Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ВПОРЯДКУВАННЯ ДАНИХ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ

RESEARCH OF DATA ORDERING ALGORITHMS USING NEURAL NETWORKS

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Бас Дмитро Васильович
_____
(підпис)
Керівник: к.т.н., доцент
Здолбіцька Ніна Василівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович
_____
(підпис)

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____В. Ліщина
«14» травня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Бас Дмитро Васильович

1. Тема кваліфікаційної роботи Дослідження алгоритмів впорядкування даних за допомогою нейронних мереж
керівник роботи: к.т.н., доцент Здолбіцька Н.В.
затверджені наказом закладу вищої освіти від *«14»травня 2025 р. № 255/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «5» грудня 2025 р.
3. Вихідні дані до роботи опубліковані зарубіжні та вітчизняні публікації в даній області, технічна документація про системи прогнозування замовлень товарів, різні інтернет-ресурси технічного спрямування
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області
Нейронні мережі та їх застосування нейронні мережі та їх застосування
Дослідження моделей із явними та неявними правилами сортування даних
5. Перелік графічного матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Здолбіцька Н. В.*		
Нейронні мережі та їх застосування нейронні мережі та їх застосування	*Здолбіцька Н. В.*		
Дослідження моделей із явними та неявними правилами сортування даних	*Здолбіцька Н. В.*		
Показник запозичень тексту	_____%		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОП	*Ліщина В. О.*		

7. Дата видачі завдання *«14» травня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 22.09.2025 р.*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 17.10.2025 р.*	
3	*Структурна схема об'єкта проектування*	*до 01.11.2025 р.*	
4	*Практична реалізація об'єкта проектування*	*до 15.11.2025 р.*	
5	*Обгрунтувати переваги використання вибраної технології*	*до 18.11.2025 р.*	
6	*Формування списку використаних джерел*	*до 25.11.2025 р.*	
7	*Формування додатків*	*до 1.12.2025 р.*	
8	*Здача чистового варіанту кваліфікаційної роботи на кафедру*	*до 05.12.2025 р.*	

Здобувач вищої освіти _____ Дмитро БАС

Керівник кваліфікаційної роботи _____ Ніна ЗДОЛБІЦЬКА

АНОТАЦІЯ

Бас Д. В. Дослідження алгоритмів впорядкування даних за допомогою нейронних мереж. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У Розділі 1 проводиться аналіз предметної області, який охоплює основні категорії, характеристики та класифікацію традиційних алгоритмів сортування. Визначаються виклики та обмеження класичних методів, а також підкреслюються переваги впорядкування даних у цілому.

Розділ 2 фокусується на нейронних мережах, розкриваючи їхню сутність, сфери застосування та переваги використання в сучасних інформаційних системах. Детально проаналізовано виклики та обмеження застосування нейронних мережах. Подано опис етапів виконання алгоритму вибору параметрів штучної нейронної мережі, що закладає теоретичну основу для її практичного застосування у завданнях впорядкування даних.

Розділ 3 містить програмну реалізацію нейромережевої моделі, проведено порівняльний аналіз ефективності НМ-моделі з класичними алгоритмами. Здійснено дослідження поведінки моделі на задачі відновлення цілісності візуальних даних, включаючи аналіз масштабованості та обчислювальної складності.

Ключові слова: нейронні мережі, Pointer Network, сортування даних, LSTM, механізм уваги, комп'ютерний зір, комбінаторна оптимізація.

ANNOTATION

Dmytro Bas. Research of data ordering algorithms using neural networks. Manuscript.

Qualifying master's work of the EP «Computer Science». Lutsk National Technical University. Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, a list of sources used, and appendices.

Section 1 provides an analysis of the subject area, covering the main categories, characteristics, and classification of traditional sorting algorithms. The challenges and limitations of classical methods are identified, and the advantages of data organization in general are emphasized.

Section 2 focuses on neural networks, revealing their essence, areas of application, and advantages of use in modern information systems. The challenges and limitations of the application of neural networks are analyzed in detail. The stages of execution of the algorithm for selecting the parameters of an artificial neural network are described, which lays the theoretical foundation for its practical application in data organization tasks.

Section 3 contains a software implementation of the neural network model, a comparative analysis of the effectiveness of the NM model with classical algorithms is carried out. The behavior of the model on the task of restoring the integrity of visual data is studied, including an analysis of scalability and computational complexity.

Keywords: neural networks, Pointer Network, data sorting, LSTM, attention mechanism, computer vision, combinatorial optimization.

ЗМІСТ

ВСТУП

Актуальність дослідження зумовлена постійно зростаючими обсягами даних, що вимагають ефективних методів обробки. Традиційні алгоритми сортування, попри їхню високу ефективність для структурованих даних, не завжди є оптимальними для вирішення складних, нетипових завдань, що виникають в умовах сучасних Big Data, а також для сортування нечислових, неструктурованих даних. Класичні алгоритми сортування (QuickSort, Timsort) є високоефективними для скалярних даних, проте виявляються непридатними для задач, де критерій впорядкування є неявним, семантичним або базується на візуальній неперервності (відновлення документів, складання пазлів). Дослідження можливостей нейромереж у цій сфері відкриває шляхи до автоматизації складних комбінаторних задач.

Застосування нейронних мереж для сортування відкриває нові можливості для створення адаптивних та оптимізованих рішень, здатних навчатися на основі даних, а не лише слідувати жорстким правилам. Назріла потреба дослідження алгоритмів впорядкування даних для спеціалізованих задач за допомогою нейронних мереж є у сучасному світі, що стрімко розвивається. Це обумовлено кількома ключовими факторами.

У добу Big Data обсяги інформації зростають експоненційно, ефективність впорядкування даних стає критично важливим для їх зберігання, пошуку та аналізу. Традиційні алгоритми сортування, хоча й оптимізовані, можуть стикатися з обмеженнями продуктивності та адаптивності до неструктурованих або динамічних наборів даних.

Ведеться пошук нових парадигм для класичних задач. Сортування є фундаментальним завданням в інформатиці. Дослідження їх вирішення за допомогою нейронних має на меті розробку абсолютно нових, гнучкіших або адаптивних алгоритмів, здатних самостійно навчатися оптимальним стратегіям впорядкування.

Відбувається зростання складності даних та потреба в адаптивності. Сучасні дані нерідко є складними, багатовимірними, з шумом або прихованими залежностями. Нейронні мережі, завдяки своїй здатності до навчання на нелінійних патернах, можуть запропонувати кращі адаптивні рішення для впорядкування таких даних там, де класичні алгоритми можуть бути менш ефективними або вимагати значних ручних налаштувань.

Так як традиційні алгоритми мають добре відому складність, нейронні мережі можуть запропонувати потенційно нові шляхи для паралелізації обчислень або оптимізації, що у певних умовах може призвести до прискорення процесу впорядкування даних або покращення його характеристик на спеціалізованому обладнанні.

Дослідження гібридних систем та штучного інтелекту зумовило розвиток інтеграції нейронних мереж у класичні алгоритмічні задачі. Сильні сторони машинного навчання поєднуються з перевагами традиційних підходів. Це є кроком до створення автономніших та інтелектуальних систем обробки даних.

Алгоритми впорядкування даних за допомогою нейронних мереж завдяки можливості виявлення нових, більш адаптивних та потенційно ефективних підходів до фундаментальної задачі сортування відповідають викликам сучасної обробки даних та розвитку штучного інтелекту.

Метою кваліфікаційної роботи є проведення порівняльного аналізу та виявлення специфіки застосування нейронних мереж для задач впорядкування даних.

Завдання дослідження:

– проаналізувати класичні алгоритми сортування та їх переваги, недоліки та обмеження застосування;

– вивчити існуючі моделі машинного навчання, які можуть бути застосовані для сортування даних спеціалізованих задач;

– сформувати набір даних для навчання та тестування розробленої моделі;

– розробити або адаптувати модель нейронної мережі, здатної виконувати операції впорядкування;

– експериментально порівняти ефективність розробленої моделі з традиційними алгоритмами сортування;

– визначити сценарії, у яких використання нейронних мереж для сортування є найдоцільнішим;

– сформулювати рекомендації щодо вибору методу впорядкування залежно від типу даних, явні чи неявні ознаки, та вимог до обчислювальної складності.

Об’єктом дослідження є процес впорядкування послідовностей даних різної природи: скалярних величин (чисел) та векторних об’єктів (фрагментів зображень).

Предметом дослідження є алгоритми впорядкування даних, реалізовані та оптимізовані за допомогою нейронних мереж.

Наукова новизна роботи полягає у систематичному дослідженні потенціалу нейронних мереж для самостійного вивчення та реалізації логіки впорядкування даних, що виходить за рамки традиційного програмування алгоритмів сортування; розробці або адаптації конкретної архітектури нейронної мережі, яка демонструє здатність до ефективного сортування, та її порівняльний аналіз з класичними алгоритмами впорядкування.

Виявленні специфічних переваг та обмежень підходу до сортування на основі нейронних мереж, зокрема щодо його масштабованості, адаптивності до різних типів даних та можливості навчання на прикладах.

Практичне значення кваліфікаційної роботи полягає у демонстрації альтернативного підходу до вирішення задачі сортування, що може відкрити нові можливості для оптимізації процесів обробки даних у складних або непередбачуваних середовищах.

Можливості застосування отриманих результатів у галузях, що вимагають адаптивного або самонавчального сортування, наприклад, у базах даних з нестабільною структурою, системах штучного інтелекту, що працюють

з потоками даних, або для оптимізації внутрішніх механізмів обробки інформації у розподілених системах.

Створенні експериментальної основи для подальших досліджень у галузі застосування машинного навчання для оптимізації класичних алгоритмів та розробки адаптивних структур даних.

Розширення професійних навичок у сфері машинного навчання, аналізу даних та експериментального програмування.

Апробація результатів роботи. Тези [1] по матеріалах даної кваліфікаційної роботи представлялися на 2 міжнародній науковій конференції (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні категорії методів сортування

Структури даних є фундаментальною частиною програмування, а методи сортування – це алгоритми, які організовують ці структури найчастіше у вигляді масивів чи списків або іншої інформації у певному порядку, тобто зростання або спадання.

Операція сортування значно прискорює пошук, злиття та обробку даних, особливо у тих сферах, де потрібно працювати з великими масивами даних.

Основні сфери застосування сортування:

– пошук даних (Data Searching), адже сортування є попередньою умовою для найшвидших алгоритмів пошуку, таких як бінарний пошук (Binary Search). Якщо дані відсортовані, час пошуку скорочується з O(n) до O(logn), тобто має лінійну часову складність алгоритму, що критично важливо для великих баз даних;

– для баз даних (Databases), зокрема реляційних використовується впорядкування результатів запиту order by та для оптимізації роботи індексів, що значно прискорює вибірку даних;

– необхідність ефективного злиття даних (Data Merging) двох або більше відсортованих списків в єдиний відсортований список. Використовується у таких алгоритмах зовнішнього сортування Merge Sort, а також при роботі з файлами великого об’єму, що не вміщуються в пам’ять;

– аналіз даних та статистика, впорядковані дані спрощують аналіз при обчисленні медіани, виявленні дублікатів елементів та формування звітів;

– у комп’ютерній графіці та візуалізації сортування допомагає при вирішенні проблеми видимості, наприклад, алгоритм «художника», де об’єкти повинні бути промальовані у правильному порядку від дальнього до ближнього, щоб створити правильну перспективу та глибину;

– в алгоритмах планування завдань, наприклад планування запитів до диска в операційних системах, відбувається сортуються за пріоритетом або фізичним розташуванням на диску для мінімізації часу доступу.

1.2 Характеристики та виклики традиційних алгоритмів

Наведемо основні характеристики традиційних алгоритмів сортування.

Часова складність вимірюється кількістю операцій залежно від розміру вхідних даних (O(NlogN), O(N^2), O(N+k)).

Поняття просторової складності визначає обсяг додаткової пам’яті, що необхідна для сортування.

Стабільність алгоритму визначає наскільки зберігається відносний порядок елементів з однаковими ключами.

Здатність ефективно працювати з різними типами даних, таких як числа, рядки, складні об’єкти або за наявності частково відсортованих даних характеризують адаптивність алгоритму.

Паралелізація означає можливість ефективного розпаралелення обчислень.

Незважаючи на високу оптимізацію та теоретично обґрунтовану складність, традиційні алгоритми є детермінованими і не завжди можуть з легкістю адаптуватися до непередбачуваних або незбалансованих даних без попередньої обробки, вимагаючи чітко визначених операцій порівняння або розподілу.

Основні цілі алгоритмів сортування полягають у впорядкуванні набору елементів (чисел, рядків, об’єктів) за певним критерієм:

– числовий порядок через зростання або спадання значень;

– лексикографічний порядок по алфавіту для текстових даних;

– спеціальний порядок, в якому наприклад за одним або кількома полями об’єктів відбувається сортування товарів за ціною, потім за назвою, і т. д.

Впорядковані дані значно спрощують та прискорюють виконання багатьох інших операцій.

Методи сортування поділяють на категорії за їх складністю та принципом дії (табл. 1.1).

Таблиця 1.1 – Основні категорії методів сортування

Категорія	Характеристика	Приклади
Просте сортування	Часова складність $O(N^2)$. Ефективні лише для невеликих наборів даних або для навчання	Сортування бульбашкою, вставками, вибором
Ефективне сортування	Часова складність O(NlogN). Найкращий вибір для великих масивів даних	Швидке сортування (Quick Sort), сортування злиттям (Merge Sort), сортування купою (Heap Sort)
Сортування без порівняння	Часова складність O(N+k), де k – діапазон значень. Не порівнюють елементи, а використовують їхні значення для прямого розміщення	сортування підрахунком (Counting Sort), порозрядне сортування (Radix Sort)

Значення сортування виходить за межі простого впорядкування і має глибокий вплив на продуктивність та ефективність алгоритмів. Сортування забезпечує попередню обробку для багатьох складніших алгоритмів.

При правильному виборі алгоритму сортування, наприклад, Quick Sort замість Bubble Sort може скоротити час обробки з годин до секунд, що прямо впливає на оптимізацію ресурсів споживання процесорного часу та енергії. Факт відсортованих даних спрощує логіку багатьох програмних модулів і зменшує кількість помилок.

Автори [2] надають комплексний аналіз продуктивності алгоритмів сортування, їх ефективність та придатність для різних реальних завдань, оцінку часової складності, стабільності, вимоги до пам’яті, поділяючи такі алгоритми по двох типах сортування: на основі порівняння та сортування без

порівняння. Розглянуто вплив об’єму вхідних даних, розподілу та порядку на продуктивність алгоритму.

Необхідно враховувати важливість вибору відповідного алгоритму сортування на основі конкретних характеристик даних та вимог застосування.

Графік продуктивності за часом алгоритмів сортування для випадково згенерованих тестових N елементів подано на рисунку 1.1 [3].

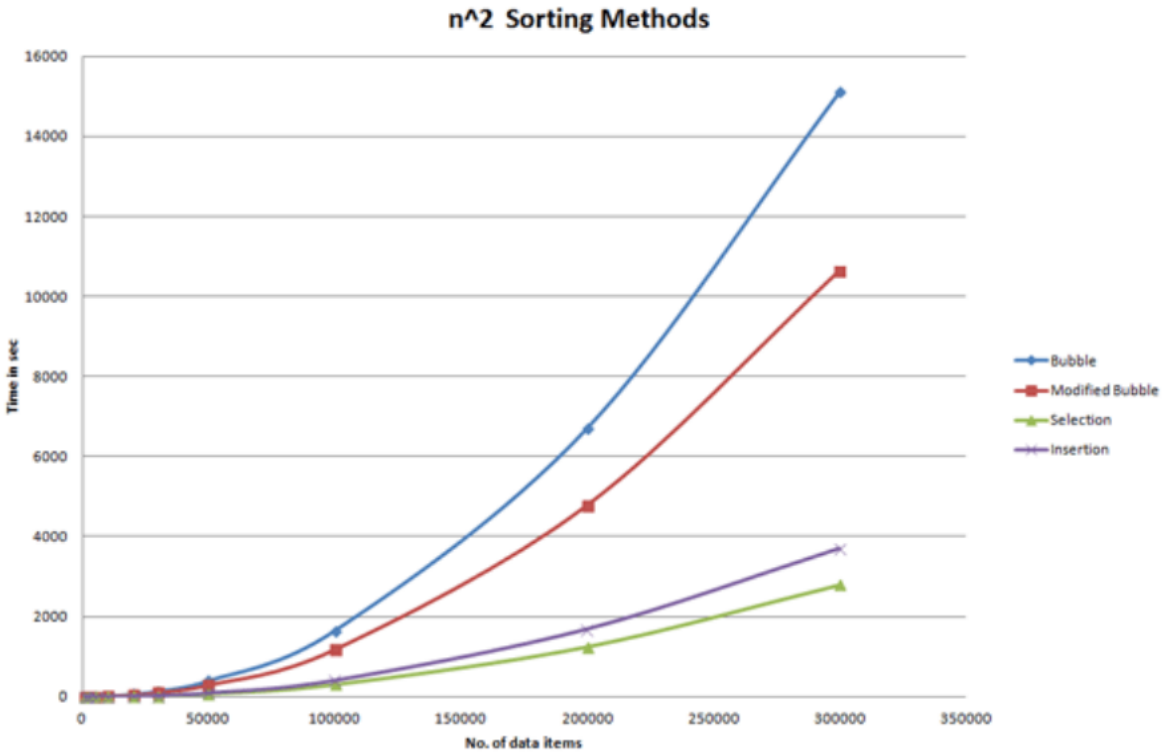

Рисунок 1.1 – Графік алгоритмів сортування класу N^2 за часом [3]

З аналізу рисунка 1.1 модифіковане бульбашкове сортування займає менше часу порівняно зі стандартним бульбашковим сортуванням, крім того, сортування вибором працює краще порівняно з іншими [3].

На рисунку 1.2 стандартне швидке сортування Quick працює ефективніше, ніж сортування купою Heap та сортування злиттям Merge.

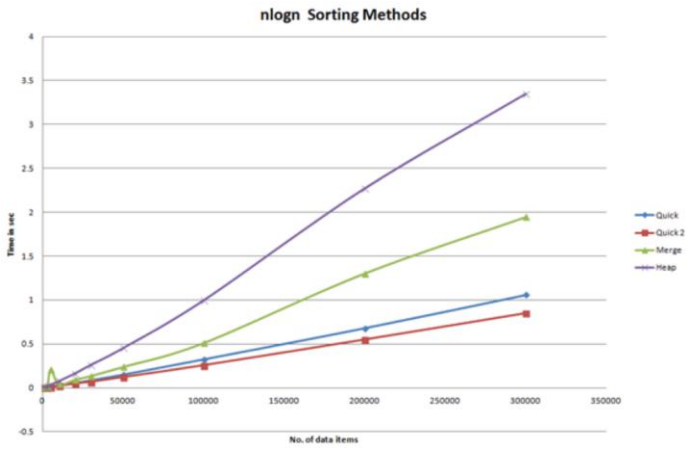

Рисунок 1.2 – Графік алгоритмів сортування класу NlogN за часом [3]

Покращена версія Quick2 краща за інші та займає менше часу порівняно з іншими, якщо масив вхідних значень замінити з цілого на рядок [3].

1.3 Класифікація та приклади традиційних алгоритмів сортування даних

У статтях [4, 5] здійснено огляд традиційних алгоритмів, порівняння наведених часових залежностей процедури сортування.

Проведемо порівняння алгоритмів сортування.

Перевагою методів сортування, наведених у таблицях 1.2-1.3 є стабільність.

Таблиця 1.2 – Сортування злиттям

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Сортування злиттям (Merge Sort)	Сортування великих наборів даних, зв'язаних списків та інфраструктури даних, зовнішнє сортування (коли дані не вміщуються у пам'ять). Використовує підхід «Розділяй і володарюй»	Стабільна ефективність O(NlogN) в усіх випадках. Стабільний алгоритм, добре підходить для паралелізації та сортування зв'язаних списків	Потребує багато додаткової пам'яті (O(N) для злиття, що може бути проблемою для систем з обмеженими ресурсами. Витрати на додаткову пам'ять

Таблиця 1.3 – Сортування вставками

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Сортування вставками (Insertion Sort)	Сортування дуже малих масивів. Використовується як етап оптимізації у складніших алгоритмах (наприклад, Timsort)	Висока ефективність на майже відсортованих даних O(N). Простий і стабільний	Дуже повільний $O(N^2)$ на великих невідсортованих масивах

Разом з тим методи сортування, наведені у таблицях 1.4-1.7 мають гіршу складність.

Таблиця 1.4 – Сортування вибором

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Сортування вибором (Selection Sort)	Використовується, коли кількість обмінів має бути мінімізована (наприклад, для сортування даних із великим кешом переміщення)	Простота реалізації. Мінімальна кількість операцій обміну (лише n обмінів, обмін відбувається лише один раз за кожну ітерацію)	Повільний ($O(N^2)$ в усіх випадках). Нестабільний Неефективний для великих наборів даних.

Таблиця 1.5 – Сортування бульбашкою

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Сортування бульбашкою (Bubble Sort)	Виключно для навчальних цілей або для демонстрації концепції сортування. Не використовується у професійних застосунках	Найпростіший для розуміння, навчання та реалізації	Найгірша ефективність ($O(N^2)$, дуже повільний. Низька швидкість на великих даних

Таблиця 1.6 – Сортування купою

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Сортування купою (Heap Sort)	Реалізація пріоритетних черг, для алгоритмів планування завдань в операційних системах та симуляціях. Сортування даних у режимі реального часу у системах управління базами даних	Гарантована часова складність O(NlogN). Ідеально підходить для систем із обмеженою мінімальною оперативною пам'яттю, оскільки це алгоритм, що сортує «на місці» (in-place)	Відсутність стабільності. Кеш-пам'ять менш ефективно використовується. Складніший для розуміння та реалізації, ніж прості алгоритми $O(N^2)$ або Quick Sort

Таблиця 1.7 – Швидке сортування

Назва алгоритму	Застосування	Переваги	Недоліки та обмеження
Швидке сортування (Quick Sort)	Найпоширеніший алгоритм для загального сортування при обробці тексту, для невеликих наборів даних в оперативній пам'яті. Використовується в більшості стандартних бібліотек	Дуже швидкий на практиці (O(NlogN) у середньому). Сортує «на місці» (in-place), вимагаючи мінімум додаткової пам'яті (O(logn))	Нестабільний. Найгірша складність $O(N^2)$ при невдалому виборі опорного елемента (хоча це рідко трапляється на практиці). Продуктивність погіршується з незбалансованими даними

Традиційні алгоритми, такі як швидке сортування та сортування злиттям, є фундаментальними для операцій сортування, кожен з яких має унікальні властивості, що роблять їх придатними для певних сценаріїв, але можуть виникати проблемами, якщо дані незбалансовані або потрібна висока швидкість при одночасному низькому використанні пам'яті [6].

Загальні обмеження класичних алгоритмів:

– неефективність $O(n^2)$ на великих даних алгоритмів Bubble, Selection, Insertion, які виявилися неприйнятно повільними для масивів, що містять мільйони елементів;

– проблема з нечисловими даними. Так як більшість класичних алгоритмів оптимізовані для порівняння чисел, а для сортування складних об’єктів або неструктурованих даних, зображень чи текстових описів потрібне попереднє вилучення ознак або використання спеціальних функцій порівняння;

– ризик зниження ефективності при наявності складнішого вхідного набору даних;

– нездатність до адаптації класичних алгоритмів, адже вони є статичними і виконують фіксований набір інструкцій. Вони не можуть «навчатися» на властивостях вхідних даних для зміни своєї стратегії у режимі реального часу, перевіряти чи дані майже відсортовані, що є перевагою алгоритмів на основі машинного навчання.

Визначення потрібного алгоритму для конкретної ситуації може бути складним завданням, оскільки існують різні фактори, що впливають на його ефективність [6].

1.4 Переваги впорядкування даних

Впорядкування даних за допомогою алгоритмів сортування надає ряд істотних переваг:

– покращення читабельності та представлення даних;

– ефективний пошук у базі даних;

– знаходження дублікатів чи унікальних елементів;

– спрощення процедури злиття та об’єднання даних в єдину базу;

– полегшення вибірки мінімальних чи максимальних значень;

– підготовка до подальших алгоритмів

Після сортування даних, пошук якогось конкретного елемента стає значно швидшим. Замість лінійного перебору всіх елементів з набору даних, можна користуватися бінарним пошуком з логарифмічною складністю O(logN). Так час пошуку у відсортованому масиві зростає дуже повільно зі зростанням обсягу даних.

Відсортовані списки легко об’єднати в один впорядкований за лінійний проміжок часу O(N), це критично важливо у базах даних та об’ємниих системах обробки даних. Без попереднього сортування таке злиття стає значно складніше та повільніше.

Дублікати елементів у відсортованому масиві розміщуються поряд, що допомагає їх ідентифікувати та вилучити за необхідності. Це є ефективним методом для очищення даних та забезпечення їхньої унікальності.

При потребі мінімальні та максимальні елементи знайти просто, вони розміщені відповідно на початку та в кінці відсортованого масиву. Медіана, середнє значення також легко визначається, бо знаходиться у центрі відсортованого набору даних. Це дозволяє швидше обчислити статистичні показники без додаткових обчислень.

Багато алгоритмів ефективно працюють або взагалі можуть функціонувати лише з відсортованими даними. Наприклад, алгоритми стиснення даних, аналізу часових рядів, або деякі алгоритми машинного навчання можуть вимагати впорядкованих вхідних даних для коректної роботи або для досягнення максимальної продуктивності.

Для аналізу та сприйняття людині легше даються впорядковані дані. Інформація подана у логічній та структурованій формі краще підходить для звітів, інтерфейсів користувача для візуалізації даних.

1.5 Постановка завдання на кваліфікаційну роботу

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– проаналізувати класичні алгоритми сортування та їх переваги, недоліки та обмеження застосування;

– вивчити існуючі моделі машинного навчання, які можуть бути застосовані для сортування даних спеціалізованих задач;

– сформувати набір даних для навчання та тестування розробленої моделі;

– розробити або адаптувати модель нейронної мережі, здатної виконувати операції впорядкування;

– експериментально порівняти ефективність розробленої моделі з традиційними алгоритмами сортування;

– визначити сценарії, у яких використання нейронних мереж для сортування є найдоцільнішим;

– сформулювати рекомендації щодо вибору методу впорядкування залежно від типу даних, явні чи неявні ознаки, та вимог до обчислювальної складності.

РОЗДІЛ 2
НЕЙРОННІ МЕРЕЖІ ТА ЇХ ЗАСТОСУВАННЯ

2.1 Сутність нейронних мереж

Нейронні мережі – це обчислювальні моделі алгоритмів, натхненні структурою та функціонуванням людського мозку. Вони складаються з взаємопов'язаних «нейронів», організованих у шари, які здатні навчатися на наборах даних, виявляти складні закономірності та робити прогнози чи класифікації без видимого програмування правил (рис. 2.1).

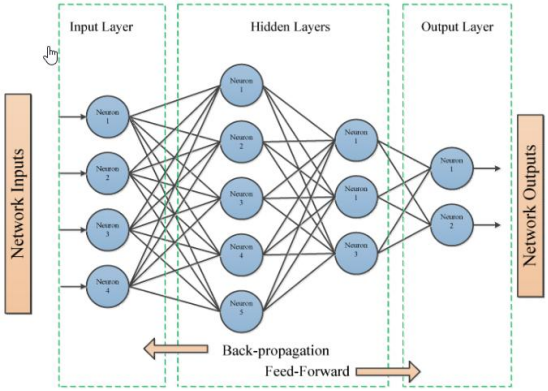

Рисунок 2.1 – Будова штучної нейронної мережі з алгоритмами прямого та зворотного поширення [7]

Існує багато інших типів штучних нейронних мереж. Основні з них наведено на рисунку 2.2. Кожен вид нейронних мереж є унікальним, якщо навіть зображення вузлів та зв'язків схоже, вони можуть набувати різного сенсу, процес їх навчання насправді зовсім різний [7].

Вичерпний перелік скласти практично неможливо, оскільки постійно розробляються нові архітектури. У статті [8] виконано огляд методів оптимізації алгоритмів на основі штучних нейронних мереж. З метою вдосконалення нейронної мережі за допомогою алгоритмів оптимізації можна скористатися класичними алгоритмами, налаштувавши відповідні параметри для отримання найкращого шаблону структури нейронної мережі для вирішення проблем.

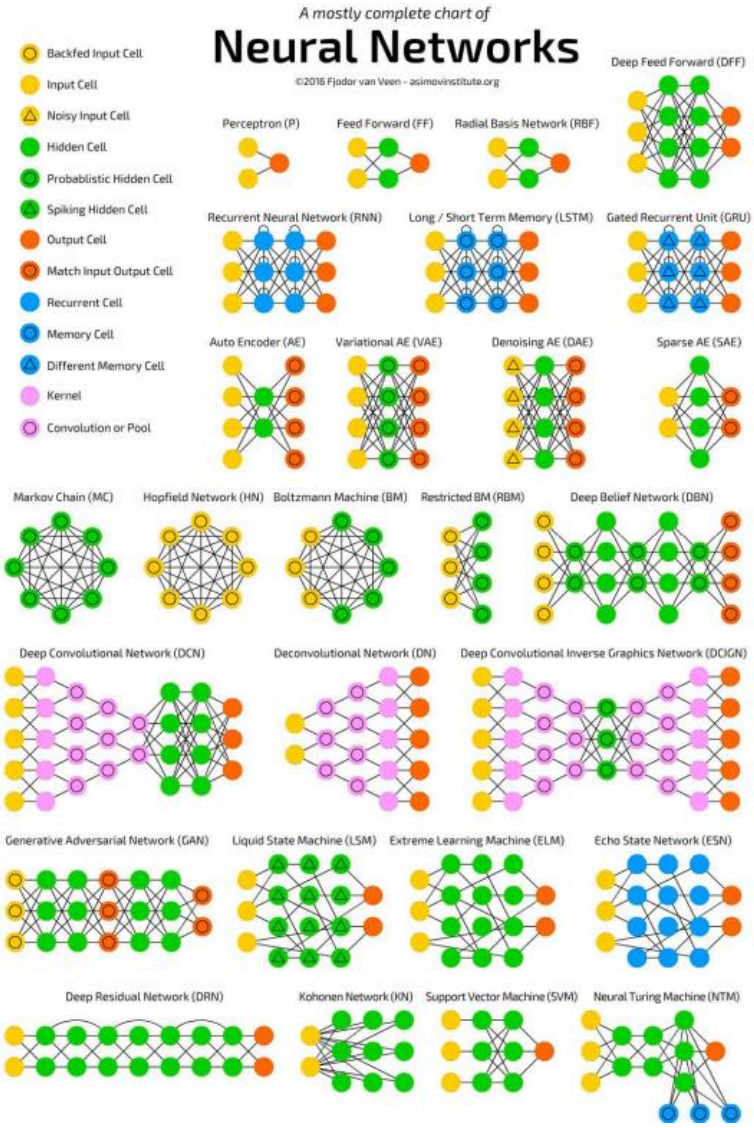

Рисунок 2.2 – Типи архітектури штучних нейронних мереж [8]

Якщо в нейронній мережі один прихований шар, то мережа називається неглибокою, якщо кількість більша – глибокою (рис. 2.3).

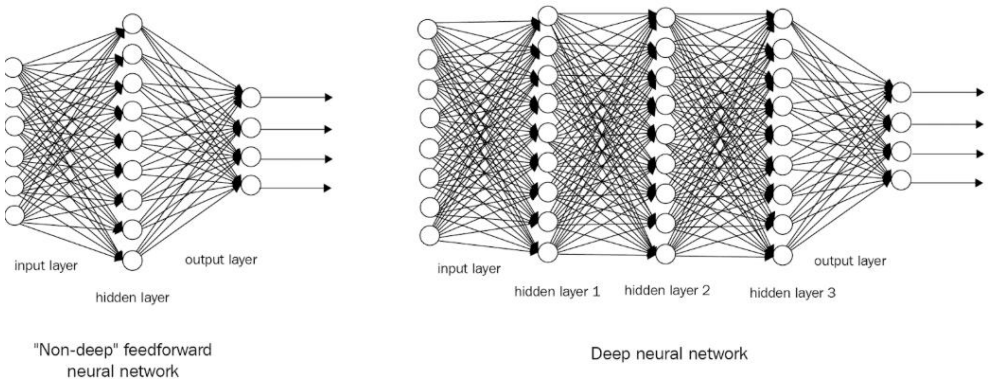

Рисунок 2.3 – Неглибока та глибока нейронні мережі [9]

Проведемо огляд існуючих моделей глибокого машинного навчання, що можна використати при для класифікації та сортування даних [10, 11]:

– нейронна мережа прямого поширення (Feedforward Neural Network FFNN / Multi-Layer Perceptron MLP), що працюють за принципом прямого поширення, де дані рухаються в одному напрямку без петель зворотного зв'язку, справляються з такими завданням, як проста класифікація, розпізнавання обличь, комп'ютерний зір та розпізнавання мовлення, завдяки односпрямованому потоку даних;

– згорткова нейронна мережа (Convolutional Neural Network, CNN), що включає тривимірне розташування нейронів, яку застосовують переважно для аналізу просторових даних, зокрема сигнали або зображення, а також розпізнавання мовлення і машинний переклад;

– рекурентна нейронна мережа (Recurrent Neural Network, RNN) підходить для аналізу послідовностей та встановлення часових залежностей даних, обробка тексту з автоматичними підказками, перевіркою граматики. Long Short-Term Memory (LSTM) – спеціальний тип рекурентної нейронної мережі, що вирішує проблему зникаючого градієнта (vanishing gradient problem), яка властива класичним RNN. Це досягається завдяки використанню складної внутрішньої структури, відомої як комірка пам'яті (memory cell), яка керується трьома основними вентилями (gates);

– модульна нейронна мережа або трансформер (Transformers), що використовує механізм уваги для виявлення найважливіших аспектів даних для класифікації, застосовується у системах прогнозування фондового ринку, як адаптивна багаторівнева нейронна мережа для розпізнавання символів та стиснення вхідних даних високого рівня;

– нейронна вказівна мережа (Neural Pointer Network, NPN) – модель глибокого навчання, яка використовує шари взаємопов'язаних вузлів (нейронів) для вивчення складних закономірностей у даних, їх впорядкування, тобто визначення правильного порядку елементів. Дана архітектура нейронної

мережі вирішує задачі упорядкування шляхом вибору індексів з вхідної послідовності.

2.2 Сфери застосування

Нейронні мережі людство навчилося успішно застосовувати для широкого спектру завдань:

– розпізнавання та сегментація об’єктів, комп’ютерний зір;

– навчання ШІ-агентів для освоєння складних стратегічних ігор;

– комунікація, що охоплює обробку природної мови, переклад, аналіз та генерацію тексту;

– розробка систем прогнозування та персоналізованих рекомендацій, створення моделей для передбачення часових рядів та попиту.

Нейронні мережі швидко набувають популярності, цим скористалися великі корпорації, визнавши їхній практично безмежний функціонал для вирішення широкого спектру бізнес-завдань (рис. 2.4).

Рисунок 2.4 – Приклади використання нейронних мереж [12]

Використання нейромереж уже стало невід’ємною частиною стратегії світових лідерів (табл. 2.1). Нейронні мережі стали потужним інструментом процесу автоматизації обслуговування клієнтів, адже вони дозволяють створювати чат-боти та віртуальні помічники. Це значно покращує

клієнтський досвід та водночас оптимізує бізнес-процеси, забезпечуючи швидке вирішення запитів без участі людини, що забезпечує підвищення якості обслуговування та раціоналізацію бізнес-витрат (табл. 2.1).

Таблиця 2.1 – Приклади застосування нейронних мереж

Компанія	Застосування нейронних мереж
Amazon, Netflix	Персоналізовані рекомендації продуктів та контенту
Samsung, Nike	Розробка нових, персоналізованих продуктів та надання персональних послуг
Tesla	Системи автономного водіння
Google	Розпізнавання мови та автоматичний переклад
Walmart	Оптимізація та підвищення ефективності управління ланцюгами поставок
PayPal	Виявлення та запобігання шахрайським транзакціям
Microsoft	Створення та вдосконалення віртуальних помічників

У статті [13] автори відтворили адаптивний алгоритм сортування на базі штучного інтелекту, спеціально розроблений для обробки великих даних завдяки інтеграції класичних та інноваційних методів. Дослідження включає вичерпний порівняльний аналіз як алгоритмів сортування з порівнянням, так і без, оцінюючи їхню ефективність, часову складність, стабільність та вимоги до пам’яті для різноманітних практичних завдань. Особливу увагу приділено впливу на продуктивність таких характеристик, як об’єм, розподіл та початковий порядок вхідних даних.

Застосування науки про дані та штучного інтелекту набуло особливої важливості у бізнес-аналітиці, дозволяючи приймати рішення, що ґрунтуються на наявних даних. ШІ та наука про дані мають величезний потенціал для перетворення необробленої інформації на практичні висновки для прогнозування та загальної обробки даних [14, 15]. Зокрема, автори статті [16] вивчають сфери застосування, переваги, методології та проблеми, пов’язані з інтеграцією цих технологій у бізнес-аналітику.

Автори [17] розробили на основі ШІ адаптивний алгоритм сортування для Big Data, перевіривши його ефективність, швидкість та використання пам’яті на різних наборах даних. Дослідження показує, що інтеграція ШІ з класичними методами істотно покращує обробку великомасштабних даних, що є критичним для сучасних динамічних інформаційних систем.

2.3 Переваги використання

Наведемо переваги використання нейронних мереж при обслуговуванні клієнтів, що забезпечує значну оптимізацію у таблиці 2.2.

Таблиця 2.2 – Переваги використання нейронних мереж

Перевага	Механізм реалізації нейронних мереж
Доступність 24/7	Робота без перерв і залежності від часових поясів дозволяє задовольняти запити клієнтів функціонуючи цілодобово
Прискорення обслуговування	Чат-боти миттєво надають негайні відповіді на типові запитання, усуваючи необхідність очікування в черзі або чекаючи відповідь на email
Персоналізовані рекомендації	Аналізуючи дані про клієнтів пропонуються індивідуальні рекомендації щодо продуктів або рішень, максимально адаптовані до інтересів та потреб конкретного користувача
Підвищення точності	Здатність навчатися на великих обсягах даних забезпечує постійне вдосконалення розуміння запитів і зменшення кількості помилок
Інформаційна безпека	Запобігання інцидентам безпеки, виявлення аномалій у мережевому трафіку, журналах подій або аудіосигналах
Економія ресурсів	Автоматизація зменшує необхідність у великій кількості операторів, знижуючи загальні витрати компанії на підтримку
Сортування даних	Впорядкування елементів як основа для ефективного виконання алгоритмів пошуку, злиття, обробки баз даних

Стандартні алгоритми сортування даних працюють за конкретно вказаними правилами, тоді як нейромережі можуть знайти закономірності у даних, що дозволяє їм адаптуватися до складних і неструктурованих масивів.

Розглянемо переваги використання нейронних мереж до класичних алгоритмів:

– нейромережі застосовуються до складних, нетрадиційних даних, можуть самостійно виділяти складні, нелінійні взаємозв’язки;

– використовуючи можливість адаптації до нових даних або умов через прийом перенавчання нейронні мережі цим перевершують звичайні алгоритми, які не мають властивості адаптивності та навчання на існуючих даних;

– нейронні мережі ефективніші при роботі з неструктурованими даними;

– у певних випадках нейронна мережа може знайти ефективніший спосіб сортування, ніж просто традиційні алгоритми, що призводить до прискорення процесу впорядкування, оптимізуючи кількість операцій для конкретних типів даних;

– окремі нейронні мережі можуть обробляти дані паралельно, що може прискорити процес сортування на відповідному програмному та апаратному забезпеченні.

2.4 Виклики та обмеження застосування

Попри значні переваги, інтеграція нейронних мереж у різні сфери, зокрема для сортування, зіштовхується з низкою суттєвих викликів (табл. 2.3).

Таблиця 2.3 – Виклики, пов’язані з ресурсами та даними

Виклик	Опис
Потреба у великих даних	Успішне навчання НМ вимагає значних обсягів високоякісних наборів даних, що не є завжди доступними
Обчислювальна складність	Тренування та експлуатація великих НМ є ресурсомістким і дорогим процесом, що потребує значних обчислювальних потужностей
Складність навчання	Навчання моделей машинного навчання є значно ресурсозатратнішим та складнішим, ніж розробка та написання детермінованого класичного алгоритму

Обмеження, пов’язані з надійністю та інтерпретацією даних наведено у таблиці 2.4.

Таблиця 2.4 – Обмеження, пов’язані з надійністю та інтерпретацією

Обмеження	Опис
Проблема «чорного ящика»	Існує складність в інтерпретації внутрішньої логіки та механізмів, за якими нейромережі приймають рішення, що ускладнює їх довіру в критичних системах
Неефективність для простих завдань	Для простих числових даних традиційні алгоритми сортування (Quick Sort, Merge Sort) залишаються неперевершеними за швидкістю та обчислювальною ефективністю, оскільки вони оптимізовані саме для цього
Ризик перенавчання (Overfitting)	Існує висока ймовірність того, що модель «запам’ятає» навчальні дані, втрачаючи здатність до узагальнення та точності на нових, раніше не використовуваних даних
Точність і передбачуваність	На відміну від класичних алгоритмів із гарантованим точним результатом, нейромережі можуть помилятися, особливо якщо стикаються з даними, які дуже відхиляються від навчального набору

Навчання нейронних мереж сортуванню даних не може на даному етапі замінити класичні алгоритми у всіх випадках, але розкриває нові можливості для вирішення складних, нетрадиційних завдань, де важлива гнучкість та адаптивність до даних.

Хоча нейромережі поки що переважно використовуються для вузьких задач, їхній потенціал для зростання залишається колосальним, розглянемо ключові перспективи:

– значне підвищення точності та загальної ефективності моделей через створення інноваційних архітектур, оптимізація процесів навчання та використання потужніших обчислювальних ресурсів;

– постійне розширення сфер застосування, де можуть бути використані моделі нейронних мереж;

– розробка методики доступності даних, які що дозволять навчати моделі на невеликих наборах даних (datasets) та надасть можливість застосування нейронних мереж за умов обмеженої інформації;

– створення самонавчальних систем, які на основі аналізу реальних даних будуть здатні до безперервного самовдосконалення вмінь і знань.

2.5 Етапи виконання алгоритму вибору параметрів штучної нейронної мережі для впорядкування даних

У роботі [18] досліджувалося застосування штучних нейронних мереж для моделювання даних, сортування, розв’язання певних складних проблем, у яких важко виявляти тенденції або закономірності.

Основними викликами дослідження є визначення репрезентації даних, тобто як подати дані для сортування нейронній мережі, обрання архітектури НМ, що найкраще підходить для вивчення логіки впорядкування, далі генерація або збір навчальних даних та вкінці оцінка ефективності роботи алгоритму.

Важливо використовувати структурований підхід до підготовки даних, адже ефективність класифікаційних моделей на основі нейронних мереж прямо залежить від якості вхідних даних. Навіть найдосконаліші нейронні мережі не дадуть коректних результатів на невідфільтрованих даних, що робить попередню підготовку даних критично важливим етапом [19].

Ключовим аспектом цієї підготовки є відбір значущих ознак. Необхідно зосередитися лише на тих характеристиках, які реально дозволяють розрізняти класи. Наприклад, ознаки, що описують габарити, не слід використовувати, якщо розміри об’єктів різних класів майже ідентичні. При цьому важливо знайти баланс: використання нерелевантних ознак ускладнює модель, тоді як виключення роздільних ознак погіршує здатність моделі до класифікації.

Розглянемо процес реалізації моделі штучних нейронних мереж [19] (рис. 2.5).

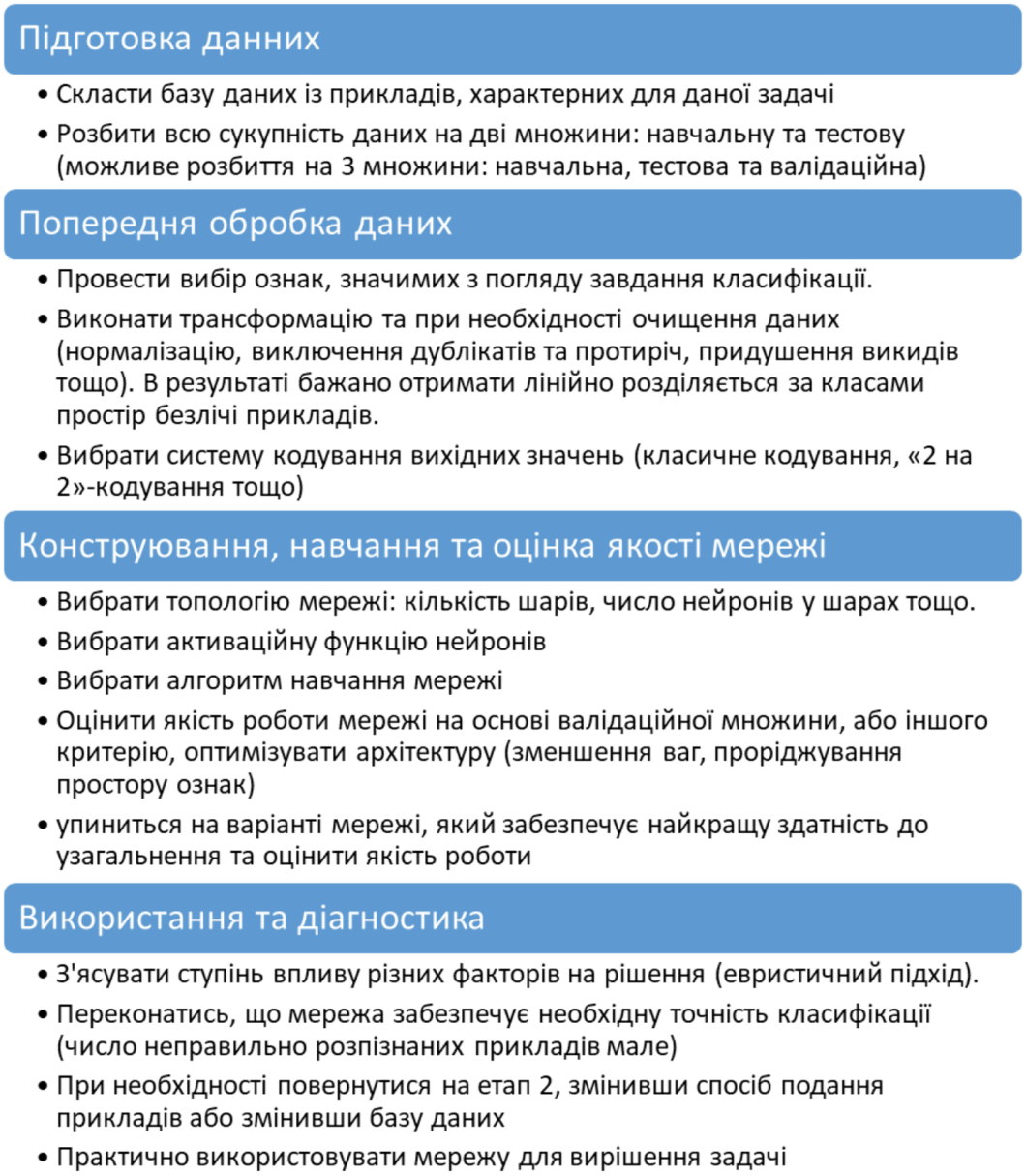

Рисунок 2.5 – Модель штучних нейронних мереж [19]

Подамо етапи виконання алгоритму вибору параметрів штучної нейронної мережі у вигляді таблиці 2.5.

Процес налаштування нейронної мережі є багатоетапним і послідовним алгоритмом, який вимагає точної формалізації задачі перед вибором архітектурних рішень. Ключовими кроками є вибір функції активації, що впливає на швидкість навчання та оптимізацію.

Таблиця 2.5 – Етапи виконання алгоритму вибору параметрів ШНМ

Етап	Крок	Опис
1. Визначення входу та виходу	Формалізація задачі (вектори X та Y)	Чітко визначити вхідний вектор X, який повинен містити всю необхідну інформацію для вирішення задачі, та вихідний вектор Y, який представлятиме повну і формалізовану відповідь
2. Вибір архітектури	Вибір функції активації	Обрати відповідну функцію активації для нейронів, враховуючи специфіку завдання, оскільки вдалий вибір прискорює процес навчання
3. Налаштування топології	Визначення структури мережі	Вибрати оптимальну кількість шарів та нейронів у кожному шарі
4. Ініціалізація параметрів	Присвоєння початкових значень	Встановити початкові значення ваг та граничних порогових рівнів. Необхідно уникати як надто великих значень, щоб не допустити насичення нейронів та уповільнення навчання, так і надто малих, щоб уникнути нульових виходів і, відповідно, також уповільнення
5. Навчання	Тренування мережі	Здійснити процес навчання, підбираючи параметри для досягнення найкращого вирішення поставленої задачі. Після завершення навчання мережа готова вирішувати завдання, на які була налаштована
6. Використання	Прогноз / розв’язання	Подати нові умови задачі у вигляді вхідного вектора X на вхід мережі, щоб розрахувати кінцевий вихідний вектор Y, який містить рішення

Успіх побудови моделі залежить від ретельного дотримання хронології цих етапів, що зрештою дозволяє провести ефективне навчання та отримати формалізоване рішення задачі.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ МОДЕЛЕЙ ІЗ ЯВНИМИ ТА НЕЯВНИМИ ПРАВИЛАМИ СОРТУВАННЯ ДАНИХ

3.1 Програмна реалізація нейромережевої моделі сортування

Для дослідження можливості навчання алгоритмів сортування розроблено програмний комплекс мовою Python (додаток Б) із використанням бібліотеки глибокого навчання PyTorch [20]. В основу реалізації покладено архітектуру мережі вказівників Pointer Networks, запропоновану Vinyals et al. Вибір цієї архітектури зумовлений специфікою задачі сортування, де вихідний алфавіт, тобто індекси відсортованих елементів, не є фіксованим, а динамічно залежить від довжини вхідної послідовності.

Структура розробленого програмного модуля складається з чотирьох функціональних компонентів:

– підсистема генерації даних DataGenerator;

– архітектура моделі NeuralSorter;

– модуль навчання Trainer;

– модуль тестування та валідації Evaluator.

Клас DataGenerator забезпечує створення синтетичних наборів даних «на льоту» (on-the-fly), що дозволяє уникнути перенавчання на фіксованій вибірці. Модуль підтримує генерацію числових послідовностей із різними законами розподілу:

– рівномірний Uniform для базового навчання;

– нормальний Gaussian та експоненційний Exponential для перевірки стійкості моделі до зміни статистичних характеристик вхідних даних.

Основна логіка сортування інкапсульована у класі NeuralSorter, який реалізує механізм Sequence-to-Sequence:

– енкодер Encoder побудований на базі рекурентної нейронної мережі (Long Short-Term Memory, LSTM). Він зчитує вхідну послідовність чисел і

перетворює її на набір прихованих векторів контексту context vectors, які зберігають інформацію про значення та позиції елементів;

– декодер Decoder з механізмом уваги Attention є ключовим елементом системи. Замість генерації абстрактних значень, декодер обчислює «ваги уваги» (attention scores) до станів енкодера. Індекс вхідного елемента, що отримав найбільшу вагу уваги, вважається вибором мережі для поточної позиції у відсортованому списку.

Клас Trainer керує процесом оптимізації параметрів мережі:

– функція втрат, що використовує CrossEntropyLoss, оскільки задача формулюється як класифікація, для кожної позиції виходу мережа повинна «вказати» на правильний індекс із N можливих варіантів входу;

– оптимізатор, використано адаптивний алгоритм Adam для градієнтного спуску;

– регуляризація, що реалізовано механізмом Early Stopping, який зупиняє навчання, якщо помилка на валідаційній вибірці перестає зменшуватися, що запобігає перенавчанню;

Клас Evaluator відповідає за проведення порівняльного аналізу. Він забезпечує:

– бенчмаркінг швидкості виконання нейромережі порівняно з класичними алгоритмами QuickSort та вбудованим у Python Timsort;

– розрахунок метрик точності: суворої Strict Accuracy, тобто повний збіг масиву, та часткової Partial Accuracy, тобто відсоток вірно розміщених елементів.

Розглянемо технічні особливості реалізації. Хоча програмна реалізація на Python поступається у швидкодії оптимізованим алгоритмам, таким як Timsort, реалізований на C, даний модуль виконує дослідницьку функцію. Він демонструє фундаментальну відмінність між алгоритмічним підходом (виконання фіксованих інструкцій порівняння) та евристичним підходом (навчання правилам впорядкування на основі прикладів). Довжина

послідовності для навчання є параметром SEQ_LEN, що дозволяє досліджувати здатність мережі до узагальнення на масивах різної розмірності.

3.2 Навчання моделі та порівняльний аналіз ефективності

Для верифікації розробленої архітектури Pointer Network та оцінки її практичної придатності проведено серію експериментів із візуалізацією динаміки навчання та результатів тестів продуктивності бенчмаркінгу (додаток Б). Для цього програмний комплекс розширено підсистемою візуалізації на базі бібліотеки matplotlib та модифіковано модуль Evaluator для динамічного тестування послідовностей змінної довжини.

Епоха (Epoch) – це одиниця виміру часу в навчанні нейромереж. Одна епоха означає, що нейромережа один раз повністю пройшла через весь навчальний набір даних.

У таблицях В.1-В.4 додатку В показано 25 епох по 1000 прикладів для навчання в кожній епосі.

Loss – функція втрат на навчанні, це показник помилки моделі на тих даних, на яких вона вчиться прямо зараз. Це різниця між тим, що мережа видала, і тим, що мало бути насправді. Чим менше це число, тим краще.

У таблиці В.2 проглядається динаміка: значення Loss падає з 2,28 до 0,57, це означає, що мережа успішно вчиться і помиляється все менше.

Val Loss – функція втрат на валідації, тобто перевірка достовірності результатів, що є найважливіший показник для оцінки якості. Це помилка моделі на нових даних валідаційної вибірки, які вона не бачила під час оновлення ваг. Використовується, щоб перевірити, чи модель справді вивчила закономірність сортування, чи просто «азубрила»навчальні приклади.

Оскільки Val Loss теж зменшується разом із Loss, перенавчання (overfitting) поки немає, процес йде правильно.

Strict Accuracy– строга точність, це відсоток випадків, коли мережа відсортувала весь масив ідеально правильно. По принципу все або нічого, якщо

у масиві з чисел хоча б одне стоїть не на своєму місці – це зараховується як помилка. У нашому випадку починається з 0 % і зростає до 15.47 %. Це свідчить про те, що задача повного сортування є складною для мережі, і їй важко відтворити ідеальну послідовність без жодної помилки.

Partial Acc – часткова точність обо Element-wise Accuracy, це відсоток окремих елементів, які стоять на своїх правильних місцях, незалежно від того, чи правильний весь масив. У таблиці додатку Б.2 часткова точність зростає з 20 % до 76 %. Це показує, що мережа загалом розуміє принцип сортування і більшість чисел ставить правильно, але часто помиляється в деталях.

Час Time – час, витрачений на проходження однієї епохи в секундах. У нашому випадку час стабільний.

Отже, нейромережа стабільно вчиться. Вона добре схоплює загальний порядок елементів, має високий Partial Acc, але їй все ще важко видати ідеально відсортований масив у 100 % випадків, тобто низький Strict Acc. Для покращення Strict Acc зазвичай потрібно більше епох навчання або зміна архітектури моделі, наприклад, додавання механізму уваги Attention.

Методика оцінювання якості. Оцінка ефективності моделі проводилася за двома ключовими критеріями:

– обчислювальна складність або швидкодія, порівняння часу виконання сортування нейромережею з класичними алгоритмами QuickSort та Timsort;

– здатність до узагальнення Generalization, перевірка точності сортування на тестових вибірках, довжина яких відрізняється від навчальної.

Для глибшого аналізу процесу навчання впроваджено дві метрики точності:

– сувора точність Strict Accuracy, це бінарна метрика, що дорівнює 1, якщо всі елементи вихідного масиву знаходяться на вірних позиціях, і 0 – в іншому випадку, дана метрика є критично важливою для алгоритмічних задач;

– часткова точність Partial Accuracy, тобто відсоток елементів масиву, які були правильно розміщені моделлю. Ця метрика дозволяє відстежувати

прогрес навчання для формування інтуїції моделі навіть тоді, коли сувора точність залишається низькою.

Навчання моделі на довжині послідовності 5 (Device: CPU) подано в таблиці 3.1, результати на рисунку 3.1.

Таблиця 3.1 – Розширений бенчмарк (Train Len: 5)

Тестова довжина	Strict Acc	Partial Acc
5	83,4 %	94,8 %
10	0,0 %	17,7 %
15	0,0 %	10,1 %
20	0,0 %	7,3 %

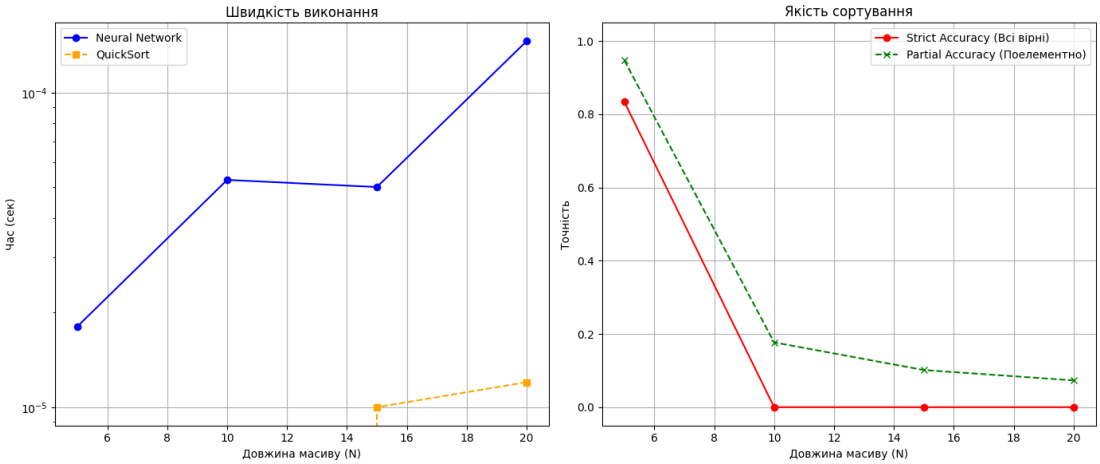

Рисунок 3.1 – Результати тестування нейромережі (навченої на N=5) порівняно з QuickSort: аналіз швидкодії та метрик точності

Навчання моделі на довжині послідовності 10 подано в таблиці 3.2, результати на рисунку 3.2.

Таблиця 3.2 – Розширений бенчмарк (Train Len: 10)

Тестова довжина	Strict Acc	Partial Acc
10	14,2 %	75,0 %
20	0,0 %	10,3 %
30	0,0 %	6,1 %
40	0,0 %	4,1 %

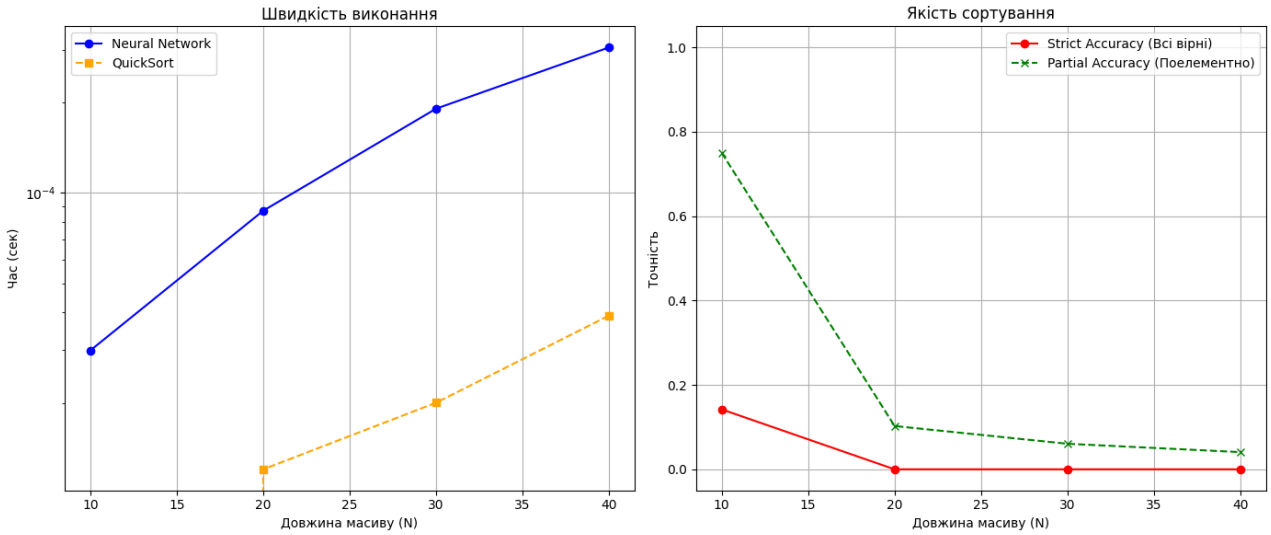

Рисунок 3.2 – Результати тестування нейромережі (навченої на N=10) порівняно з QuickSort: аналіз швидкодії та метрик точності

Навчання моделі на довжині послідовності 20 подано в таблиці 3.3, результати на рисунку 3.3 табличні дані наведено у додатку В.

Таблиця 3.3 – Розширений бенчмарк (Train Len: 20)

Тестова довжина	Strict Acc	Partial Acc
20	0,0 %	46,4 %
40	0,0 %	4,8 %
60	0,0 %	3,0 %
80	0,0 %	2,1 %

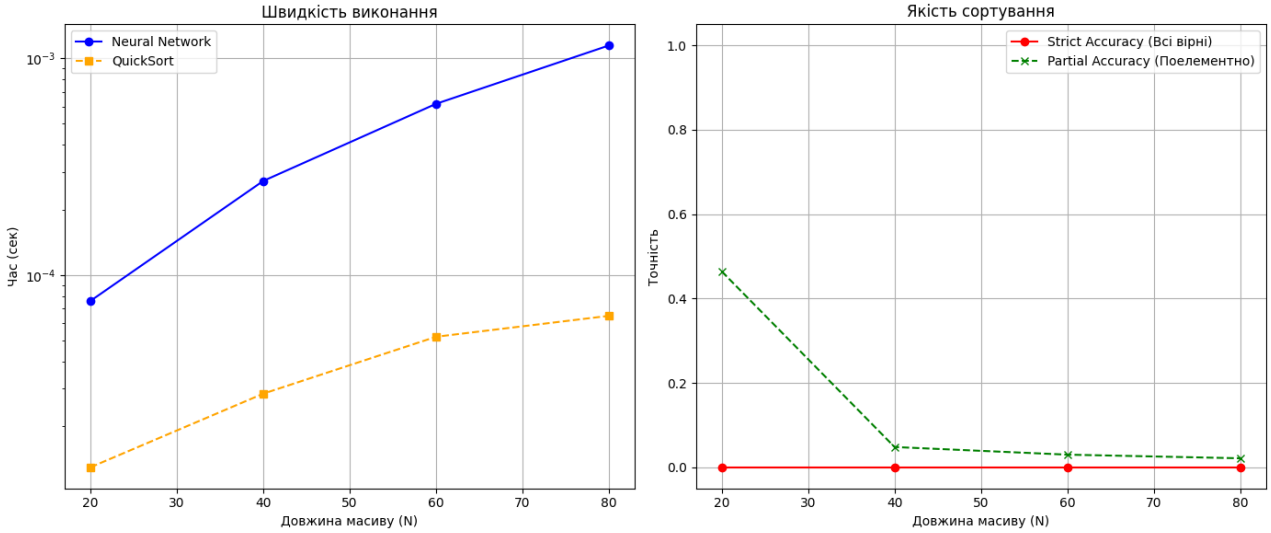

Рисунок 3.3 – Результати тестування нейромережі (навченої на N=10) порівняно з QuickSort: аналіз швидкодії та метрик точності

Навчання моделі на довжині послідовності 100 подано в таблиці 3.4, результати на рисунку 3.4.

Таблиця 3.4 – Розширений бенчмарк (Train Len: 100)

Тестова довжина	Strict Acc	Partial Acc
100	0,0 %	9,9 %
200	0,0 %	1,2 %
300	0,0 %	0,7 %
400	0,0 %	0,5 %

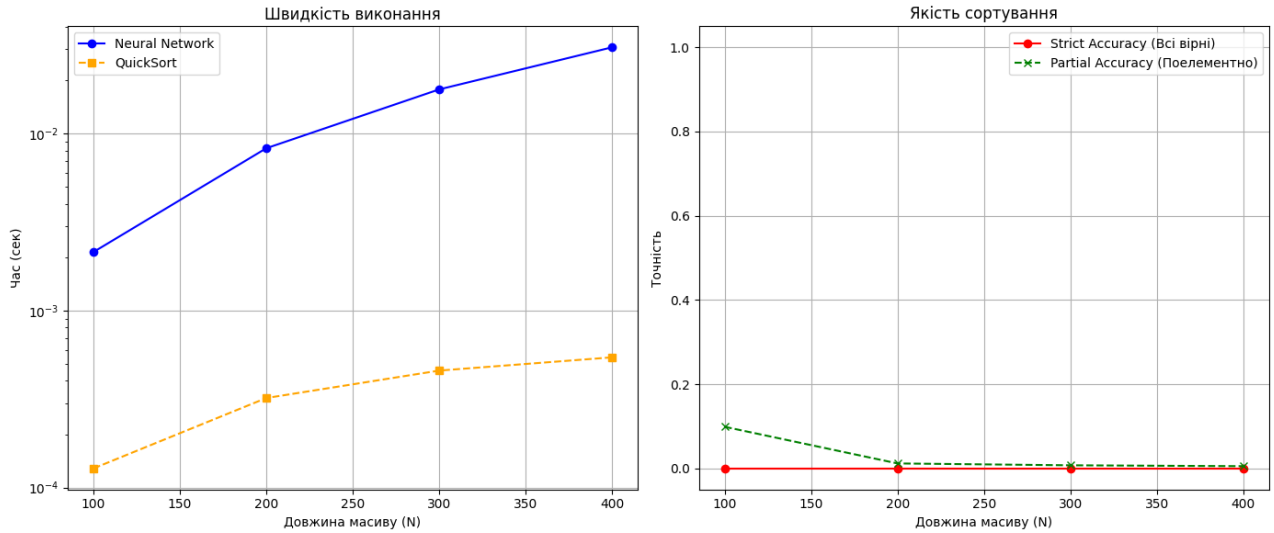

Рисунок 3.4 – Результати тестування нейромережі (навченої на N=100) порівняно з QuickSort: аналіз швидкодії та метрик точності

Проблема узагальнення Generalization Gap. Аналіз графіків, зображених на рисунках 3.1-3.4 та таблицях 3.1-3.4 демонструє неспроможність моделі екстраполювати знання на послідовності, довші за навчальні Out-of-Distribution Generalization.

Виявлено феномен: при навчанні на N=5, тестування на N=10 показує різке падіння точності. Причиною є те, що механізм уваги Attention використовує функцію Softmax, яка «розмазується» по довжині вхідного вектора. При збільшенні кількості елементів розподіл ймовірностей стає менш піковим, тобто розмитим, що призводить до помилок вказування. Мережа

вивчає не абстрактне правило $a < b$, а статистичні залежності для конкретної розмірності вектора.

Аналіз швидкодії алгоритмів. Порівняльний аналіз часу виконання Time Complexity на логарифмічній шкалі виявив суттєву розбіжність між нейромережевим та класичним підходами:

– Timsort Python sorted продемонстрував найкращий результат в наносекундах для малих N. Це пояснюється низькорівневою оптимізацією, реалізованою на C та гібридною природою алгоритму;

– QuickSort показав стабільну ефективність зі складністю O(NlogN);

– Neural Network виявилася найповільнішою, на порядок повільніше за QuickSort.

Низька швидкодія нейромережі зумовлена архітектурними особливостями Pointer Network. Для генерації кожного елемента вихідної послідовності механізм уваги Attention повинен обчислити ваги для всіх n елементів входу. Це призводить до квадратичної складності $O(N^2)$ та виконання великої кількості операцій з плаваючою комою, матричних множень всередині LSTM, на відміну від простих операцій порівняння цілих чисел у класичних алгоритмах.

Аналіз точності та проблеми комбінаторної складності. Результати тестування точності виявили обмеження Sequence-to-Sequence моделей при роботі з жорсткими алгоритмічними правилами:

– проблема «холодного старту», на початкових етапах (до 15 епох) метрика Strict Accuracy залишається близькою до 0 %. Це пов’язано з комбінаторним вибухом: для послідовності довжиною N=10 простір пошуку складає $10! \approx 3{,}6 \cdot 10^6$ варіантів. Знайти єдиний вірний варіант методом градієнтного спуску за малу кількість ітерацій вкрай важко;

– інформативність Partial Accuracy, на відміну від суворої метрики, Partial Accuracy демонструє стабільне зростання. Це свідчить про те, що модель не діє випадково, а засвоює локальні патерни впорядкування, хоча й допускає помилки в глобальній структурі;

– розрив узагальнення Generalization Gap, експерименти показали різке падіння точності при тестуванні на довжинах $N > N_{\text{train}}$. Механізм уваги, навчений фокусуватися на коротких векторах, розмивається зменшуючи значення Softmax при збільшенні довжини входу, що призводить до помилок вказівок.

Візуалізація даних підтвердила, що нейромережа Pointer Network здатна навчатися принципам сортування, проте значно поступається класичним алгоритмам у швидкості та точності на великих масивах даних. Використання метрики Partial Accuracy є необхідним для моніторингу процесу навчання, оскільки Strict Accuracy є занадто жорстким критерієм для стохастичних моделей на ранніх етапах навчання.

3.3 Аналіз результатів експерименту та поведінки моделі

На основі отриманих результатів навчання та бенчмаркінгу проведено детальний аналіз поведінки Pointer Network при вирішенні алгоритмічної задачі сортування.

Розглянемо аналіз динаміки навчання Training Phase. Процес навчання моделі на послідовностях довжиною N=10 продемонстрував стійку тенденцію до збіжності, що підтверджується наступними показниками:

– функція втрат Loss, зафіксовано зниження значення функції втрат з 2,2891 до 0,6947. Це свідчить про коректну роботу алгоритму градієнтного спуску та здатність моделі мінімізувати ентропію вибору, поступово навчаючись вказувати правильні елементи;

– сувора точність Strict Accuracy, показник зріс з 0,00 % до 11,09 %.

Хоча абсолютне значення здається низьким, для комбінаторної задачі це значущий результат. Ймовірність випадкового вгадування порядку для 10 елементів становить $1/10! \approx 2{,}7 \cdot 10^{-7}$. Досягнення 11 % підтверджує, що модель вивчила алгоритмічну залежність, проте їй бракує впевненості для безпомилкового розв’язання задачі у 100 % випадків на даному етапі навчання.

Порівняльний аналіз швидкодії Benchmark Results. Інженерний аналіз часу виконання на вибірці з 1000 масивів (N=10) виявив суттєві відмінності в обчислювальній ефективності підходів:

– Timsort Standard Python за 0,00251 с. Алгоритм показав найкращий результат, що є очікуваним для гібридного сортування, реалізованого на мові C;

– QuickSort Python за 0,00598 с. Рекурсивна реалізація виявилася повільнішою за Timsort приблизно у 2,4 рази через накладні витрати інтерпретатора Python;

– Neural Network за 0,02317 с. Нейромережа виявилася найповільнішою, приблизно у 9 разів повільніше за Timsort.

Низька швидкість нейромережі зумовлена природою обчислень. Якщо класичні алгоритми виконують прості операції порівняння чисел, то Pointer Network здійснює тисячі операцій множення матриць Floating Point Operations у шарах LSTM. Це підтверджує, що нейромережі недоцільно використовувати для класичного сортування чисел у реальних системах.

Масштабування складності Combinatorial Explosion. Експеримент виявив чітку залежність між довжиною послідовності N та деградацією якості навчання, що пояснюється факторіальним зростанням простору пошуку:

– при N=5 модель досягла 83 % суворої точності та 95 % часткової. Простір пошуку (5!=120) є достатньо малим для повного запам’ятовування патернів;

– при N=10 сувора точність впала до 14 %, проте часткова точність збереглася на рівні 75 %. Модель все ще ефективно розпізнає загальні закономірності, впорядковує менші числа на початок, але частіше помиляється в деталях;

– при N=20 сувора точність впала до 0 %, часткова – до 46 %. На множині пошуку $20! \approx 2{,}4 \cdot 10^{18}$ модель за обмежену кількість епох не змогла вивчити ідеальний алгоритм, перейшовши до евристичної апроксимації;

– при N=100 результат на рівні шуму, ≈10 % часткової точності. Мережа втратила здатність встановлювати зв’язок між значенням входу та його позицією.

Порівняння двох метрик Strict та Partial Accuracy дозволяє глибше зрозуміти інтелект моделі.

Strict Accuracy (червона лінія на графіках рисунків 3.1-3.4) – занадто жорстка метрика для оцінки процесу навчання. Вона показує 0% для N=20, створюючи хибне враження повної непрацездатності моделі.

Partial Accuracy (зелена лінія на графіках рисунків 3.1-3.4) виступає індикатором реального прогресу. Показник у 46 % для N=20 доводить, що мережа не діє випадково. Вона розуміє концепцію порядку, але допускає локальні інверсії, наприклад, плутає сусідні за значенням елементи, що є типовим для стохастичних методів оптимізації.

Провівши дослідження ефективності застосування нейронних мереж архітектури Pointer Network для вирішення алгоритмічної задачі сортування числових послідовностей на основі отриманих експериментальних даних сформульовано наступні висновки.

Експеримент підтвердив фундаментальні обмеження нейромережевого підходу для задач із чіткою логічною детермінацією:

– висока складність апроксимації дискретних правил. На відміну від задач розпізнавання образів, де допускається певна варіативність, сортування вимагає абсолютної точності. Нейромережі важко засвоїти жорстке правило порівняння ($a<b$) без надзвичайно великого обсягу навчальних даних та тривалого часу навчання;

– проблема екстраполяції Out-of-Distribution Generalization. Виявлено критичну вразливість моделі при зміні довжини вхідних даних. Модель, навчена на послідовностях довжиною N, демонструє різку деградацію точності на послідовностях довжиною N+k. Це свідчить про те, що мережа схильна запам’ятовувати статистичні залежності конкретної розмірності, а не абстрактний алгоритм;

– обчислювальна неефективність. Порівняльний аналіз швидкодії показав, що нейромережа значно поступається класичним алгоритмам. Часова складність Pointer Network складає $O(N^2)$ через необхідність обчислення матриці уваги на кожному кроці декодування, тоді як класичні алгоритми, такі як QuickSort, Timsort працюють за O(NlogN).

Попри зазначені обмеження, експеримент довів працездатність обраної архітектури Pointer Network.

Досягнення 83 % Strict Accuracy та 95 % Partial Accuracy на коротких послідовностях при N=5 є доказом концепції Proof of Concept. Це підтверджує, що механізм уваги Attention здатний виконувати функцію вказівника і може бути використаний для перевпорядкування вхідних даних.

Висока часткова точність Partial Accuracy на довших послідовностях вказує на те, що модель не діє випадково, а намагається апроксимувати порядок, що відкриває перспективи її використання в задачах нечіткого впорядкування.

Таким чином визначено рекомендації щодо оптимізації навчання. Для підвищення якості роботи моделі у подібних задачах визначено ряд необхідних модифікацій процесу навчання:

– стратегія Curriculum Learning, де замість навчання одразу на цільовій довжині, рекомендовано застосовувати поетапне ускладнення задачі (наприклад, навчання на N=5, далі N=10 та N=20). Це дозволяє сформувати стійкі ваги на простих прикладах перед переходом до складних комбінаторних множин;

– збільшення тривалості навчання. Для збіжності рекурентних мереж LSTM у задачах такого класу 15-20 епох є недостатнім. Експериментально обґрунтована необхідність збільшення кількості епох до 100-500;

– адаптація метрик. Використання виключно метрики повного збігу Strict Accuracy не дозволяє адекватно оцінювати градієнт навчання. Впровадження метрики покомпонентної точності Partial Accuracy є обов’язковим для моніторингу прогресу;

– масштабування моделі. Для запам’ятовування складніших залежностей при N>10 доцільно збільшити розмірність прихованого стану Hidden Dimension із 64 до 128 або 256 нейронів.

Програмна реалізація успішно продемонструвала здатність Pointer Network навчатися принципам впорядкування з нуля. Хоча цей підхід є економічно недоцільним для сортування чисел через наявність ефективніших алгоритмів, отримані результати є фундаментом для застосування цієї архітектури у задачах, де алгоритмічні критерії відсутні.

Як показали результати тестування даних із чітко визначеними правилами (числа, дати , стрічки), нейромережа не може змагатися з QuickSort чи іншими схожими стандартними бібліотечними алгоритмами сортування даних. Мережі перемагають там, де правило сортування є неявним, складним або візуальним, і його важко описати формулою $a<b$.

3.4 Постановка задачі відновлення цілісності візуальних даних моделей із неявними правилами сортування даних

Задача сортування смуг зображень полягає у відновленні початкового двовимірного зображення I, яке було розділене на N вертикальних фрагментів (смуг) $S = \{s_1, s_2, \dots, s_N\}$ і перемішане випадковим чином. Метою є знаходження такої перестановки індексів k, при якій послідовність $S' = \{s_{k(1)}, s_{k(2}), \dots, s_{k(N)}\}$ реконструює оригінальне зображення з дотриманням семантичної та геометричної неперервності.

На відміну від задачі сортування чисел, де критерій порядку є явним і транзитивним (якщо $a<b$ і $b<c$, то $a<c$), у задачі візуального складання впорядкованого зображення функція порівняння $f(s_i, s_j)$ є неявною. Вона базується на локальній сумісності границь суміжних смуг (visual continuity).

Розглянемо можливі обмеження класичних алгоритмічних підходів. Застосування класичних алгоритмів сортування, наприклад QuickSort до цієї задачі стикається з фундаментальними перешкодами:

– відсутність скалярної метрики, комп’ютер не оперує поняттям «цілісна картинка». Спроба звести смужку зображення до скалярного значення, тобто обчислення середньої яскравості пікселів, призводить до втрати просторової інформації;

– неефективність евристик, сортування за градієнтом яскравості (найпоширеніша евристика) працює лише для простих лінійних градієнтів. У випадку складних патернів (синусоїда, дуга) різні фрагменти зображення можуть мати ідентичні статистичні характеристики, що робить їх нерозрізнюваними для класичних методів порівняння.

Для вирішення цієї задачі пропонується концепція нейромережевого рішення з використанням архітектури Pointer Network. Припущення полягає в тому, що нейронна мережа здатна навчитися функції сумісності фрагментів шляхом аналізу стиків між смужками.

Feature Extraction з використанням згорткових шарів CNN дозволяє виділяти локальні ознаки на краях смужок (нахил лінії, колірний перехід).

Sequence Modeling, тобто рекурентний блок LSTM моделює глобальну залежність між фрагментами, визначаючи, який фрагмент є найймовірнішим продовженням попереднього.

Методологія експерименту та адаптація даних. Для демонстрації переваги нейромережевого підходу та нівелювання впливу випадкових факторів розроблено спеціальну методику підготовки та модифікації даних, визначення стратегії навчання.

Модифікація вхідних даних включає:

– збільшення ширини смуг, ширину фрагмента W_{strip} збільшено з 8 до 16 пікселів. Це забезпечує CNN-енкодеру достатнє контекстне вікно для ідентифікації напрямку ліній та градієнтів;

– спрощення патернів, для етапу валідації замість складних періодичних функцій використано чіткі геометричні примітиви, такі як діагональний градієнт, що мінімізує неоднозначність на стиках.

Стратегія навчання полягає в наступному:

– вибір фіксованого набору Fixed Dataset. Замість стохастичної генерації нових зразків у кожній епосі використовується статичний датасет із 1000 прикладів. Це дозволяє перевірити здатність моделі до запам’ятовування складних залежностей, що у даному контексті розглядається як доказ здатності мережі вивчити правила складання впорядкованого зображення;

– збільшення тривалості навчання, кількість епох збільшено до 100 для гарантування збіжності ваг моделі.

Такий підхід дозволяє створити контрольоване середовище для порівняння ефективності нейромережі (зір та пам’ять) та класичного алгоритму (сліпа евристика).

3.5 Експериментальне дослідження ефективності на різних типах візуальних патернів

Для перевірки притущення про здатність нейромереж навчатися неявним правилам сортування було проведено серію експериментів на чотирьох типах синтетичних зображень. Кожен тип патерну моделював специфічну проблему комп’ютерного зору (градієнт, неперервність ліній, симетрія, періодичність).

Параметри експериментальної установки:

– розмірність вхідних даних: смуги розміром 32 x 16 пікселів;

– кількість елементів N, експерименти проведено для N=5, 6 та 10;

– архітектура: мережа нейронних вказівників Neural Pointer Network (прихована щільність: 128/256/512);

– як контрольний метод вибрано класичний алгоритм сортування за критерієм середньої яскравості Mean Brightness Heuristic;

Розглянемо результати експерименту 1: лінійний градієнт (Linear Gradient). Опис патерну: зображення з монотонною зміною інтенсивності кольору зліва направо.

Результати для зразка в 5 та 6 смуг подано на рисунку 3.5 та рисунку 3.6, табличні дані наведено у додатку Д.

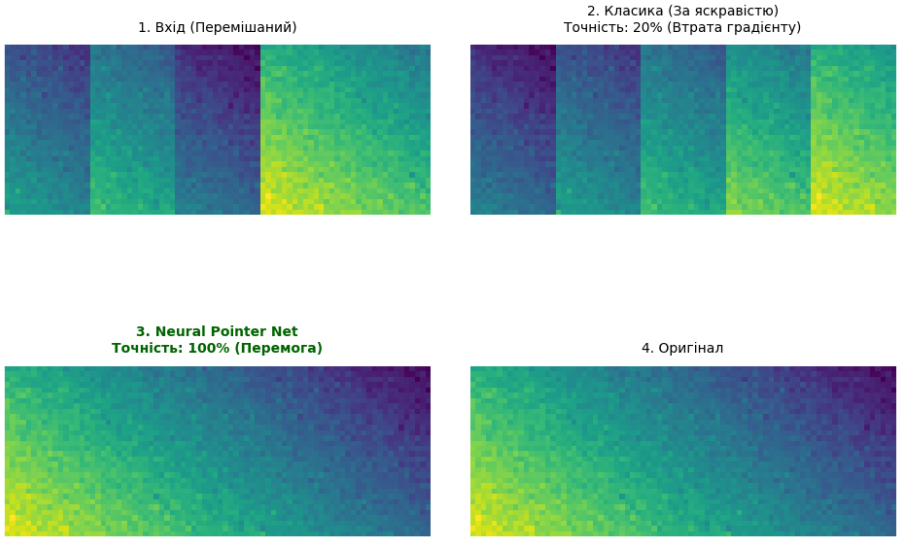

Рисунок 3.5 – Зразок 5 смуг, вхід перемішаний

Для нейромережі маємо 100% точності за 80 епох (для N=5). Для класичного алгоритму також 100% точності.

Цей випадок (рис. 3.5) слугував базовою перевіркою baseline. Висока ефективність обох методів пояснюється наявністю прямої кореляції між позицією смужки та її середньою яскравістю (Position prop to Brightness). Для нейромережі це найпростіша задача, яка зводиться до порівняння інтенсивності.

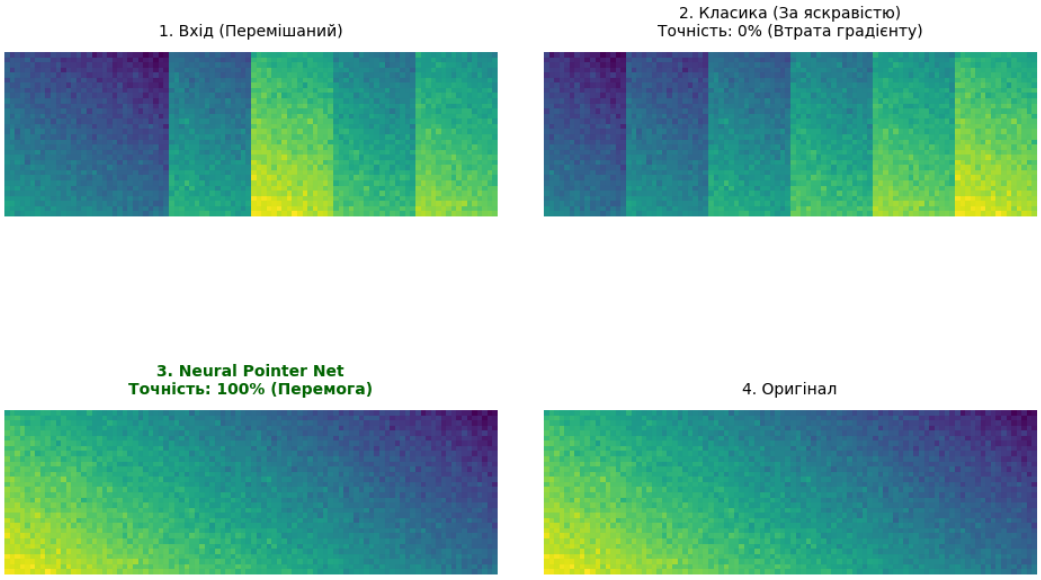

Рисунок 3.6 – Зразок 6 смуг, вхід перемішаний

Результати для зразка в 10 смуг подано на рисунку 3.7.

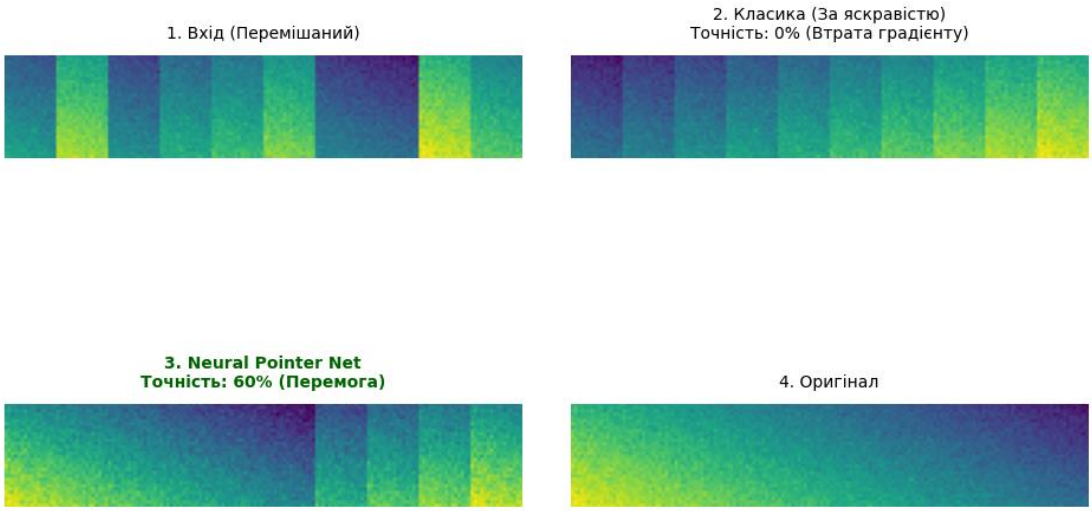

Рисунок 3.7 – Зразок 10 смуг, вхід перемішаний

Для зразка з10 смуг виявлено, що не достатнє навчання, точність 60 %

Розглянемо результати експерименту 2: мандрівна лінія (Traveling Line). Опис патерну: біла діагональна лінія на чорному тлі. Цей патерн перевіряє здатність моделі відстежувати геометричну неперервність.

Для нейромережі Strict Accuracy маємо 100 % точності. Класичний алгоритм має 0% точності як результат на рівні випадкового вгадування.

Аналіз феномену:

– провал класичних алгоритмів, адже середня яскравість, тобто кількість білих пікселів на всіх смужках є статистично однаковою. Сортування за скалярним значенням стає неможливим;

– успіх нейромережі, CNN-енкодер успішно виділив локальні ознаки на краях смужок, а саме координати виходу лінії на правій межі та входу на лівій. Механізм уваги реконструював глобальну структуру, зіставляючи ці координати.

Результати подано на рисунках 3.8 та 3.9, табличні дані наведено у додатку Е.

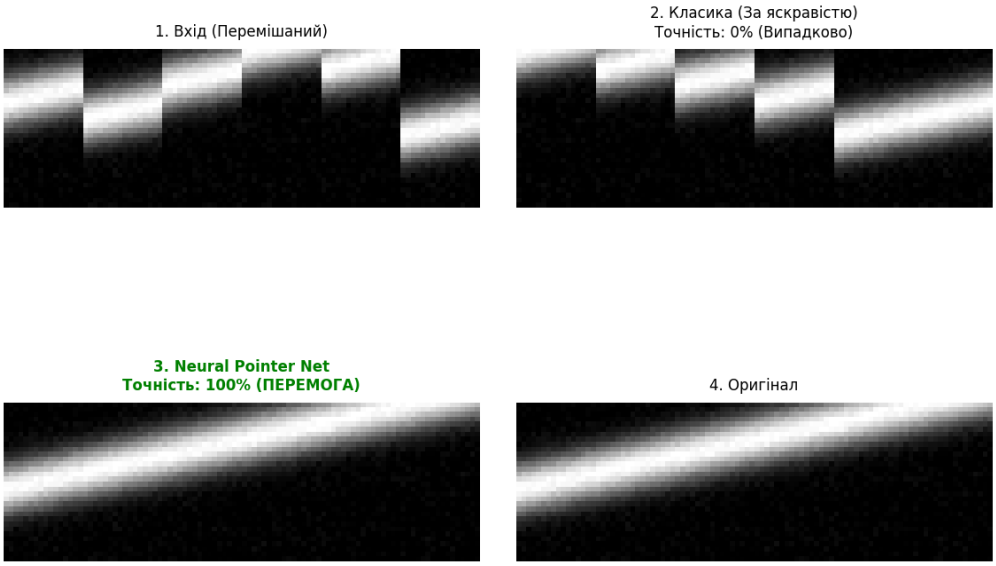

Рисунок 3.8 – Приклад «мандрівна лінія» (Traveling Line, 6)

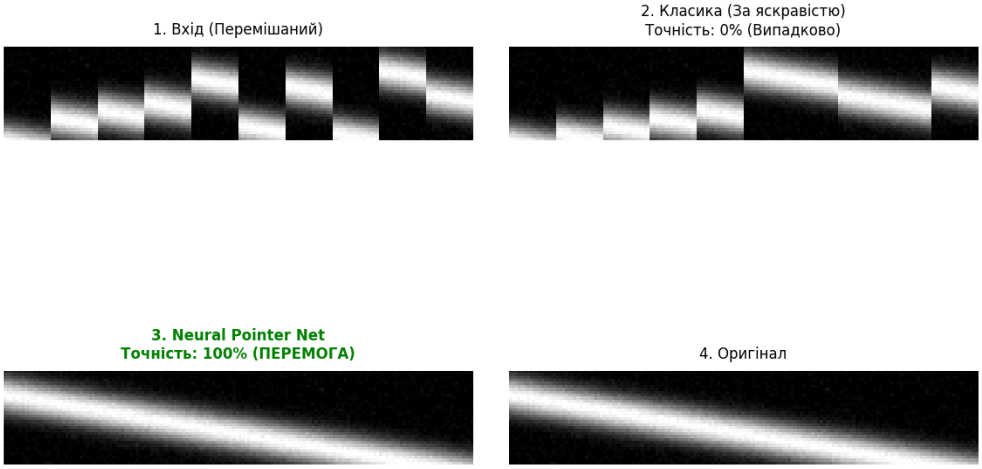

Рисунок 3.9 – Приклад «мандрівна лінія» (Traveling Line, 10)

Параметри вхідних даних відповідно до рисунків 3.8 та 3.9 подано у лістингах 3.1 та 3.2.

Лістинг 3.1 – Параметри вхідних вимог для Traveling Line_6

```
NUM_STRIPS = 6       # вхідна вимога
H, W = 32, 16
HIDDEN = 256         # Оптимально для 6 смужок
EPOCH = 1500         # Достатньо для збіжності
BATCH_SIZE = 32      # Менший розмір для частіших оновлень
NUM_SAMPLES = 2000
```

кінець лістингу 3.1

Лістинг 3.2 – Параметри вхідних вимог для Traveling Line_10

```
NUM_STRIPS = 10       # вхідна вимога
H, W = 32, 16
HIDDEN = 256          # Оптимально для 10 смужок
EPOCH = 1500          # Достатньо для збіжності
BATCH_SIZE = 32       # Менший розмір для частіших оновлень
NUM_SAMPLES = 2000
```

кінець лістингу 3.2

Експеримент 2 мав на меті дослідити здатність Neural Pointer Network вирішувати задачу упорядкування елементів, представлених у вигляді послідовності графічних смуг.

Ключовим результатом є ефективність нейромережі. Нейронна вказівна мережа досягла ідеальної точності упорядкування, точність 100 %, успішно відновивши початковий порядок «мандрівної лінії» з перемішаного вхідного зображення. На противагу цьому, класичний метод, вказаний як «Класика за яскравістю» показав точність: 0 %, що підкреслює непридатність простих евристик для цієї задачі. Стабільна точність 100 % при різних вхідних вимогах (6 та 10 смужок) свідчить про те, що архітектура Neural Pointer Network є масштабованою та ефективною для завдань упорядкування, незалежно від незначних змін у вхідній послідовності.

Розглянемо результати експерименту 3: дуга / півколо (Arc Pattern). Опис патерну: симетричне зображення дуги. Патерн перевіряє стійкість до симетрії.

Для нейромережі маємо 100 % точності. Класичний алгоритм показав 10 -50 % точності відповідно для 6 та 10 смужок, виникли системні помилки перестановки лівої та правої сторін (рис. 3.10-3.11), та додатку Є.

Через дзеркальну симетрію лівої висхідної та правої спадна частини дуги мають ідентичні статистичні показники яскравості. Класичний алгоритм не розрізняє їх. Нейромережа вирішила задачу завдяки здатності CNN розпізнавати напрямок нахилу лінії, тобто геометрію, що дозволило правильно орієнтувати фрагменти відносно центру.

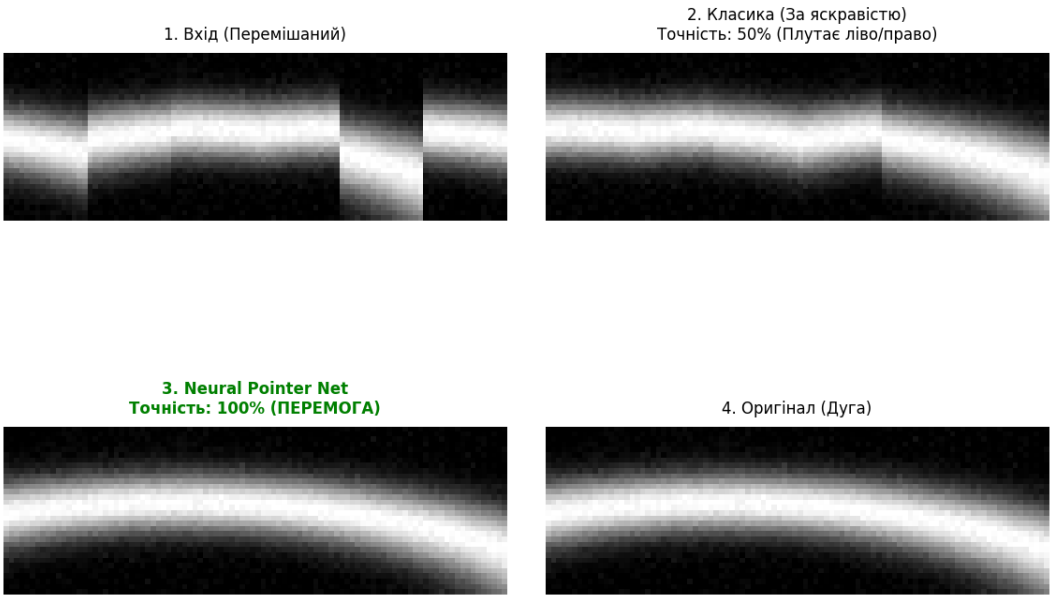

Рисунок 3.10 – Тест «Arc Pattern» , кількість смужок 6

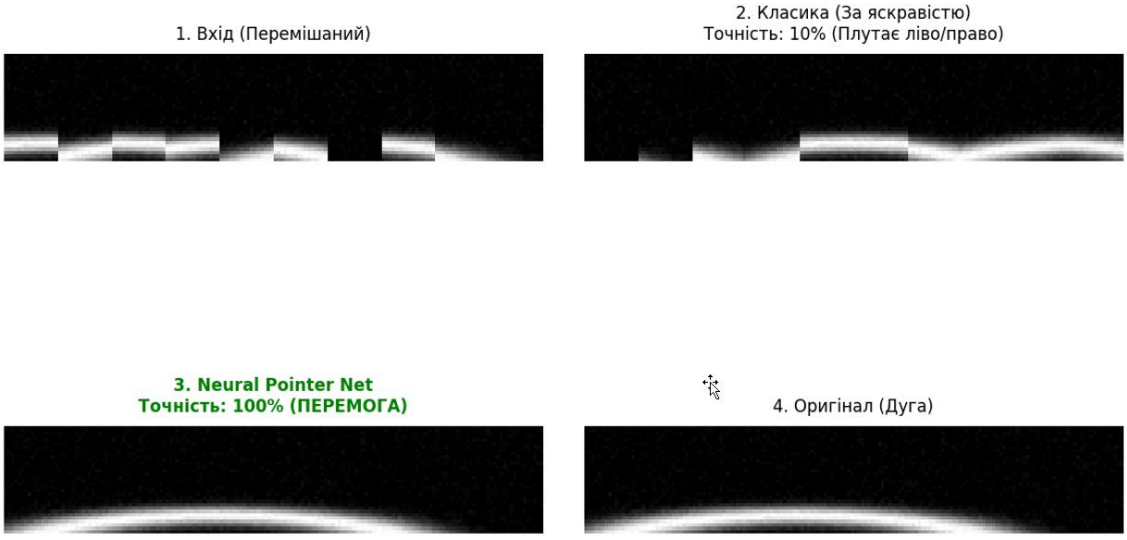

Рисунок 3.11 – Тест «Arc Pattern» , кількість смужок 10

Результати експерименту 3 підтверджують, що Neural Pointer Network є високоефективним інструментом для вирішення задач, що вимагають вивчення послідовності та упорядкування елементів. Ця архітектура успішно виявляє приховану логіку структури навіть тоді, коли елементи вхідного набору сильно перемішані.

Розглянемо результати експерименту 4: синусоїда (Sine Wave). Опис патерну: періодична функція синуса.

Результатом є відсутність збіжності (Strict Accuracy = 0 %) навіть після 2000 епох інтенсивного навчання. Адже модель зіткнулася з проблемою неоднозначності. Через періодичність функції різні сегменти хвилі, наприклад, підйом у першому періоді та підйом у другому виглядають ідентично для локального енкодера. Без ширшого контексту або вищої роздільної здатності модель не змогла встановити унікальну позицію для кожного фрагмента (рис. 3.12).

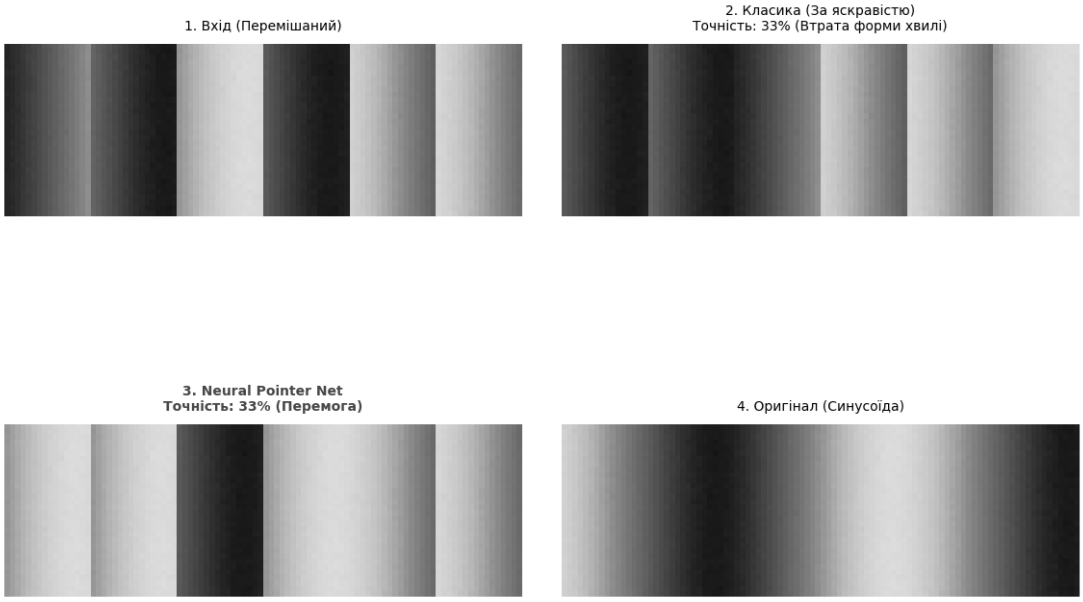

Рисунок 3.12 – Зразок «Sine Wave»

Параметри вхідних вимог подано у лістингу 3.3, табличні дані наведено у додатку З.

Лістинг 3.3 – Параметри вхідних вимог моделі «Sine Wave»

```
NUM_STRIPS = 6
H, W = 32, 16
HIDDEN = 512
EPOCH = 2000
BATCH_SIZE = 64
NUM_SAMPLES = 3000
```

кінець лістингу 3.3

Нейронна вказівна мережа виявилася нездатною до відтворення порядку елементів, що генеруються періодичною функцією синуса, через проблему неоднозначності, на відміну від успішного вирішення задачі з «мандрівною лінією».

3.6 Дослідження масштабованості та обчислювальної складності

Окремий етап дослідження присвячений аналізу поведінки моделі при збільшенні складності задачі (кількості смужок N).

Експоненційне зростання складності навчання. Порівняння динаміки навчання для патерну «Traveling Line» при N=6 та N=10 виявило нелінійну залежність ресурсних витрат (табл. 3.5).

Таблиця 3.5 – Порівняння динаміки навчання для патерну

Кількість смужок (N)	Кількість перестановок (N!)	Епохи до збіжності	Час навчання (відносно N=6)
6	720	~300	1,0
10	3628800	~1200	4,0

Зростання множини пошуку на 4 порядки призвело до збільшення часу навчання у 4 рази. Це підтверджує, що задача залишається важкою, і нейромережа лише апроксимує рішення, вимагаючи значно більше часу для стабілізації ваг у складніших множинах, для задач класу NP-hard складність $O(N^k)$.

Адаптація гіперпараметрів для N=10. Для досягнення збіжності на 10 смужках було визначено необхідність модифікації архітектури, оскільки базова конфігурація демонструвала точність лише 60 %.

Внесені зміни:

– збільшено вимір із 128 до 256, це дозволило LSTM-блоку утримувати довший контекст послідовності без втрати інформації для задачі зниклого градієнта;

– збільшено розмір датасету з 1500 до 3000 зразків для запобігання перенавчанню (overfitting) на обмеженій вибірці;

– регулювання швидкості навчання, зменшення кроку навчання до 0.0005 забезпечило стабільніший спуск градієнта.

Досліджено ефективність застосування нейромережевої архітектури Pointer Network для розв’язання задач впорядкування даних із неявними семантичними правилами сортування. Фундаментальна перевага нейромереж у семантичному впорядкуванні.

Експерименти довели, що нейромережі здатні вирішувати задачі сортування, недоступні для класичних алгоритмів:

– обмеженість класичних підходів. Алгоритми, що базуються на скалярних евристиках, такі як сортування за середньою яскравістю, виявилися ефективними лише для найпростіших патернів Linear Gradient, де яскравість корелює з позицією. У задачах зі складною геометрією («Traveling Line», «Arc Pattern») класичні методи мають результати, близькі до випадкових (0-10 %), оскільки не змогли розрізнити смуги з ідентичною середньою яскравістю;

– успіх навчання представлень: модель Neural Pointer Network досягла 100 % точності у відновленні патернів «Traveling Line» та «Arc Pattern». Це стало можливим завдяки використанню згорткового енкодера (CNN), який аналізує локальні ознаки, напрямок ліній, стики на краях та механізму уваги, що відновлює глобальну структуру зображення.

Встановлено ієрархію складності візуальних патернів, залежність швидкості та успішності навчання від типу візуального патерну:

– найнижча складність Linear Gradient. Модель збігається миттєво за 80 епох, оскільки задача зводиться до порівняння інтенсивності пікселів;

– середня складність Traveling Line, 300 епох, проте вимагає відстеження геометричної неперервності лінії;

– висока складність Arc Pattern, 500 епох, вимагає розрізнення симетричних частин, таких як нахил вліво чи нахил вправо, які статистично схожі;

– критична складність Sine Wave, спостерігається відсутність збіжності. Періодичність синусоїди створює проблему неоднозначності, коли різні сегменти хвилі виглядають ідентично для локального енкодера, що призводить до неможливості однозначного впорядкування.

Визначено технічні умови, необхідні для успішного розв’язання комбінаторних візуальних задач нейромережами:

– використання примусового навчання з коефіцієнтом 0.8 є обов’язковим. Без подачі істинних попередніх токенів під час навчання модель не здатна вийти зі стану локальних мінімумів і блукає у випадкових перестановках;

– статичний датасет, для демонстрації принципової можливості вирішення задачі ефективним є використання фіксованого набору даних, що дозволяє моделі стабілізувати ваги перед узагальненням;

– інформативність ознак: Збільшення ширини смуг (з 8 до 16 px) критично покращило роботу CNN-енкодера, надавши йому достатній контекст для розпізнавання градієнтів на стиках.

Підтверджено гіпотезу про експоненційне зростання складності задачі при лінійному збільшенні кількості елементів. Збільшення кількості смуг з 6 до 10 призвело до зростання часу навчання у 3-4 рази (з 300 до 1200 епох для «Traveling Line»). Це вказує на те, що нейромережевий підхід, попри свою гнучкість, має обмеження щодо розмірності вхідних даних без застосування додаткових евристик.

Практичні рекомендації щодо застосування Neural Pointer Networks:

– сфера застосування даної архітектури є доцільною для задач, де критерій порядку неможливо формалізувати простою математичною функцією, але він очевидний візуально або семантично, таких як відновлення документів, склеювання панорам, геномне секвенування;

– архітектурні вимоги для успішної роботи модель повинна включати вказівник ознак: CNN для зображень, Transformer/LSTM для тексту.

– стратегія навчальної програми для складних патернів, навчання слід починати зі спрощених версій з поступовим ускладненням задачі.

Робота продемонструвала, що хоча нейронні мережі поступаються класичним алгоритмам у швидкості обробки чисел, вони є безальтернативним інструментом для задач семантичного впорядкування. Здатність Pointer Networks розуміти контент і відновлювати структуру даних відкриває широкі можливості для автоматизації процесів, які раніше вимагали людської інтуїції.

ВИСНОВКИ

У кваліфікацій роботі вирішено актуальне науково-прикладне завдання дослідження ефективності нейромережевих методів у задачах впорядкування даних.

Проведено комплексний аналіз предметної області, включаючи основні категорії, характеристики та виклики традиційних алгоритмів сортування. Це підтвердило актуальність пошуку нових, більш адаптивних методів для впорядкування даних.

Розроблено алгоритм вибору параметрів штучної нейронної мережі для вирішення задачі впорядкування. Шляхом порівняльного аналізу класичних алгоритмів та архітектури Neural Pointer Network встановлено фундаментальну відмінність між алгоритмічним та семантичним сортуванням. Доведено, що класичні алгоритми є безальтернативними для даних із явними правилами порівняння завдяки швидкодії O(NlogN) та абсолютній точності. Натомість, нейромережеві підходи є доцільними виключно для даних із неявними, контекстними або геометричними критеріями впорядкування, де неможливо формалізувати функцію порівняння двох елементів.

Експериментально підтверджено перевагу архітектури Pointer Network над евристичними методами у задачах складання впорядкованого зображення, успішно відновивши глобальну структуру зображення на основі локальних ознак. У тих же умовах класичний підхід продемонстрував неспроможність, оскільки не враховував геометричну неперервність об’єктів.

Обмеження нейромереж у строгих алгоритмічних задачах: обчислювальна неефективність та складність механізму уваги $O(N^2)$ робить нейромережі на порядки повільнішими за класичні алгоритми при $N > 10$; проблема узагальнення, приякій модель не здатна ефективно екстраполювати вивчені правила на послідовності, довжина яких перевищує навчальну вибірку; при збільшенні розмірності вхідних даних простір пошуку зростає

факторіально, що унеможливлює збіжність моделі до ідеального рішення без застосування допоміжних стратегій.

Обґрунтовано вибір гібридної архітектури, що поєднує CNN-енкодер, LSTM для збереження послідовного контексту та Pointer Mechanism для роботи зі змінним словником виходу. Перевірено, що саме згортковий блок дозволяє моделі розрізняти симетричні об’єкти, лівий чи правий нахил дуги, які є статистично нерозрізнюваними для скалярних метрик.

Визначено, що успішне навчання Pointer Networks можливе лише за дотримання ряду умов: використання примусового навчання, поступове ускладнення патернів та збільшення довжини послідовності, масштабованість та межі можливостей.

Встановлено нелінійну залежність часу навчання від складності задачі. Збільшення кількості елементів сортування з 6 до 10 призвело до зростання кількості необхідних епох у 3-4 рази. Також ідентифіковано проблемні класи даних, такі як періодичні функції, наприклад, синусоїда, де локальна подібність фрагментів перешкоджає коректній роботі моделі.

Практична цінність роботи полягає у формулюванні рекомендацій щодо використання нейромереж для задач впорядкування: метод рекомендовано для систем відновлення документів, визначення повної, вичерпної послідовності та комп’ютерного зору, але не рекомендовано для задач, що вимагають суворої математичної впорядкованості числових масивів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zdolbitska N., Bas D., Zdolbitsky S. Training neural networks to sort: a new approach to classical algorithms. Trends, Issues, and Challenges in Modern Science: Proceedings of the 2nd International Scientific Conference (Cambridge, United Kingdom, 5 September 2025). Lulu Press, Inc., 2025. P. 93-95.

2. Rizvi D. Q. M., Rai H., Jaiswa R. Sorting Algorithms in Focus: A Critical Examination of SortingAlgorithm Performance. Conference: International Conference on Emerging Trends in IoT & Computing Technologies-2023At: Lucknow, India. 2023. P. 1-5.

3. A survey, discussion and comparison of sorting algorithms. URL: https://scispace.com/pdf/a-survey-discussion-and-comparison-of-sorting-algorithms-dtoju4huzr.pdf (дата звернення: 5.10.2025).

4. Мартинюк Т. Б., Круківський Б. І. Класифікаційний аналіз методів сортування. Вісник Вінницького політехнічного інституту. № 3. 2023. С. 77-83.

5. Khalafallah S., Abu-Naser S. S. AI-Driven Sorting Algorithms for Big Data: Techniques and RealWorld Applications. International Journal of Academic Engineering Research (IJAER). Vol. 9. Issue 6. 2025. P. 1-10.

6. Wibowo F. R., Faisal M. Comparative Analysis of Sorting Algorithms: TimSort Python and Classical Sorting Methods. JISA (Jurnal Informatika dan Sains). Vol. 07, No. 1. 2024. P. 11-18.

7. The Neural Network Zoo. Available online: URL: http://www.asimovinstitute.org/neural-network-zoo (дата звернення: 12.10.20245).

8. Abdolrasol M. G. M., Hussain S. M. S., Ustun T. S., Sarker M. R., Hannan M. A., Mohamed R., Ali J. A., Mekhilef S., Milad A. Artificial Neural Networks Based Optimization Techniques: A Review. 10. Electronics 2021. P. 2689.

9. Що таке нейронні мережі та де їх використовують? URL: https://incrypted.com/ua/shcho-take-nejromerezhi/ (дата звернення: 18.10.2025 р.).

9. Нейронні мережі для автоматичної класифікації даних URL: https://cpashka.biz/blog/nejronni-merezhi-dlia-avtomatychnoi-klasyfikatsii-danykh/ (дата звернення: 18.10.2025).

11. Types of Neural Networks and Definition of Neural Network. URL: https://www.mygreatlearning.com/blog/types-of-neural-networks/ (дата звернення: 19.10.2025).

12. Нейромережі у 2025 – що це і як працює. URL: https://www.site2b.ua/ua/web-blog-ua/nejromerezhi-shho-ce-i-yak-pracyuye.html (дата звернення: 23.10.2025).

13. Abdulkadirov R., Lyakhov P., Nagornov N. Survey of Optimization Algorithms in Modern Neural Networks. Mathematics. 11, 2023. P. 2466.

14. Zdolbitska N., Ostapchuk O., Lavrenchuk S., Terletskyi T., Kaidyk O., Zhyharevych O. Business information system for forecasting raw material stocks for the production of flexible packaging. 14th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece. 2024. P. 1-8.

15. Здолбіцька Н., Здолбіцький А., Давидюк Д. Машинне навчання для прогнозування попиту на товари онлайн-замовлень. Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем: матеріали II міжнар. науково-практ. конф. Луцьк, 9-11 червня 2025 р. Луцьк, 2025. С. 8-9.

16. Ayoola V. B., Ugochukwu U. N., Adeleke I., Michael C. I., Adewoye M. B., Adeyeye Y. Data-driven decision making in IT: Leveraging AI and data science for business intelligence World Journal of Advanced Research and Reviews, 23(01). 2024. P. 472-480.

17. Khalafallah S., Abu-Naser S. S. AI-Driven Sorting Algorithms for Big Data: Techniques and RealWorld Applications. International Journal of Academic Engineering Research (IJAER) ISSN: 2643-9085. Vol. 9, Issue 6. 2025. P. 1-10.

18. Розмаїтий Д. О., Грабовецький Н. В., Цвик О. С., Лемешко А. В. Штучні нейронні мережі в сучасному світі: особливості застосування. Зв'язок, № 5, 2021. С. 11-15.

19. Ткаліченко С. В. Штучні нейронні мережі: Навчальний посібник. Кривий Ріг: Державний університет економіки і технологій, 2023. 150 с.

20. pytorch. URL: https://pytorch.org (дата звернення: 24.10.2025).

ДОДАТКИ

Додаток А

Участь в міжнародній конференції

Research Europe IEDC

CERTIFICATE

№ 250905033

confirms that

Dmytro Bas

has participated in the 2nd International Scientific Conference
"Trends, Issues, and Challenges in Modern Science"

6 hours
(0.2 ECTS credits)

Executive secretary
of the conference

Tetiana Lesiuk

PRIVATE ENTREPRENEUR
2504316747
TETIANA LESIUK

September 5, 2025
Cambridge, United Kingdom

The authenticity of the certificate can be verified via the following link: https://researcheurope.org/verification

IEDC

2nd International
Scientific Conference

Trends, Issues, and Challenges in Modern Science

Proceedings

September 5, 2025

Cambridge, United Kingdom

Technical Sciences

Philological Sciences

Legal Sciences

UDC 004.8 DOI: https://doi.org/10.64076/iedc250905.18

Training neural networks to sort: a new approach to classical algorithms

Nina Zdolbitska
Lutsk National Technical University, Lutsk
https://orcid.org/0000-0002-1345-3581
Dmytro Bas
Lutsk National Technical University, Lutsk
Serhii Zdolbitskyi
Lutsk National Technical University, Lutsk

Abstract. *The paper explores a new approach to data sorting based on the use of neural networks. Traditional sorting algorithms work according to rigidly defined rules, while neural networks learn to find patterns in data, which allows them to adapt to complex and unstructured arrays. It analyzes how machine learning models can choose optimal algorithms for specific data or perform sorting independently based on experience. The advantages of such an approach are discussed, including adaptability, potential optimization, and the possibility of parallel processing. The limitations associated with computational complexity and accuracy are also considered. The conclusions indicate that neural network sorting will not replace classical methods, but opens up new prospects for solving non-standard data sorting tasks.*
Keywords*: algorithm, neural networks, data sorting, data science, business analytics.*

In the era of Big Data, where the volume of information is growing exponentially, there's a corresponding need for innovative approaches to data sorting. Traditional sorting algorithms, despite their high speed for numerical and structured data, often struggle with the non-typical tasks that arise when working with unstructured and non-numerical data. The application of neural networks is highly relevant because they can learn from examples, creating adaptive solutions that can outperform classical algorithms in specific situations. This ability to learn from data rather than following fixed rules paves the way for new, optimized algorithms, allowing for the development of more flexible and effective tools for modern computing systems.

The research aims to analyze approaches to data sorting based on neural networks and to compare their efficiency with classical algorithms.

The research's objectives are to analyze classical sorting algorithms and define their application domains, review existing machine learning models that can be applied to data sorting, and examine the architecture of a neural network capable of performing sorting operations. To do this, it is necessary to create a required dataset for training and testing the developed model and to experimentally compare the developed model's performance in terms of execution speed and accuracy against traditional sorting algorithms. The final goal is to determine which neural networks are the most suitable for sorting.

Over the last few years, research has been conducted with the goal of improving neural networks based on optimization methods from classical algorithms.

In the article [1], a review of methods for optimizing algorithms based on artificial neural networks was performed. In order to improve a neural network using optimization algorithms, one can use classical algorithms by adjusting the appropriate parameters to obtain the best template of the neural network structure for solving problems.

The authors of the research paper [2] examine a sorting algorithm based on artificial intelligence, which is adapted for processing Big Data by combining traditional and advanced methods to improve data processing. A critical comprehensive analysis of the performance of comparison-based and non-comparison-based sorting algorithms is proposed, taking into account their effectiveness and suitability for various real-world tasks [2]. An exhaustive comparative analysis is conducted to evaluate their time complexity, stability, adaptability, and memory requirements. The influence of data characteristics such as input size, distribution, and order on the algorithm's performance is also investigated.

The application of data science and artificial intelligence has become particularly useful in business analytics for making data-driven decisions. When making decisions based on modern IT technologies in data processing [3] and business analytics, for example for forecasting [4, 5], the immense capability of artificial intelligence and data science is used to create actionable insights after transforming raw data. The authors of article [3] explore the application areas, advantages, challenges, and methodologies associated with the integration of AI and data science into business analytics.

The creation of self-learning algorithms, the development of neural networks, and the improvement of various methods capable of self-learning are key in the theory of machine learning for various areas of human activity, while helping to improve the quality of products. Artificial intelligence and neural networks help programmers to write software code [7], in tasks of image or sound recognition, detection of moving objects, analysis of big data, and numerical methods in tasks of time series forecasting, etc. [8].

Traditional sorting algorithms work according to hard-coded rules, while neural networks learn to find patterns in data, which allows them to adapt to complex and unstructured arrays.

Consider the advantages of using neural networks over classical algorithms:

– adaptability, neural networks can adapt to complex, unconventional data for which it is difficult to write a simple algorithm. For example, if you need to sort objects by their visual appearance or other non-obvious characteristics, a neural network can find patterns that would be impossible to program manually;

– optimization, in certain cases, a neural network can find a more efficient way to sort than traditional algorithms. It can optimize the number of operations for specific data types, which leads to a faster process;

– parallel processing, some neural network architectures can process data in parallel, which can speed up the sorting process on suitable hardware, such as graphics processing units (GPUs).

Despite its significant advantages, this approach also has its challenges:

– training complexity: training a machine learning model requires a significant amount of data and large computational resources, which makes the process much more complex and resource-intensive than writing a classical sorting algorithm;

– accuracy and predictability: unlike classical algorithms, which always give an accurate and predictable result, a neural network can sometimes make mistakes, especially on data that is very different from the one on which it was trained;

– inefficiency for simple tasks: For simple numerical data, traditional sorting algorithms (such as Quick Sort or Merge Sort) remain unsurpassed in speed and efficiency, since they are optimized specifically for this task.

Training neural networks to sort will not replace classical algorithms in all cases, but it opens up new possibilities for solving complex, unconventional tasks where flexibility and adaptability to data are important.

References

1. Abdolrasol, M.G.M.; Hussain, S.M.S.; Ustun, T.S.; Sarker, M.R.; Hannan, M.A.; Mohamed, R.; Ali, J.A.; Mekhilef, S.; Milad, A. Artificial Neural Networks Based Optimization Techniques: A Review. Electronics 2021, 10, P. 2689. https://doi.org/10.3390/electronics10212689

2. Dr. Qaim Mehdi Rizvi, 1Harsh Rai, 1Ragini Jaiswa. Sorting Algorithms in Focus: A Critical Examination of SortingAlgorithm Performance. Conference: International Conference on Emerging Trends in IoT & Computing Technologies-2023At: Lucknow, India. 2023. P. 1-5.

3. Michael C.I., Ipede O.J., Adejumo A.D., Adenekan I.O., Adebayo D., Ojo A.S., Ayodele P. A. Data-driven decision making in IT: Leveraging AI and data science for business intelligence World Journal of Advanced Research and Reviews, 2024. 23(01). P. 472-480. DOI: 10.30574/wjarr.2024.23.1.2010.

4. N. Zdolbitska, O. Ostapchuk, S. Lavrenchuk, T. Terletskyi, O. Kaidyk and O. Zhyharevych, Business information system for forecasting raw material stocks for the production of flexible packaging, 2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2024, P. 1-8. DOI: 10.1109/DESSERT65323.2024.11122240.

5. Ніна Здолбіцька, Андрій Здолбіцький, Денис Давидюк. Машинне навчання для прогнозування попиту на товари онлайн-замовлень. Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем: матеріали II міжнар. науково-практ. конф. Луцьк, 9–11 червн. 2025 р. Луцьк, 2025. С. 8-9.

6. Shatha Khalafallah and Samy S. Abu-Naser. AI-Driven Sorting Algorithms for Big Data: Techniques and RealWorld Applications. International Journal of Academic Engineering Research (IJAER) ISSN: 2643-9085. Vol. 9 Issue 6 June – 2025. P. 1-10.

7. Здолбіцька Н. В., Жигаревич О. К., Бондарук А. С. Використання штучного інтелекту для написання програмного коду. Математика. Інформаційні технології. Освіта. Луцьк-Світязь. 31 травня – 2 червня 2024 р. С. 114-116.

8. Abdulkadirov, R.; Lyakhov, P.; Nagornov, N. Survey of Optimization Algorithms in Modern Neural Networks. Mathematics 2023, 11, 2466.

Додаток Б

Програма сортування із явними параметрами

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import numpy as np
import random
import time
import math
from typing import List, Tuple, Dict

DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class DataGenerator:
    def __init__(self, seq_len: int, min_val: float = 0.0, max_val: float = 1.0):
        self.seq_len = seq_len
        self.min_val = min_val
        self.max_val = max_val

    def generate_batch(self, batch_size: int, distribution: str = 'uniform') -> Tuple[torch.Tensor, torch.Tensor]:
        if distribution == 'uniform':
            data = np.random.uniform(self.min_val, self.max_val, size=(batch_size, self.seq_len))
        elif distribution == 'gaussian':
            data = np.random.normal(0, 1, size=(batch_size, self.seq_len))
        elif distribution == 'exponential':
            data = np.random.exponential(1.0, size=(batch_size, self.seq_len))
        else:
            raise ValueError(f"Unknown distribution: {distribution}")

        targets = np.argsort(data, axis=1)

        inputs_tensor = torch.FloatTensor(data).unsqueeze(-1).to(DEVICE)
        targets_tensor = torch.LongTensor(targets).to(DEVICE)

        return inputs_tensor, targets_tensor

class Encoder(nn.Module):
    def __init__(self, input_size: int, hidden_size: int):
        super(Encoder, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)

    def forward(self, x):
        outputs, (hidden, cell) = self.lstm(x)
        return outputs, (hidden, cell)

class Attention(nn.Module):
    def __init__(self, hidden_size: int):
        super(Attention, self).__init__()
        self.W1 = nn.Linear(hidden_size, hidden_size, bias=False)
        self.W2 = nn.Linear(hidden_size, hidden_size, bias=False)
        self.V = nn.Linear(hidden_size, 1, bias=False)

    def forward(self, encoder_outputs, decoder_hidden):
        seq_len = encoder_outputs.size(1)
        decoder_hidden_expanded = decoder_hidden.unsqueeze(1).repeat(1, seq_len, 1)
        energy = torch.tanh(self.W1(encoder_outputs) + self.W2(decoder_hidden_expanded))
        attention_scores = self.V(energy).squeeze(-1)
        return attention_scores

class PointerDecoder(nn.Module):
    def __init__(self, hidden_size: int):
        super(PointerDecoder, self).__init__()
        self.lstm = nn.LSTM(hidden_size, hidden_size, batch_first=True)
```

```
        self.attention = Attention(hidden_size)

    def forward(self, input_vector, hidden, encoder_outputs):
        lstm_out, hidden = self.lstm(input_vector.unsqueeze(1), hidden)
        pointer_logits = self.attention(encoder_outputs, lstm_out.squeeze(1))
        return pointer_logits, hidden

class NeuralSorter(nn.Module):
    def __init__(self, input_dim: int, hidden_dim: int, seq_len: int):
        super(NeuralSorter, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.encoder = Encoder(input_dim, hidden_dim)
        self.decoder = PointerDecoder(hidden_dim)
        self.start_token = nn.Parameter(torch.randn(hidden_dim))

    def forward(self, x, teacher_forcing_targets=None, teacher_forcing_ratio=0.5):
        batch_size = x.size(0)
        encoder_outputs, (hidden, cell) = self.encoder(x)
        outputs = []
        decoder_input = self.start_token.unsqueeze(0).repeat(batch_size, 1)

        for i in range(self.seq_len):
            pointer_logits, (hidden, cell) = self.decoder(decoder_input, (hidden, cell), encoder_outputs)
            outputs.append(pointer_logits)

            if teacher_forcing_targets is not None and random.random() < teacher_forcing_ratio:
                target_idx = teacher_forcing_targets[:, i]
                idx_expanded = target_idx.unsqueeze(1).unsqueeze(2).repeat(1, 1, self.hidden_dim)
                decoder_input = torch.gather(encoder_outputs, 1, idx_expanded).squeeze(1)
            else:
                pred_idx = torch.argmax(pointer_logits, dim=1)
                idx_expanded = pred_idx.unsqueeze(1).unsqueeze(2).repeat(1, 1, self.hidden_dim)
                decoder_input = torch.gather(encoder_outputs, 1, idx_expanded).squeeze(1)

        return torch.stack(outputs, dim=1)

class EarlyStopping:
    def __init__(self, patience=5, min_delta=0.001):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = float('inf')
        self.early_stop = False

    def __call__(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
        else:
            self.counter += 1
            if self.counter >= self.patience:
                self.early_stop = True

class Trainer:
    def __init__(self, model, optimizer, criterion, device):
        self.model = model
        self.optimizer = optimizer
        self.criterion = criterion
        self.device = device

    def calculate_accuracy(self, outputs, targets):
        predictions = torch.argmax(outputs, dim=2)
        row_matches = torch.all(predictions.eq(targets), dim=1)
        return row_matches.float().mean().item()

    def run_epoch(self, dataloader_func, batch_size, batches_per_epoch, is_training=True):
        if is_training:
            self.model.train()
```

```
        else:
            self.model.eval()

        total_loss = 0
        total_acc = 0

        with torch.set_grad_enabled(is_training):
            for _ in range(batches_per_epoch):
                inputs, targets = dataloader_func(batch_size, distribution='uniform')

                if is_training:
                    outputs = self.model(inputs, teacher_forcing_targets=targets, teacher_forcing_ratio=0.5)
                else:
                    outputs = self.model(inputs, teacher_forcing_targets=None)

                loss = self.criterion(outputs.view(-1, inputs.size(1)), targets.view(-1))

                if is_training:
                    self.optimizer.zero_grad()
                    loss.backward()
                    torch.nn.utils.clip_grad_norm_(self.model.parameters(), 1.0)
                    self.optimizer.step()

                total_loss += loss.item()
                total_acc += self.calculate_accuracy(outputs, targets)

        return total_loss / batches_per_epoch, total_acc / batches_per_epoch

    def train(self, data_gen, epochs, batch_size, batches_per_epoch):
        early_stopper = EarlyStopping(patience=5)

        print(f"{'='*60}")
        print(f"{'Epoch':<10} | {'Train Loss':<12} | {'Val Loss':<12} | {'Val Acc':<10} | {'Time (s)':<10}")
        print(f"{'='*60}")

        for epoch in range(epochs):
            start_time = time.time()

            train_loss, _ = self.run_epoch(data_gen.generate_batch, batch_size, batches_per_epoch, is_training=True)
            val_loss, val_acc = self.run_epoch(data_gen.generate_batch, batch_size, 20, is_training=False)

            elapsed = time.time() - start_time

            print(f"{epoch+1:<10} | {train_loss:.4f}       | {val_loss:.4f}       | {val_acc:.2%}     | {elapsed:.2f}")

            early_stopper(val_loss)
            if early_stopper.early_stop:
                print(f"!!! Early Stopping at epoch {epoch+1}")
                break

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

class Evaluator:
    def __init__(self, model, data_gen, device):
        self.model = model
        self.data_gen = data_gen
        self.device = device

    def evaluate(self, num_samples=100):
        print("\n" + "="*30)
        print(" BENCHMARK START")
        print("="*30)
```

```
        inputs_tensor, targets_tensor = self.data_gen.generate_batch(num_samples, distribution='uniform')
        arrays_np = inputs_tensor.cpu().numpy().squeeze(-1)

        results = {'NeuralNet': [], 'QuickSort': [], 'Timsort': []}

        start = time.time()
        for arr in arrays_np:
            sorted(arr)
        results['Timsort'] = time.time() - start

        start = time.time()
        for arr in arrays_np:
            quicksort(arr.tolist())
        results['QuickSort'] = time.time() - start

        self.model.eval()
        start = time.time()
        with torch.no_grad():
            outputs = self.model(inputs_tensor)
            pred_indices = torch.argmax(outputs, dim=2)
        results['NeuralNet'] = time.time() - start

        matches = torch.all(pred_indices.eq(targets_tensor), dim=1).float().mean().item()

        print(f"Sample size: {num_samples} arrays, length {self.data_gen.seq_len}")
        print("-" * 40)
        print(f"Timsort (Standard): {results['Timsort']:.5f} s")
        print(f"QuickSort (Python): {results['QuickSort']:.5f} s")
        print(f"Neural Network:     {results['NeuralNet']:.5f} s")
        print("-" * 40)
        print(f"NN Accuracy: {matches:.2%}")

if __name__ == "__main__":
    SEQ_LEN = 10
    HIDDEN_DIM = 64
    BATCH_SIZE = 32
    EPOCHS = 20

    print(f"Device: {DEVICE}")
    print(f"Sequence Length: {SEQ_LEN}")

    data_gen = DataGenerator(seq_len=SEQ_LEN)

    model = NeuralSorter(input_dim=1, hidden_dim=HIDDEN_DIM, seq_len=SEQ_LEN).to(DEVICE)

    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss()

    trainer = Trainer(model, optimizer, criterion, DEVICE)
    trainer.train(data_gen, epochs=EPOCHS, batch_size=BATCH_SIZE, batches_per_epoch=50)

    evaluator = Evaluator(model, data_gen, DEVICE)
    evaluator.evaluate(num_samples=1000)

    print("\n--- Single Example Demo ---")
    x, target = data_gen.generate_batch(1)
    model.eval()
    with torch.no_grad():
        out = model(x)
        pred_idx = torch.argmax(out, dim=2)

    original = x.cpu().numpy().flatten()
    sorted_nn_idx = pred_idx.cpu().numpy().flatten()
    sorted_arr = original[sorted_nn_idx]

    print(f"Input:       {np.round(original, 2)}")
    print(f"NN Output:   {np.round(sorted_arr, 2)}")
    print(f"Is Sorted:   {np.all(sorted_arr[:-1] <= sorted_arr[1:])}")
```

Додаток В

Сортування масиву із метрикою Partial Accuracy

Neural Sorting - тест із доданою метрикою Partial Accuracy
1. Навчання на довжині послідовності: 5 (Device: CPU)

Таблиця В.1 – Логи навчання (по епохах)

Епоха	Loss	Val Loss	Strict Acc	Partial Acc	Час (Time)
1	1.5992	1.5513	0.00%	39.34%	0.6s
2	1.3582	1.1904	0.00%	40.56%	0.4s
3	1.0598	0.9323	4.53%	57.72%	0.5s
4	0.8275	0.7591	20.78%	69.25%	0.5s
5	0.6690	0.6130	29.69%	76.41%	0.4s
6	0.5546	0.5245	35.16%	78.06%	0.5s
7	0.4916	0.4615	45.78%	83.12%	0.4s
8	0.4353	0.4495	49.06%	82.31%	0.5s
9	0.3906	0.4096	53.59%	83.91%	0.5s
10	0.3440	0.3701	57.81%	86.03%	0.4s
11	0.3200	0.3548	60.16%	86.22%	0.5s
12	0.3070	0.3438	62.03%	87.91%	0.4s
13	0.2850	0.3364	63.44%	87.25%	0.4s
14	0.2713	0.2574	71.88%	91.38%	0.4s
15	0.2575	0.2768	68.28%	89.66%	0.4s
16	0.2473	0.2404	74.38%	92.22%	0.4s
17	0.2487	0.2179	78.91%	93.16%	0.4s
18	0.2313	0.2105	79.38%	93.13%	0.4s
19	0.2374	0.2446	69.38%	90.19%	0.5s
20	0.2102	0.2207	79.06%	92.63%	0.5s
21	0.1932	0.2044	76.88%	92.69%	0.5s
22	0.2048	0.2205	69.53%	90.97%	0.5s
23	0.1948	0.2345	71.09%	90.13%	0.4s
24	0.1948	0.2149	76.09%	91.66%	0.5s
25	0.1869	0.1688	82.34%	94.31%	0.4s

2. Навчання на довжині послідовності: 10 (Device: CPU)

Таблиця В.2 – Логи навчання (по епохах)

Епоха	Loss	Val Loss	Strict Acc	Partial Acc	Час (Time)
1	2.2859	2.2154	0.00%	20.37%	1.0s
2	2.0210	1.8320	0.00%	20.00%	0.9s
3	1.6895	1.5471	0.00%	37.00%	0.9s
4	1.3825	1.2497	0.00%	48.86%	0.9s
5	1.1719	1.1312	0.16%	51.98%	0.8s
6	1.1072	1.0626	0.62%	53.86%	0.9s
7	1.0564	1.0248	0.31%	55.44%	0.9s
8	1.0046	0.9947	0.31%	56.58%	0.9s
9	0.9810	0.9478	1.09%	58.47%	0.9s
10	0.9223	0.9471	0.62%	57.20%	0.8s
11	0.8813	0.8588	2.50%	62.64%	0.8s
12	0.8567	0.8071	3.59%	66.33%	0.9s
13	0.8150	0.8107	2.19%	66.28%	0.8s
14	0.7785	0.7922	4.38%	66.09%	0.8s
15	0.7559	0.7599	2.97%	67.55%	0.8s
16	0.7308	0.7591	4.38%	67.59%	0.9s
17	0.7058	0.7239	6.56%	69.20%	0.9s
18	0.6896	0.6820	7.66%	70.95%	0.8s
19	0.6711	0.6623	9.06%	73.34%	0.8s
20	0.6520	0.6741	12.03%	71.84%	0.8s
21	0.6516	0.7259	5.78%	68.98%	0.8s
22	0.6309	0.6675	6.25%	71.92%	0.7s
23	0.6279	0.6786	10.16%	71.31%	0.8s
24	0.5837	0.6025	15.00%	75.28%	0.8s
25	0.5740	0.5888	15.47%	76.58%	0.8s

3. Навчання на довжині послідовності: 20 (Device: CPU)

Таблиця В.3 – Логи навчання (по епохах)

Епоха	Loss	Val Loss	Strict Acc	Partial Acc	Час (Time)
1	2.9798	2.9074	0.00%	10.02%	1.7s
2	2.6751	2.4710	0.00%	10.28%	1.7s
3	2.3134	2.1406	0.00%	20.06%	1.6s
4	1.9970	1.8745	0.00%	28.71%	1.7s
5	1.8050	1.7532	0.00%	29.77%	1.7s
6	1.6876	1.7756	0.00%	25.96%	1.6s
7	1.6321	1.5722	0.00%	34.90%	1.7s
8	1.6018	1.6252	0.00%	31.55%	1.7s
9	1.5256	1.5389	0.00%	36.46%	1.6s
10	1.5298	1.4755	0.00%	37.63%	1.6s
11	1.4700	1.4715	0.00%	36.95%	1.7s
12	1.4357	1.4227	0.00%	40.05%	1.6s
13	1.4215	1.4374	0.00%	38.15%	1.6s
14	1.4170	1.4698	0.00%	37.20%	1.6s
15	1.3883	1.4408	0.00%	39.30%	1.7s
16	1.3496	1.3859	0.00%	40.23%	1.7s
17	1.3289	1.6119	0.00%	32.50%	1.7s
18	1.3996	1.4789	0.00%	36.84%	1.7s
19	1.3129	1.3287	0.00%	43.02%	1.6s
20	1.2820	1.3979	0.00%	40.47%	1.6s
21	1.2867	1.3697	0.00%	41.27%	1.7s
22	1.2418	1.2824	0.00%	44.12%	1.7s
23	1.2178	1.2712	0.00%	44.21%	1.7s
24	1.1944	1.2144	0.00%	47.06%	1.7s
25	1.1667	1.2172	0.00%	46.46%	1.7s

4. Навчання на довжині послідовності: 100 (Device: CPU)

Таблиця В.4 – Логи навчання (по епохах)

Епоха	Loss	Val Loss	Strict Acc	Partial Acc	Час (Time)
1	4.5960	4.5468	0.00%	2.19%	12.2s
2	4.3409	4.1347	0.00%	2.07%	12.0s
3	3.9596	3.7989	0.00%	3.55%	12.6s
4	3.6790	3.7329	0.00%	2.93%	12.7s
5	3.4193	3.4862	0.00%	5.90%	12.9s
6	3.1680	3.7472	0.00%	2.61%	13.8s
7	3.0316	3.0891	0.00%	9.45%	13.4s
8	2.9670	2.9768	0.00%	8.76%	13.8s
9	2.8427	2.9708	0.00%	9.57%	13.8s
10	2.7921	3.7073	0.00%	4.08%	14.0s
11	2.6953	2.9322	0.00%	9.36%	13.5s
12	2.6854	3.4488	0.00%	7.02%	14.1s
13	2.5892	3.3689	0.00%	6.76%	15.8s
14	2.5349	4.6352	0.00%	3.95%	15.6s
15	2.5068	4.0251	0.00%	5.30%	15.1s
16	2.4664	2.8218	0.00%	11.03%	16.1s
17	2.5528	4.1668	0.00%	6.28%	15.8s
18	2.5610	2.9192	0.00%	10.15%	17.2s
19	2.3963	2.7715	0.00%	11.30%	18.7s
20	2.3548	2.7285	0.00%	12.48%	20.5s
21	2.3087	3.4599	0.00%	7.47%	21.2s
22	2.2801	2.8427	0.00%	11.27%	19.3s
23	2.2296	2.9209	0.00%	11.20%	18.2s
24	2.2434	3.3118	0.00%	9.20%	17.6s
25	2.2002	3.3519	0.00%	9.81%	18.8s

Додаток Д

Відновлення зображення, датасет LINEAR GRADIENT

Таблиця Д.1 – Параметри експерименту

Конфігурація	5 смужок, ширина 16px
Датасет	1500 зразків (LINEAR GRADIENT)
Режим	Інтенсивне тренування з Teacher Forcing

Таблиця Д.2 – Результати тренування

Епоха (Epoch)	Втрати (Loss)	Точність Strict	Точність Partial
010	1.6094	0.0%	20.3%
020	1.5764	0.0%	29.1%
030	1.0192	0.0%	44.7%
040	0.5432	17.2%	80.6%
050	0.1834	95.3%	98.8%
060	0.1164	90.6%	98.1%
070	0.0720	95.3%	98.1%
080	**0.0152**	**100.0%**	**100.0%**

Таблиця Д.3 – Параметри експерименту

Конфігурація	6 смужок, ширина 16px
Датасет	1500 зразків (LINEAR GRADIENT)
Режим	Інтенсивне тренування з Teacher Forcing

Таблиця Д.4 – Результати тренування

Епоха (Epoch)	Втрати (Loss)	Точність Strict	Точність Partial
010	1.7916	0.0%	16.7%
020	1.7515	0.0%	24.5%
030	1.2449	0.0%	39.8%
040	0.8813	4.7%	53.9%
050	0.7448	0.0%	56.2%
060	0.5055	26.6%	79.7%
070	0.2940	71.9%	93.0%
080	0.3831	34.4%	83.3%
090	**0.2403**	**81.2%**	**93.8%**
100	0.3827	71.9%	87.5%

Таблиця Д.5 – Параметри експерименту

Конфігурація	10 смужок, ширина 16px
Датасет	3000 зразків (LINEAR GRADIENT)
Режим	Інтенсивне тренування з Teacher Forcing

Таблиця Д.6 – Результати тренування

Епоха (Epoch)	Втрати (Loss)	Точність Strict	Точність Partial
010	2.2985	0.0%	11.1%
020	1.9093	0.0%	21.9%
030	1.4934	0.0%	27.5%
040	1.1628	0.0%	49.1%
050	1.1653	1.6%	53.0%
060	0.8799	0.0%	61.7%
070	0.7402	0.0%	63.3%
080	1.9956	0.0%	33.8%
090	1.2457	0.0%	50.0%
100	1.1069	0.0%	53.1%
110	1.0016	0.0%	53.0%
120	0.8863	0.0%	63.0%
130	0.6563	9.4%	70.5%
140	0.6300	3.1%	73.0%
150	0.4998	7.8%	78.8%
160	0.3303	32.8%	91.4%
170	0.1984	57.8%	94.7%
180	0.0905	92.2%	99.1%
190	0.0520	96.9%	99.7%
200	0.0313	96.9%	99.5%
210	**0.0289**	**100.0%**	**100.0%**

Додаток Е

Відновлення зображення, датасет TRAVELING LINE

Таблиця Е.1 – Параметри експерименту

Конфігурація	6 смужок (Traveling Line)
Датасет	2000 зразків (TRAVELING LINE)

Таблиця Е.1 – Результати тренування

Епоха (Epoch)	**Втрати (Loss)**	**Точність Strict**	**Точність Partial**
0050	1.5704	0.0%	35.4%
0100	0.8866	6.2%	69.3%
0150	0.5959	18.8%	80.7%
0200	0.3788	56.2%	89.1%
0250	0.0934	96.9%	99.5%
0300	**0.0447**	**100.0%**	**100.0%**

Таблиця Е.1 – Параметри експерименту

Конфігурація	10 смужок (Traveling Line)
Датасет	2000 зразків (TRAVELING LINE)

Таблиця Е.1 – Результати тренування

Епоха (Epoch)	**Втрати (Loss)**	**Точність Strict**	**Точність Partial**
0050	1.8225	0.0%	24.1%
0100	1.0122	0.0%	63.4%
0150	0.7858	6.2%	75.6%
0200	0.9425	0.0%	66.9%
0250	0.6298	9.4%	77.5%
0300	0.6214	0.0%	80.3%
0350	0.5479	18.8%	86.9%
0400	0.4501	21.9%	86.6%
0450	0.7086	0.0%	81.6%
0500	0.3632	28.1%	90.0%
0550	0.2076	65.6%	96.6%
0600	0.3864	21.9%	90.0%
0650	0.3080	31.2%	88.1%
0700	0.1673	56.2%	95.3%
0750	0.1118	78.1%	97.2%
0800	0.1042	81.2%	97.8%
0850	0.0775	90.6%	99.1%
0900	0.0836	93.8%	99.4%
0950	0.1072	81.2%	97.5%
1000	0.0622	96.9%	99.7%
1050	0.0416	96.9%	99.4%
1100	0.0684	93.8%	99.4%
1150	0.0395	96.9%	99.7%
1200	**0.0353**	**100.0%**	**100.0%**

Додаток Є

Відновлення зображення, датасет ARC PATTERN

Таблиця Є.1 – Параметри експерименту

Конфігурація	6 смужок (Arc Pattern)
Датасет	2000 зразків (ARC / SEMICIRCLE)

Таблиця Є.2 – Результати тренування

Епоха (Epoch)	**Втрати (Loss)**	**Точність Strict**	**Точність Partial**
0050	1.5626	0.0%	27.1%
0100	1.1716	3.1%	52.1%
0150	0.7496	34.4%	71.9%
0200	0.3192	62.5%	92.7%
0250	0.4264	50.0%	83.3%
0300	0.3962	53.1%	85.4%
0350	0.2188	78.1%	94.8%
0400	0.1327	84.4%	96.4%
0450	0.0527	96.9%	99.5%
0500	**0.0368**	**100.0%**	**100.0%**

Додаток З

Відновлення зображення, датасет SINE WAVE

Таблиця З.1 – Параметри експерименту

Конфігурація	6 смужок, ширина 16px
Датасет	3000 зразків (SINE WAVE - CLEAN)
Режим	Інтенсивне тренування (High Teacher Forcing)

Таблиця З.2 – Результати тренування

Епоха (Epoch)	Втрати (Loss)	Точність Strict	Точність Partial	LR (Learning Rate)
0050	1.7918	0.0%	17.4%	0.00100
0100	1.7914	0.0%	18.2%	0.00100
0150	1.7914	0.0%	16.7%	0.00100
0200	1.7146	0.0%	21.9%	0.00100
0250	1.7337	0.0%	20.3%	0.00100
0300	1.6996	0.0%	22.9%	0.00100
0350	1.6566	0.0%	27.3%	0.00100
0400	1.5332	0.0%	30.2%	0.00100
0450	1.5937	0.0%	24.5%	0.00100
0500	1.5578	0.0%	27.6%	0.00050
0550	1.3793	0.0%	32.8%	0.00050
0600	1.5032	0.0%	27.6%	0.00050
0650	1.5080	0.0%	27.9%	0.00050
0700	1.1795	0.0%	41.4%	0.00050
0750	1.2700	0.0%	34.1%	0.00050
0800	1.4803	0.0%	22.4%	0.00050
0850	1.6639	0.0%	19.5%	0.00050
0900	1.1296	0.0%	41.9%	0.00050
0950	1.5324	0.0%	23.7%	0.00050
1000	1.1096	0.0%	43.8%	0.00025
1050	1.4155	0.0%	25.8%	0.00025
1100	1.1139	0.0%	40.6%	0.00025
1150	1.0965	0.0%	44.0%	0.00025
1200	1.4064	0.0%	25.5%	0.00025
1250	1.4029	0.0%	27.6%	0.00025
1300	1.3794	0.0%	26.6%	0.00025
1350	1.0978	0.0%	46.1%	0.00025
1400	1.1073	0.0%	45.1%	0.00025
1450	1.3341	0.0%	31.0%	0.00025
1500	1.3465	0.0%	28.1%	0.00013
1550	1.2691	0.0%	34.1%	0.00013
1600	1.0965	0.0%	47.9%	0.00013
1650	1.0904	0.0%	46.9%	0.00013
1700	1.0854	0.0%	45.6%	0.00013
1750	1.3978	0.0%	30.5%	0.00013
1800	1.0974	0.0%	44.5%	0.00013
1850	1.3279	0.0%	29.4%	0.00013
1900	1.4407	0.0%	28.1%	0.00013
1950	1.2039	0.0%	39.8%	0.00013
2000	**1.3249**	**0.0%**	**33.1%**	**0.00006**