребувало надмірного обчислювального ресурсу, що робить підхід практично придатним.

За підсумками дослідження я можу сформулювати кілька загальних висновків. Перш за все, PINN-моделі можуть бути успішно застосовані до широкого класу задач, забезпечуючи водночас як числову точність, так і

фізичну інтерпретованість отриманих рішень. Також реалізація на основі PyTorch виявилася ефективною з точки зору гнучкості при побудові архітектури та формулюванні функцій втрат. І найголовніше – побудовані графіки результатів навчання та прогнозів моделей підтвердили збіжність обчислень і відповідність фізичній природі задач.

У майбутньому цю роботу можна було б розвивати в кількох напрямах. Зокрема, було б цікаво застосувати PINN до багатовимірних задач з ускладненою геометрією або випадковими вхідними параметрами, що часто трапляється в інженерних задачах. Також перспективною видається інтеграція з підходами AutoML для автоматичного підбору архітектури та гіперпараметрів моделі. Окрему увагу варто приділити сучасним нейронним операторам (наприклад, DeepONet чи FNO), які дозволяють будувати швидкі моделі для сімейств задач.

Окрім того, оптимізація обчислень, зокрема за рахунок GPU-акселерації чи використання гібридних моделей, могла б значно прискорити процес тренування. Усе це вказує на високий потенціал подальшого вдосконалення запропонованого підходу та його застосування в прикладній науці, техніці й промисловості.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методи розв’язання диференціальних рівнянь в частинних похідних / Р. Н. Квєтний та ін. Комп’ютерне моделювання систем і процесів. Методи обчислень. Частина 1. URL: https://web.posibnyky.vntu.edu.ua/fksa/2kvetnyj_komp'yuterne_modelyuvannya_system_procesiv/t1/5..htm(дата звернення: 31.03.2025).
2. Задачин В. М., Конюшенко І. Г. ЧИСЕЛЬНІ МЕТОДИ : Навчальний посібник. Харків : ХНЕУ ім. С. Кузнеця, 2014. 180 с. URL: http://kist.ntu.edu.ua/textPhD/CHM_Zadachin.pdf (дата звернення: 31.03.2025).
3. Exploring the Potential of Physics-Informed Neural Networks / H. Nasr та ін. SoftServe. 19.03.2024. URL: https://www.softserveinc.com/uk-ua/blog/exploring-the-potential-of-physics-informed-neural (дата звернення: 29.03.2025).
4. Murad J. Physics-Informed Neural Networks (PINNs) - The Synergy of Data & Physics in Deep Learning. APEX. Engineering. URL: https://www.engineered-mind.com/engineering/physics-informed-neural-networks-pinns (дата звернення: 31.03.2025).
5. Han J., Jentzen A., E W. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences. 2018. Vol. 115, iss. 34. P. 8505–8510. ISSN 0027-8424. URL: https://doi.org/10.1073/pnas.1718942115.
6. Liu Y., Wu R., Jiang Y. Binary structured physics-informed neural networks for solving equations with rapidly changing solutions. Journal of Computational Physics. 2024. Vol. 518. P. 25. ISSN 0021-9991. URL: https://doi.org/10.1016/j.jcp.2024.113341 (дата звернення: 30.03.2025).
7. Romera G. Neural Networks in Numerical Analysis and Approximation Theory : Master of Mathematical Analysis and Applied Mathematics / Universidad

Complutense de Madrid. Madrid, 2024. 65 p. URL: https://doi.org/10.48550/arXiv.2410.02814 (дата звернення: 29.03.2025).

8. Rout S. Numerical approximation in cfd problems using physics informed machine learning: Master of Technology / Indian Institute of Technology Madras. Chennai, India, 2019. 72 p. URL: https://doi.org/10.48550/arXiv.2111.02987 (дата звернення: 29.03.2025).
9. Navarro L. M., Moreno L. M., Rodrigo S. G. Solving differential equations with Deep Learning: a beginner's guide. European Journal of Physics. 2023. Vol. 45, iss. 1. ISSN 1361-6404. URL: https://doi.org/10.48550/arXiv.2307.11237.
10. Multi-level physics informed deep learning for solving partial differential equations in computational structural mechanics / W. He та ін. Nature. 2024. Т. 3, вип. 151 : Communications Engineering. URL: https://www.nature.com/articles/s44172-024-00303-3 (дата звернення: 29.03.2025).
11. Physics-Informed Neural Networks for Quantum Eigenvalue Problems / M. Mattheakis et al. 2022 International Joint Conference on Neural Networks, Padua, Italy, 18–23 Jul. 2022. Institute of Electrical and Electronics Engineers Conference, 2022. ISBN 978-1-7281-8671-9. URL: https://doi.org/10.1109/IJCNN55064.2022.9891944 .
12. Sirignano J., Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics. 2018. Vol. 375. P. 1339–1364. ISSN 0021-9991. URL: https://doi.org/10.1016/j.jcp.2018.08.029 .
13. E W., Jentzen A., Han J. Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations. Communications in Mathematics and Statistics. 2017. Vol. 5. P. 349–380. ISSN 2194-671X. URL: https://doi.org/10.1007/s40304-017-0117-6 .

14. Neural Operator: Learning Maps Between Function Spaces / N. Kovachki et al. The Journal of Machine Learning Research. 2023. Vol. 24, iss. 89. P. 1–97. URL: http://jmlr.org/papers/v24/21-1524.html (дата звернення: 31.03.2025).
15. Fourier Neural Operator for Parametric Partial Differential Equations / Z. Li et al. International Conference on Learning Representations, 3–7 May 2021. URL: https://iclr.cc/virtual/2021/poster/3281 (дата звернення: 31.03.2025).
16. Lu L., Jin P., and Karniadakis G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. 2019. URL: https://doi.org/10.48550/arXiv.1910.03193.
17. Physics-informed attention-based neural network for hyperbolic partial differential equations: application to the Buckley–Leverett problem / R. Rodriguez-Torrado et al. Nature. 2022. Vol. 12, iss. 7557 : Scientific Reports. URL: https://doi.org/10.1038/s41598-022-11058-2.
18. Shang Y., Wang F., and Sun J. Deep Petrov-Galerkin Method for Solving Linear And Nonlinear Partial Differential Equations. 2023. URL: http://dx.doi.org/10.2139/ssrn.4420653.
19. Lu Y., Lu J., Wang M. A Priori Generalization Analysis of the Deep Ritz Method for Solving High Dimensional Elliptic Partial Differential Equations. Proceedings of Thirty Fourth Conference on Learning Theory, 15–19 Aug. 2021. PMLR, 2021. P. 3196—3241. URL: https://proceedings.mlr.press/v134/lu21a.html (дата звернення: 31.03.2025).
20. Hashimoto K., Matsuo K., Murata M., and Ogiwara G. Comparative Study of Neural Network Methods for Solving Topological Solitons. 2024. URL: https://arxiv.org/html/2411.14942v1.
21. Wavelets based physics informed neural networks to solve non-linear differential equations / Z. Uddin та ін. Nature. 2023. Vol. 13, iss. 2882 : Scientific Reports. URL: https://www.nature.com/articles/s41598-023-29806-3 (дата звернення: 29.03.2025).
22. Abbasi, J., Jagtap, A., Moseley, B., Hiorth, A., and Andersen P. Challenges and Advancements in Modeling Shock Fronts with Physics-Informed Neural

Networks: A Review and Benchmarking Study. 2025. URL: https://doi.org/10.48550/arXiv.2503.17379

23. Seiler E., Lei W., Protopapas P. Stiff Transfer Learning for Physics-Informed Neural Networks. 2025. URL: https://arxiv.org/abs/2501.17281.
24. Baty H. Solving stiff ordinary differential equations using physics informed neural networks (PINNs): simple recipes to improve training of vanilla-PINNs. 2023. URL: https://arxiv.org/abs/2304.08289.
25. Neural Differential Equations: A Comprehensive Review and Applications / A. Nair et al. Advances in Nonlinear Variational Inequalities. 2025. Vol. 28, iss. 1. P. 147–160. ISSN 1092-910X. URL: https://doi.org/10.52783/anvi.v28.2235.
26. Pandey H., Singh A., Behera R. An efficient wavelet-based physics-informed neural networks for singularly perturbed problems. 2024. URL: https://doi.org/10.48550/arXiv.2409.11847.
27. Wu, R., Christara C.C. Solving High-Dimensional PDEs: Deep Galerkin Method with Timestepping. University of Toronto. 2022. URL: https://www.cs.toronto.edu/~rwu/papers/dgmt.pdf (дата звернення: 31.03.2025).
28. Chen G., Xu S., Ni D., Zeng T. DGNet: A Neural Network Framework Induced by Discontinuous Galerkin Methods. 2025. URL: https://arxiv.org/html/2503.10021v1.
29. Niraj K., Evan P., and Elfving V. Integral Transforms in a Physics-Informed (Quantum) Neural Network setting: Applications & Use-Cases. 2022. URL: https://doi.org/10.48550/arXiv.2206.14184.
30. Wojtak W., Bicho E., Erlhagen W. Solving neural field equations using physics informed neural networks. 2023. URL: https://hdl.handle.net/1822/90178.
31. Holl P., Koltun V., Thuerey N. Learning to Control PDEs with Differentiable Physics. 2020. URL: https://doi.org/10.48550/arXiv.2001.07457.
32. Mukherjee A., Liu J. Bridging Physics-Informed Neural Networks with Reinforcement Learning: Hamilton-Jacobi-Bellman Proximal Policy

Optimization (HJBPPO). ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems. 2023. URL: https://openreview.net/forum?id=TZMUQ6mkhs (дата звернення: 31.03.2025).

33. Dibaji S. M., Safi M., Sharifi I. Resilient Distributed Averaging: Adversary Detection and Topological Insights. 2023. URL: https://dx.doi.org/10.2139/ssrn.4584951.

ДОДАТОК А
СТРУКТУРА ПРОГРАМНОГО ПРОЄКТУ

Програмна частина дипломної роботи реалізована у вигляді Python-проєкту, що складається з модульної структури, адаптованої для побудови та навчання PINNs. Код організовано у вигляді каталогів відповідно до функціонального призначення окремих частин.

Структура проєкту виглядає наступним чином:

```
project_pinn/
├── data/                          # Каталог для зберігання вхідних/вихідних даних
│   └── __init__.py
├── models/                        # Архітектури PINN-моделей
│   ├── __init__.py
│   ├── pinn_base.py               # Базовий клас PINN
│   ├── pinn_heat.py               # Модель для рівняння теплопровідності
│   ├── pinn_beam.py               # Модель для задачі балки
│   └── pinn_burgers.py            # Модель для рівняння Бюргерса
├── utils/                         # Допоміжні функції
│   ├── __init__.py
│   ├── autodiff.py                # Автоматичне диференціювання (похідні до 4-го порядку)
│   ├── collocation.py             # Генерація точок колокації, граничних і початкових умов
│   ├── loss_terms.py              # Обчислення окремих складових функції втрат
│   └── visualization.py           # Побудова графіків та 3D-візуалізацій
├── experiments/                   # Сценарії запуску для конкретних задач
│   ├── run_heat.py                # Модель для теплопровідності
│   ├── run_beam.py                # Модель для задачі балки
│   └── run_burgers.py             # Модель для рівняння Бюргерса
├── results/                       # Вивід результатів (графіки, логи, MSE, час)
```

```
│   ├── heat_solution.png
│   ├── beam_solution.png
│   ├── burgers_solution.png
│   └── result_log.txt
├── requirements.txt            # Список програмних залежностей
└── README.md                   # Опис проєкту та інструкція
```

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА

У процесі реалізації програмного забезпечення я працювала в середовищі PyCharm, яке є зручним і потужним інструментом для розробки застосунків на Python. Саме PyCharm надає широкі можливості для роботи з віртуальними середовищами, встановлення бібліотек, написання, запуску й налагодження коду – тому це середовище рекомендоване для подальшого використання та аналізу проєкту.

Проєкт створений мовою програмування Python, з використанням бібліотеки PyTorch для побудови й навчання PINNs. Також використовувалися допоміжні пакети, такі як NumPy (для роботи з масивами), Matplotlib (для побудови графіків) та інші. Усі залежності зазначені у файлі requirements.txt, тому після відкриття проєкту в PyCharm можна швидко встановити всі необхідні бібліотеки.

Щоб розпочати роботу, достатньо відкрити папку з проєктом у PyCharm, створити (або обрати вже наявне) віртуальне середовище з інтерпретатором Python версії 3.10 або вище, після чого встановити залежності за допомогою команди *pip install -r requirements.txt*. PyCharm сам запропонує це зробити автоматично, якщо побачить файл з переліком бібліотек.

Усі запускні сценарії для моделей розміщені в каталозі experiments/. Наприклад, щоб запустити модель для розв’язання рівняння теплопровідності, достатньо відкрити файл run_heat.py і натиснути Shift + F10. Аналогічно запускаються інші задачі: run_beam.py — для моделювання деформації балки, run_burgers.py — для задачі з рівнянням Бюргерса.

Після виконання моделі в автоматичному режимі буде побудовано відповідний графік результату у форматі .png, а також збережено текстовий лог-файл, у якому фіксуються MSE та загальний час навчання моделі. Ці

файли можна знайти в каталозі results/, де вони автоматично зберігаються з назвами відповідно до задачі.

Також у коді можна змінювати параметри навчання моделі, зокрема кількість епох, кількість нейронів на шарі, кількість шарів, кількість точок колокації тощо. Це дозволяє підлаштовувати точність і швидкість розрахунків під конкретну задачу. Такі налаштування змінюються безпосередньо у файлах run_*.py, і всі зміни будуть автоматично враховані при наступному запуску.