НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

«На правах рукопису»
УДК 004.032.26:004.932(043.3)

До захисту допущено:
В. о. завідувачки кафедри
_____Ірина ДЖИГИРЕЙ
«___»_____20___р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системи і методи штучного інтелекту» зі спеціальності 122 «Комп'ютерні науки»

на тему: «Прунінг згорткових нейронних мереж за допомогою інтерпретованості мереж Колмогорова-Арнольда»

Виконав:
студент ІІ курсу, групи КІ-41мп
Єфанов Ілля Сергійович _____

Науковий керівник:
доцент кафедри ШІ, к.т.н.,
Шаповал Наталія Віталіївна _____

Рецензент:
доцент кафедри ММСА, д.т.н.,
Савченко Ілля Олександрович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____Ірина ДЖИГИРЕЙ
«29» серпня 2025 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Єфанов Іллі Сергійовичу

1. Тема роботи «Прунінг згорткових нейронних мереж за допомогою інтерпретованості мереж Колмогорова-Арнольда», науковий керівник дисертації Шаповал Наталія Віталіївна, доцент кафедри ШІ, затверджені наказом по НН ІПСА від «06» листопада 2025 р. № 4837-с.
2. Термін подання студентом роботи 13.12.2025.
3. Об'єкт дослідження: процес стиснення та оптимізації глибоких згорткових нейронних мереж для зменшення обчислювальної складності
4. Вихідні дані до роботи: набір даних для класифікації зображень CIFAR-10.
5. Перелік завдань, які потрібно розробити: аналіз сучасних методів прунінгу та математичних основ мереж Колмогорова-Арнольда; розробка алгоритму структурного стиснення CNN з використанням інтерпретованості KAN-шарів; програмна реалізація гібридної архітектури та проведення експериментів на наборі даних CIFAR-10; порівняльна оцінка ефективності методу за точністю та ступенем стиснення.

6. Перелік ілюстративного матеріалу: використані метрики, опис набору даних, принцип роботи програмного продукту, результати прогнозування, схеми роботи алгоритмів.
7. Орієнтовний перелік публікацій: заплановано апробувати роботу «Прунінг згорткових нейронних мереж за допомогою інтерпретованості мереж Колмогорова-Арнольда» авторів Єфанова І.С., Шаповал Н.В. на IV ВНПК «Системні науки та інформатика» у грудні 2025 р, а також заплановано опублікувати однойменну статтю у журналі “Artificial intelligence” у грудні 2025 р, яку вже подано до друку.
8. Дата видачі завдання: 30 серпня 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури за темою	01.09.2025 – 07.09.2025	Виконано
2.	Підготовка першого розділу	08.09.2025 – 01.10.2025	Виконано
3.	Підготовка другого розділу	02.10.2025 – 24.10.2025	Виконано
4.	Розробка програмного продукту	25.10.2025 – 25.11.2025	Виконано
5.	Підготовка третього розділу	12.11.2025 – 08.12.2025	Виконано
6.	Підготовка четвертого розділу	01.12.2025 – 09.12.2025	Виконано
7.	Оформлення розділів відповідно до нормоконтролю	01.12.2025 – 09.12.2025	Виконано
8.	Підготовка презентації доповіді	05.12.2025 – 06.12.2025	Виконано
9.	Оформлення дипломної роботи	01.12.2025 – 09.12.2025	Виконано

Студент Ілля ЄФАНОВ

Керівник Наталія ШАПОВАЛ

РЕФЕРАТ

Дипломна робота – 140 с., 17 табл., 2 рис., додаток, 35 джерел.

НЕЙРОННІ МЕРЕЖІ, CNN, ПРУНІНГ, КОМПРЕСІЯ МОДЕЛЕЙ, МЕРЕЖІ КОЛМОГОРОВА-АРНОЛЬДА, KAN, B-СПЛАЙНИ, EDGE AI

Тема: Прунінг згорткових нейронних мереж за допомогою інтерпретованості мереж Колмогорова-Арнольда.

У роботі розглянуто проблему надмірної параметризації сучасних глибоких нейронних мереж та проаналізовано існуючі методи їх стиснення, зокрема прунінг та дистиляцію знань. Запропоновано новий підхід до структурного прунінгу, що базується на використанні інтерпретованості мереж Колмогорова-Арнольда (KAN).

Об’єкт дослідження: процес оптимізації згорткових нейронних мереж шляхом структурного прунінгу.

Предмет дослідження: метод KAN-керованого структурного прунінгу, заснований на аналізі функціональної важливості компонентів мережі.

Мета роботи: розробка та дослідження методу структурного прунінгу CNN, який використовує аналіз важливості ознак, отриманий з інтерпретованого KAN-шару («bottleneck»), для зменшення обчислювальної складності моделі при збереженні точності.

В ході виконання роботи досліджено математичні основи KAN та теорію B-сплайнів. Розроблено гібридну архітектуру CNN-KAN та алгоритм ітеративного прунінгу, де критерій видалення фільтрів базується на аналізі коефіцієнтів сплайнів функцій активації. Програмна реалізація виконана мовою Python з використанням бібліотек PyTorch та efficient-kan. Ефективність методу перевірено на наборі даних CIFAR-10, продемонстровано переваги семантично обґрунтованого прунінгу над класичними евристичними підходами.

ABSTRACT

Thesis – 140 p., 17 tables, 2 figures, appendix, 35 sources.

NEURAL NETWORKS, CNN, PRUNING, MODEL COMPRESSION, KOLMOGOROV-ARNOLD NETWORKS, KAN, B-SPLINES, EDGE AI.

Topic: Pruning of Convolutional Neural Networks using Kolmogorov-Arnold Network Interpretability.

The thesis addresses the issue of over-parameterization in modern deep neural networks and analyzes existing compression methods, specifically pruning and knowledge distillation. A novel approach to structural pruning based on the interpretability of Kolmogorov-Arnold Networks (KAN) is proposed.

Object of research: the process of optimizing convolutional neural networks through structural pruning.

Subject of research: the method of KAN-guided structural pruning based on the analysis of the functional importance of network components.

Purpose of the work: to develop and investigate a structural pruning method for CNNs that utilizes feature importance analysis derived from an interpretable KAN bottleneck layer to reduce computational complexity while maintaining accuracy.

During the research, the mathematical foundations of KAN and B-spline theory were investigated. A hybrid CNN-KAN architecture and an iterative pruning algorithm were developed, where the filter removal criterion is based on the analysis of spline coefficients of activation functions. The software implementation was carried out in Python using PyTorch and efficient-kan libraries. The method's effectiveness was validated on the CIFAR-10 dataset, demonstrating the advantages of semantically grounded pruning over classical heuristicapproaches.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1 ОСНОВИ ТА СУЧАСНІ МЕТОДИ СТИСНЕННЯ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ 11
1.1 Актуальність задачі стиснення нейронних мереж 11
1.1.1 Високі обчислювальні вимоги 11
1.1.2 Енергоспоживання та екологічний вплив 12
1.1.3 Економічні фактори 13
1.1.4 Обмеження Edge-пристроїв 14
1.2 Таксономія сучасних методів стиснення нейронних мереж 16
1.3 Сучасні стратегії застосування прунінгу. 23
1.3.1 Одноразовий прунінг (One-shot Pruning) 23
1.3.2 Ітеративний прунінг (Iterative Pruning) 25
1.4 Гіпотеза лотерейного квитка 27
1.5 Дистиляція знань 28
1.6 Постановка задачі 30
Висновки до розділу 1 30
РОЗДІЛ 2 АНАЛІЗ МЕРЕЖ КОЛМОГОРОВА–АРНОЛЬДА ТА ПІДХОДІВ KAN-КЕРОВАНОГО ПРУНІНГУ 32
2.1 Теорема представлення Колмогорова-Арнольда: математичні основи 32
2.1.1 Історичний контекст та тринадцята проблема Гільберта 32
2.1.2 Формулювання теореми Колмогорова-Арнольда 33
2.1.3 Математичні властивості та обмеження теореми 34
2.1.4 Від теореми до практичної архітектури KAN 35
2.2 Математична теорія B-сплайнів 36
2.2.1 Визначення та мотивація B-сплайнів 36
2.2.2 Вузловий вектор (Knot Vector) 37

2.2.3 Рекурсивне визначення B-сплайнів (формула Кокса-де Бура) 38
2.2.4 Властивості B-сплайнів 39
2.2.5 Апроксимація функцій за допомогою B-сплайнів 40
2.2.6 Обчислення B-сплайнів: алгоритм де Бура 42
2.3 Архітектура Мереж Колмогорова-Арнольда (KAN) 43
2.3.1 Формальне визначення KAN-шару 43
2.3.2 Багатошарова архітектура KAN 45
2.4 Порівняльний аналіз архітектур MLP та KAN 46
2.4.1 Фундаментальні відмінності архітектур 46
2.4.2 Детальне порівняння властивостей 47
2.4.3 Закони масштабування (Scaling Laws) 47
2.5 Метод KAN-керованого прунінгу 48
2.5.1 Гібридна архітектура CNN-KAN 49
2.5.2 Критерій важливості фільтрів на основі KAN 50
2.5.3 Алгоритм ітеративного KAN-керованого прунінгу 52
Висновки до розділу 2 54
РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ 56
3.1 Опис експериментальної установки 56
3.1.1 Датасет CIFAR-10 56
3.1.2 Предобробка даних та аугментація 58
3.1.3 Апаратне та програмне забезпечення 59
3.1.4 Архітектури нейронних мереж для тестування 60
3.1.5 Гіперпараметри та протокол навчання 65
3.2 Параметри прунінгу 68
3.2.1 Конфігурація KAN-шару 68
3.2.2 Варіанти прунінгу 69
3.2.3 Baseline методи 70
3.3 Результати експериментів 70
3.3.1 Загальний огляд 70
3.3.2 Порівняння на рівних коефіцієнтах 71

3.3.3 Аналіз за типами архітектур 72
3.4 Детальний аналіз 73
3.4.1 Ефективність KAN vs Magnitude 73
3.4.2 Knowledge Distillation як верхня межа 74
3.4.3 Парето-фронт для ResNet-18 75
3.4.4 Аналіз важливості фільтрів 76
3.5 Обчислювальна ефективність 77
3.5.1 Час виконання 77
3.5.2 Пам'ять (GPU VRAM) 77
3.5.3 FLOPS 78
3.6 Обмеження та виклики 78
3.6.1 Обмеження KAN-методу 79
3.6.2 Порівняння з Distillation 79
3.6.3 Майбутні напрямки 80
Висновки до розділу 3 81
РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЄКТУ 83
4.1 Опис ідеї проекту та обґрунтування актуальності 83
4.2 Аналіз ринкового середовища та конкурентів 85
4.3 Бізнес-модель та монетизація 88
4.4 Технологічний аудит та roadmap розвитку продукту 91
4.5 Go-to-market стратегія та канали дистрибуції 94
4.6 Команда та організаційна структура 96
4.7 Фінансовий план та інвестиційні потреби 98
4.8 Ризики та стратегії їх мітигації 101
Висновки до розділу 4 103
ВИСНОВКИ 106
ПЕРЕЛІК ПОСИЛАНЬ 109
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ 113

ВСТУП

Стрімкий[1] розвиток технологій штучного інтелекту (ШІ) за останнє десятиліття докорінно змінив підходи до розв’язання складних задач у сферах комп’ютерного зору, обробки природної мови та аналізу даних. Сучасні глибокі нейронні мережі (Deep Neural Networks, DNN) демонструють вражаючі результати, часто перевершуючи можливості людини у задачах класифікації та розпізнавання образів. Проте, ця еволюція супроводжується експоненціальним зростанням складності моделей: архітектури, що встановлюють нові рекорди точності (SOTA), часто налічують сотні мільйонів або навіть мільярди параметрів.

Така тенденція до "надмірної параметризації" створює суттєвий бар'єр для практичного впровадження передових алгоритмів. Високі вимоги до обчислювальної потужності та обсягів пам'яті обмежують використання сучасних нейромереж переважно потужними серверними кластерами. Водночас, існує гостра потреба у розгортанні інтелектуальних систем безпосередньо на кінцевих пристроях (Edge AI) – смартфонах, вбудованих системах, дронах та IoT-датчиках, де ресурси жорстко лімітовані, а енергоефективність є критичним фактором.

Для вирішення проблеми ресурсоємності активно розробляються методи стиснення нейронних мереж, серед яких ключове місце посідає прунінг (pruning) – процедура видалення надлишкових зв'язків або нейронів. Традиційні методи прунінгу зазвичай спираються на евристичні критерії, найпоширенішим з яких є абсолютна величина ваг. Цей підхід базується на припущенні, що ваги з меншою амплітудою роблять менший внесок у

[1]Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

вихідний сигнал мережі. Однак такий метод є по суті "сліпим": він не враховує функціональну роль нейрона в контексті всієї мережі та семантичне значення ознак, які він кодує, що може призводити до втрати важливої інформації при високих ступенях стиснення.

Поява нового класу архітектур – мереж Колмогорова-Арнольда (Kolmogorov-Arnold Networks, KAN) – відкриває принципово нові можливості для оптимізації. На відміну від класичних багатошарових перцептронів, де функції активації фіксовані, KAN використовують навчані функції активації на ребрах графа, параметризовані B-сплайнами. Ця особливість забезпечує високу інтерпретабельність моделі: кожен зв'язок можна візуалізувати та проаналізувати як конкретну математичну функцію. Це породжує гіпотезу, що саме інтерпретабельність KAN може стати основою для нового, більш досконалого критерію прунінгу.

Актуальність теми дослідження обумовлена необхідністю створення ефективних методів автоматичної оптимізації нейронних мереж, які б враховували не лише статистичні характеристики ваг, а й семантичну важливість ознак. Поєднання потужності згорткових мереж (CNN) у виділенні візуальних патернів з інтерпретабельністю KAN дозволяє створити гібридний підхід до стиснення, що забезпечує високу точність при значному зменшенні обчислювальної складності. Це має вирішальне значення для розвитку енергоефективного штучного інтелекту та наближення передових технологій до реальних застосувань.

Наукова новизна одержаних результатів. Запропоновано новий метод визначення важливості фільтрів у CNN, який, на відміну від існуючих евристик, використовує аналіз коефіцієнтів сплайнів у KAN-bottleneck шарі, що дозволяє зберегти найбільш семантично значущі ознаки при стисненні мережі.

РОЗДІЛ 1 ОСНОВИ ТА СУЧАСНІ МЕТОДИ СТИСНЕННЯ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ

1.1 Актуальність задачі стиснення нейронних мереж

Сучасна епоха розвитку методів глибокого навчання (Deep Learning) характеризується безпрецедентним зростанням складності архітектур нейронних мереж. Ця тенденція, яку часто називають «гонкою озброєнь» у сфері штучного інтелекту, призвела до появи моделей, що демонструють надлюдську точність у широкому спектрі задач – від комп'ютерного зору до обробки природної мови. Такі архітектури, як ResNet, VGG, Inception, а також новітні трансформерні моделі, досягають високих показників якості, проте ціною цього успіху стає колосальне зростання кількості параметрів, що обчислюються мільйонами, а в окремих випадках – мільярдами.

Явище «перепараметризації» (over-parameterization), коли кількість параметрів моделі значно перевищує кількість навчальних даних або необхідну складність для вирішення задачі, створює низку критичних викликів для індустрії. Ці виклики можна умовно розділити на три групи: обчислювальні, екологічні та економічні.

1.1.1 Високі обчислювальні вимоги

Сучасні моделі глибокого навчання характеризуються безпрецедентним зростанням складності та розміру. Це зростання відбувається експоненціально: якщо класична архітектура AlexNet (2012) містила близько 60 мільйонів параметрів, то VGG-19 (2014) [17] вже мала 144 мільйони параметрів. ResNet-152 (2015) [16] містив близько 60 мільйонів параметрів, але зі значно більшою глибиною – 152 шари проти 19 у VGG.

Сучасні моделі, такі як Vision Transformers (ViT) або Swin Transformers, можуть містити понад 300 мільйонів параметрів.

У галузі обробки природної мови масштаби ще більш вражаючі. GPT-3 має 175 мільярдів параметрів, що вимагає близько 350 ГБ пам'яті лише для зберігання моделі у форматі float32. Деякі експериментальні моделі досягають трильйонів параметрів, що робить їх доступними лише для найбільших технологічних корпорацій з відповідною інфраструктурою. Тренування таких моделей вимагає потужних обчислювальних кластерів. Для прикладу, навчання GPT-3 зайняло приблизно 3640 petaflop-днів обчислень, що еквівалентно 355 GPU-рокам на NVIDIA V100. Вартість такого тренування оцінюється у діапазоні від 4 до 12 мільйонів доларів США лише за обчислювальні ресурси, не враховуючи витрати на електроенергію, охолодження та підтримку інфраструктури. Це створює значний бар'єр входу для академічних установ та малих компаній. Дослідження 2020 року показало, що 98% публікацій у топових конференціях з машинного навчання походять від авторів, які мають доступ до значних обчислювальних ресурсів через великі корпорації або добре фінансовані університети. Це явище, відоме як "computational divide", загрожує демократизації досліджень у галузі штучного інтелекту.

1.1.2 Енергоспоживання та екологічний вплив

Екологічний вплив тренування великих моделей стає все більш актуальною проблемою. Дослідження Strubell та співавторів (2019) показало, що тренування великої трансформер-моделі з neural architecture search може генерувати близько 284 тонн CO_2 – еквівалент п'яти середніх автомобілів протягом їх повного життєвого циклу, включаючи виробництво.

Детальний аналіз енергоспоживання виявляє наступні цифри:

- тренування BERT-base (110M параметрів): приблизно 1507 фунтів CO_2;

- тренування GPT-2 (1.5B параметрів): приблизно 552 тонни CO_2;
- тренування GPT-3 з усіма експериментами: за оцінками понад 1200 тонн CO_2.

Для порівняння, середній громадянин США генерує близько 16 тонн CO_2 на рік. Таким чином, одне тренування GPT-3 еквівалентне річним викидам 75 американців. Центри обробки даних, що використовуються для тренування та розгортання ШІ-моделей, споживають величезну кількість електроенергії. За даними International Energy Agency, у 2020 році центри обробки даних споживали близько 200 TWh електроенергії – приблизно 1% від світового споживання електроенергії. З зростанням застосування ШІ ця частка може зрости до 3-8% до 2030 року. Охолодження серверного обладнання також вимагає значних ресурсів. Типові співвідношення Power Usage Effectiveness (PUE) для центрів обробки даних становлять 1.5-2.0, що означає, що на кожен ватт, спожитий обчислювальним обладнанням, витрачається ще 0.5-1.0 ватт на охолодження та інфраструктуру. Крім того, виробництво спеціалізованого обладнання (GPU, TPU) має свій екологічний слід. Виробництво одного високопродуктивного GPU може генерувати 100-200 кг CO2, а враховуючи короткий цикл оновлення обладнання (2-3 роки), екологічний вплив значно зростає.

1.1.3 Економічні фактори

Економічні наслідки надмірної параметризації моделей є багатогранними. Для компаній, що надають сервіси на основі ШІ, вартість інференсу може становити значну частину операційних витрат. Розглянемо конкретні приклади:

Google перекладач обробляє понад 100 мільярдів слів щодня. Якщо припустити, що кожен запит потребує одного forward pass через велику

модель, навіть мінімальне покращення ефективності може призвести до економії мільйонів доларів на рік. За оцінками, зменшення латентності на 10% або зменшення розміру моделі на 20% може заощадити компанії розміром Google десятки мільйонів доларів щорічно лише на електроенергії. Для стартапів та малих компаній вартість може бути ще більш критичною. Типова вартість запуску одного екземпляру p3.8xlarge (4×V100 GPU) на AWS становить близько $12 за годину, або $8,760 за місяць безперервної роботи. Для компанії, що розгортає модель для обслуговування клієнтів 24/7, це може становити понад $100,000 на рік лише за обчислювальні ресурси для однієї моделі. Латентність та пропускна здатність також мають пряму економічну цінність. Дослідження Amazon показали, що кожні 100 мс додаткової затримки призводять до зменшення продажів на 1%. Для великих платформ електронної комерції це може означати мільйони доларів втрачених доходів. Крім прямих витрат на обчислення, існують супутні економічні фактори:

- вартість зберігання моделей (особливо при версіонуванні та A/B тестуванні);
- витрати на bandwidth для передачі великих моделей;
- вартість обслуговування та оновлення інфраструктури;
- необхідність найму висококваліфікованих спеціалістів для роботи з великими моделями

1.1.4 Обмеження Edge-пристроїв

Паралельно з розвитком великих моделей зростає попит на розгортання ШІ безпосередньо на кінцевих пристроях (Edge AI). Ця тенденція обумовлена кількома факторами: Приватність та безпека даних: Багато застосувань, таких як медична діагностика, розпізнавання обличь для

розблокування пристроїв, або обробка фінансових даних, вимагають локальної обробки без передачі чутливої інформації на сервери. Відповідність до регуляцій на кшталт GDPR часто вимагає мінімізації передачі персональних даних. Надійність та доступність: Застосування у критичній інфраструктурі (автономні транспортні засоби, медичне обладнання, промислові системи контролю) не можуть покладатися на наявність стабільного інтернет-з'єднання. Локальний інференс забезпечує функціонування навіть при відсутності мережі. Латентність у реальному часі: Деякі застосування вимагають екстремально низької латентності (менше 10-50 мс), що важко досягти при необхідності передачі даних на віддалені сервери. Для прикладу: - Автономне водіння: потребує реакції менше 100 мс - AR/VR застосування: потребують латентності менше 20 мс для уникнення motion sickness - Промислова робототехніка: часто потребує реакції менше 10 мс. Для масових споживчих пристроїв вартість постійної передачі даних на сервери може бути значною, особливо в регіонах з дорогим мобільним інтернетом. Однак Edge-пристрої мають жорсткі апаратні обмеження:

Пам'ять: Типовий смартфон має 4-8 GB RAM, але більша частина зайнята операційною системою та іншими застосунками. Для моделі машинного навчання може залишатися лише 100-500 MB. Мікроконтролери для IoT часто мають лише 256 KB - 2 MB RAM.

Обчислювальна потужність: Мобільні процесори та GPU мають продуктивність на 1-2 порядки нижчу за серверні GPU:

- NVIDIA A100 (серверний): ~312 TFLOPS (FP16);
- Apple A15 Bionic (смартфон): ~2.5 TFLOPS;
- Qualcomm Snapdragon 888: ~1.5 TFLOPS;
- Raspberry Pi 4: ~0.1 GFLOPS (CPU).

Енергоспоживання: Інференс моделі не повинен швидко розряджати батарею. Енергоефективність вимірюється у GFLOPS/Watt і є критичним параметром. Типові смартфони мають батареї ємністю 3000-5000 mAh (~11-19 Wh), і навіть 1W постійного споживання на ШІ-обчислення значно

скоротить час автономної роботи.

Розмір пристрою та охолодження: На відміну від серверних систем з активним охолодженням, мобільні пристрої мають обмежені можливості відведення тепла. Тривалі інтенсивні обчислення призводять до thermal throttling – зниження тактової частоти для запобігання перегріву, що ще більше знижує продуктивність.

Ці обмеження роблять неможливим пряме розгортання сучасних state-of-the-art моделей на Edge-пристроях. Наприклад, GPT-3 з його 175B параметрів та 350GB розміру не може бути завантажений навіть у пам'ять найпотужніших смартфонів, не кажучи вже про виконання інференсу у розумні терміни.

Саме тому методи стиснення моделей стають критично важливими для реалізації Edge AI. Сучасні підходи до стиснення включають квантизацію [28], прунінг [7, 8, 15], дистиляцію знань [11] та архітектурні оптимізації [18, 19]. Ефективне стиснення дає змогу:

- зменшити розмір моделі до меж, прийнятних для обмеженої пам'яті;
- зменшити кількість обчислень (FLOPS) для прискорення інференсу;
- знизити енергоспоживання для продовження часу автономної роботи;
- зберегти прийнятну точність для практичного застосування.

1.2 Таксономія сучасних методів стиснення нейронних мереж

Методи стиснення нейронних мереж можна класифікувати за різними критеріями. Розглянемо основні категорії детальніше, включаючи їх

теоретичне обґрунтування, практичні аспекти застосування та порівняльні характеристики.

Прунінг (Pruning) – видалення надлишкових елементів. Прунінг є одним з найпопулярніших методів стиснення нейронних мереж. Його основна ідея запозичена з нейробіології, де процес синаптичного прунінгу в мозку людини видаляє невикористовувані нейронні з'єднання під час розвитку, оптимізуючи роботу нервової системи. У дорослої людини відбувається видалення до 50% синапсів, що були присутні у дитячому віці, що призводить до більш ефективної та спеціалізованої нейронної мережі. Детальний огляд сучасного стану методів прунінгу наведено у роботі [15]. Неструктурний прунінг (Unstructured Pruning) передбачає видалення окремих ваг з матриць ваг незалежно від їхнього розташування в архітектурі. Результатом є розріджена (sparse) матриця з багатьма нульовими елементами. Піонерські роботи в цій галузі [7, 8] показали можливість досягнення високих рівнів розрідженості без значної втрати точності. Математична формалізація: Нехай W є матрицею ваг шару. Неструктурний прунінг створює маску M того ж розміру:

$$M_{ij} = \begin{cases} 1, \text{якщо } w_{ij} \text{ залишається}, \\ 0, \text{якщо } w_{ij} \text{ видаляється}. \end{cases} \quad (1.1)$$

Обрізана матриця ваг визначається як:

$$W_{pruned} = M \odot W \quad (1.2)$$

де $\odot$ позначає поелементне множення (Hadamard product).

Рівень розрідженості (sparsity level) s визначається як:

$$s = \frac{\sum_{i,j} \mathbb{1}[M_{ij} = 0]}{|M|} \quad (1.3)$$

де |M| – загальна кількість елементів у матриці.

Методи визначення маски M, представлено нижче.

1. Прунінг за величиною (Magnitude-based pruning).

Найпростіший підхід – видаляти ваги з найменшою абсолютною величиною. Для заданого рівня розрідженості s, визначається поріг τ такий, що:

$$M_{ij} = \mathbb{1}\left[\left|w_{ij}\right| > \tau\right] \tag{1.4}$$

де τ обирається так, щоб забезпечити бажаний рівень *s*.

2. Прунінг за градієнтом (Gradient-based pruning).

Використовується інформація про градієнти для оцінки важливості ваг:

$$I\left(w_{ij}\right) = \left|\frac{\partial L}{\partial w_{ij}} \cdot w_{ij}\right| \tag{1.5}$$

де *L* – функція втрат. Ваги з найменшими значеннями *I* видаляються.

3. Прунінг за гессіаном (Hessian-based pruning).

Більш точна оцінка впливу видалення ваги використовує другу похідну.

$$I\left(w_{ij}\right) = \frac{1}{2} H_{ij} w_{ij}^2 \tag{1.6}$$

де H_{ij} – діагональний елемент матриці Гессе функції втрат.

Класичні методи Optimal Brain Damage [13] та Optimal Brain Surgeon [14] використовують інформацію про гессіан для визначення важливості ваг. OBD припускає діагональну структуру гессіану, тоді як OBS використовує повну матрицю для більш точної оцінки.

Переваги неструктурного прунінгу:

- висока гнучкість: можна досягти дуже високих рівнів

розрідженості (до 90-95% ваг можна видалити) зі збереженням прийнятної точності;

- точкова оптимізація: можна селективно видаляти найменш важливі ваги незалежно від їх розташування;
- мінімальна втрата точності: при правильному підході та поступовому прунінгу втрата точності може бути менше 1%;
- теоретично обґрунтований: існують теореми про апроксимаційні властивості розріджених мереж.

Недоліки неструктурного прунінгу:

- нерегулярна структура розрідженості: нульові елементи розташовані хаотично у матриці, що ускладнює ефективну апаратну реалізацію;
- відсутність реального прискорення на стандартному обладнанні: стандартні бібліотеки лінійної алгебри (BLAS, cuBLAS) оптимізовані для щільних матриць і не використовують розрідженість;
- потреба у спеціалізованому апаратному забезпеченні: реальне прискорення можна отримати лише на спеціалізованих процесорах (наприклад, NVIDIA Sparse Tensor Cores, що з'явилися у архітектурі Ampere);
- Overhead на зберігання: для представлення розрідженої матриці у форматах CSR (Compressed Sparse Row) або COO (Coordinate format) потрібно зберігати індекси ненульових елементів, що може нівелювати економію пам'яті при низьких рівнях розрідженості;
- ускладнення процесу навчання: необхідність підтримки масок та спеціальних оптимізаторів.

Формати зберігання розріджених матриць. Для матриці розміром $m \times n$ з k ненульовими елементами:

- повне зберігання: m × n елементів;

- CSR формат: k значень + k індексів стовпців + (m+1) покажчиків рядків = 2k + m + 1;
- COO формат: k значень + k індексів рядків + k індексів стовпців = 3k.

Економія пам'яті досягається лише якщо $k \ll m \times n$.

Структурний прунінг видаляє цілі структурні блоки: окремі нейрони, канали (фільтри в згорткових мережах), групи нейронів, attention heads у трансформерах, або навіть цілі шари. Результатом є менша, але щільна архітектура без розрідженості.

Типи структурного прунінгу.

1. Filter pruning (прунінг фільтрів) у згорткових мережах.
   Згортковий шар має тензор ваг W розміром$[C_{out}, C_{in}, K_h, K_w]$, де C_{out} – кількість вихідних каналів (фільтрів), C_{in} – кількість вхідних каналів, K_h, K_w – розміри ядра згортки.
   Прунінг фільтрів видаляє деякі з C_{out} фільтрів. Якщо видалити n_p фільтрів, новий тензор має розмір $[C_{out} - n_p, C_{in}, K_h, K_w]$.
2. Channel pruning (прунінг каналів).
   Видалення вхідних каналів, що еквівалентно видаленню фільтрів у попередньому шарі. Якщо поточний шар має розмір $[C_{out}, C_{in}, K_h, K_w]$ і попередній шар має C_{in} вихідних фільтрів, то видалення каналів змінює розмір на $[C_{out}, C_{in} - n_p, K_h, K_w]$.
3. Neuron pruning у повнозв'язних шарах.
   Для шару з матрицею ваг $W \in R^{n_{out} \times n_{in}}$, видалення n_p нейронів зменшує розмір до $[n_{out} - n_p, n_{in}]$.
4. Layer pruning.
   Видалення цілих шарів з мережі. Особливо ефективно для дуже глибоких архітектур (ResNet, Transformer) з residual connections, де можна видалити цілі residual блоки.

Attention head pruning у трансформерах:

Multi-head attention має H голів. Прунінг видаляє найменш важливі голови, зменшуючи обчислювальну складність.

Математична формалізація для прунінгу фільтрів. Для i-го фільтра F_i розміром $[C_{in}, K_h, K_w]$, обчислюється критерій важливості $S(F_i)$.

Найпоширеніші критерії:

L1-норма:

$$S(F_i) = \left||F_i|\right|_1 = \sum_{c=1}^{C_{in}} \sum_{h=1}^{K_h} \sum_{w=1}^{K_w} \left|w_{i,c,h,w}\right| \tag{1.7}$$

L2-норма:

$$S(F_i) = \left||F_i|\right|_2 = \sqrt{\sum_{c=1}^{C_{in}} \sum_{h=1}^{K_h} \sum_{w=1}^{K_w} w_{i,c,h,w}^2} \tag{1.8}$$

Після обчислення $S(F_i)$ для всіх фільтрів, вони сортуються, і n_p фільтрів з найменшими значеннями видаляються.

Переваги структурного прунінгу:

- реальне прискорення: зменшена архітектура має менше обчислень (FLOPS), що призводить до реального прискорення на будь-якому обладнанні (CPU, GPU, мобільних пристроях);
- зменшення реального розміру моделі: немає потреби зберігати маски або індекси, розмір моделі прямо пропорційний кількості параметрів;
- простота імплементації: стандартні фреймворки (PyTorch, TensorFlow) природно підтримують моделі зі змінною кількістю каналів;
- сумісність з апаратним прискоренням: оптимізовані бібліотеки

(cuDNN, MKL-DNN) працюють ефективно з будь-якими розмірами каналів;

- зменшення споживання пам'яті під час інференсу: менше активацій потрібно зберігати.

Недоліки структурного прунінгу:

- менша гнучкість: видалення цілого фільтра є більш грубою операцією порівняно з видаленням окремих ваг;
- нижчі коефіцієнти стиснення: зазвичай можна досягти 30-60% стиснення без значної втрати точності, тоді як неструктурний прунінг може досягти 80-95%;
- потенційно більша втрата точності: при однаковому рівні стиснення структурний прунінг зазвичай призводить до більшої деградації точності;
- складність вибору правильної гранулярності: не завжди очевидно, чи краще видаляти фільтри, канали, чи цілі шари;
- міжшарові залежності: видалення фільтрів в одному шарі вимагає коригування вхідних каналів у наступному шарі.

Порівняння впливу на продуктивність. Для ілюстрації, розглянемо згортковий шар з параметрами:

- вхід: $224 \times 224 \times 3$;
- фільтри: 64 канали, ядро 3×3;
- вихід: $224 \times 224 \times 64$ (з padding);

Кількість операцій (FLOPS):

FLOPS = $2 \times K_h \times K_w \times C_{in} \times C_{out} \times H \times W$

FLOPS = $2 \times 3 \times 3 \times 3 \times 64 \times 224 \times 224 \approx 174M$

Після структурного прунінгу 50% фільтрів (залишилося 32): $FLOPS_{pruned} \approx 87M$.

При неструктурному прунінгу 50% ваг без спеціалізованого обладнання: $\text{FLOPS}_{sparse} \approx 174M$ (без прискорення на стандартному

обладнанні).

Гібридні підходи. Деякі сучасні методи комбінують структурний та неструктурний прунінг.

1. Спочатку застосовується структурний прунінг для видалення цілих фільтрів/каналів.
2. Потім застосовується неструктурний прунінг до ваг, що залишилися.
3. Це дозволяє отримати як реальне прискорення (від структурного), так і високі коефіцієнти стиснення (від неструктурного).

1.3 Сучасні стратегії застосування прунінгу.

Окрім вибору критерію важливості та гранулярності, важливим є питання про стратегію застосування прунінгу: коли видаляти параметри, скільки параметрів видаляти за один раз, та як відновлювати точність після прунінгу.

1.3.1 Одноразовий прунінг (One-shot Pruning)

При одноразовому прунінгу мережа спочатку повністю навчається до досягнення високої точності на навчальному датасеті. Після цього, за один крок, видаляється бажаний відсоток параметрів або структурних елементів на основі обраного критерію важливості. Після прунінгу модель може бути донавчена (fine-tuned) протягом невеликої кількості епох для часткового відновлення точності. Алгоритм одноразового прунінгу:

Вхід: навчальний датасет $\mathcal{D}_{train}$, валідаційний датасет $\mathcal{D}_{val}$, цільовий рівень розрідженості s_{target}.

1. Навчити щільну мережу до збіжності:

$$M_0 \to M_{trained}.$$

2. Обчислити критерій важливості I(e) для всіх елементів $e \in$ {ваги, нейрони, фільтри}.

3. Відсортувати елементи за зростанням важливості:

$$e_{\sigma(1)}, e_{\sigma(2)}, \dots, e_{\sigma(N)}, \text{ де } I\left(e_{\sigma(1)}\right) \le I\left(e_{\sigma(2)}\right) \le \cdots \le I\left(e_{\sigma(N)}\right).$$

4. Видалити $n_p = \lfloor s\_\{target\} \cdot N \rfloor$ найменш важливих елементів:

$$M_{trained} \to M_{pruned}.$$

5. (Опціонально) Донавчити обрізану модель протягом E_{ft} епох:

$$M_{pruned} \to M_{final}.$$

6. Оцінити точність на валідаційному датасеті:

$$acc_{final} = \text{evaluate}\left(M_{final}, \mathcal{D}_{val}\right).$$

Переваги одноразового прунінгу:

- швидкість: прунінг виконується за один прохід, загальний час значно менший ніж при ітеративному підході;
- простота реалізації: не потребує складного циклу з повторним прунінгом та навчанням;

- передбачуваність: можна точно контролювати кінцевий рівень стиснення;
- не потребує багаторазового перенавчання: економія обчислювальних ресурсів;
- легко інтегрується у існуючі пайплайни: можна застосувати до будь-якої вже навченої моделі.

Недоліки одноразового прунінгу:

- при високих коефіцієнтах прунінгу $(s_{target} > 0.5)$ може призводити до значної деградації точності;
- мережа не має можливості поступово адаптуватися до зменшеної архітектури під час навчання;
- донавчання може не повністю відновити точність, особливо якщо було видалено критично важливі елементи;
- важко знайти оптимальний рівень s_{target} занадто агресивний прунінг може бути незворотнім;
- критерій важливості обчислюється один раз на повністю навченій моделі і може не відображати динаміку навчання.

Емпіричні спостереження показують, що одноразовий прунінг працює прийнятно для помірних рівнів стиснення (до 40-50%), але при більш агресивному прунінгу ітеративний підхід дає значно кращі результати.

1.3.2 Ітеративний прунінг (Iterative Pruning)

Ітеративний прунінг розбиває процес видалення параметрів на кілька етапів. На кожній ітерації видаляється невеликий відсоток найменш важливих елементів, після чого мережа донавчається протягом кількох епох. Цей процес повторюється до досягнення цільового рівня розрідженості або

стиснення, або до моменту, коли подальший прунінг призводить до неприйнятної деградації точності. Алгоритм ітеративного прунінгу:

Вхід: початкова навчена модель M_0, кількість ітерацій N, відсоток прунінгу на ітерації p_k, кількість епох донавчання E

1. Ініціалізація: $M_{(\text{current}\leftarrow)} M_0$.

2. Для k = 1, 2, ..., N:

   a) обчислити критерій важливості для поточної моделі M_{k-1}: $I(e)$ для всіх елементів e;

   b) видалити $p_k\%$ найменш важливих елементів $M_{k-1} \rightarrow p_{k\backslash}\%M'_k$;

   c) донавчити модель протягом E епох;

   d) оцінити точність $\text{acc}_\text{k} = \text{evaluate}(\text{M}_\text{k}, \mathcal{D}_{val})$.

3. Результат: фінальна обрізана модель M_N.

Стратегії визначення p_k (відсоток прунінгу на ітерації).

1. Постійний відсоток: $p_k = p = \text{const}$.

2. Експоненціальний розклад для досягнення цільової розрідженості s_{target}: $s_k = s_{target} + \left(s_0 - s_{target}\right)\left(1 - \frac{k}{N}\right)^3$, де s_0 – початкова розрідженість (зазвичай 0), s_k – розрідженість після k-ї ітерації. Відсоток прунінгу на k-й ітерації:$p_k = \frac{s_k - s_{k-1}}{1 - s_{k-1}}$

Переваги ітеративного прунінгу:

- значно краща фінальна точність при високих коефіцієнтах стиснення (60-90%);
- мережа має змогу поступово адаптуватися до нової архітектури через багаторазове донавчання;
- критерій важливості переобчислюється на кожній ітерації, враховуючи поточний стан моделі;
- можливість "відновлення" від помилкових рішень про прунінг на ранніх етапах: деякі ваги, що були малими на початку, можуть стати важливими після адаптації;
- більш гнучкий контроль над процесом: можна зупинитися на

будь-якій ітерації при досягненні бажаного компромісу між розміром та точністю;

- менший ризик катастрофічного забування порівняно з одноразовим прунінгом.

Недоліки ітеративного прунінгу:

- значно вища обчислювальна вартість: потребує багаторазового донавчання, загальний час може бути у N разів довшим за одноразовий прунінг;
- необхідність налаштування додаткових гіперпараметрів: кількість ітерацій N, відсоток прунінгу p_k на кожній ітерації, кількість епох донавчання E;
- складніша реалізація: потребує організації циклу прунінгу-донавчання;
- потребує більше пам'яті для зберігання проміжних моделей (якщо потрібна історія або можливість відкату);
- не завжди очевидно, коли зупинитися: потрібні критерії зупинки.

1.4 Гіпотеза лотерейного квитка

Однією з найважливіших теоретичних розробок останніх років у галузі прунінгу є гіпотеза лотерейного квитка (Lottery Ticket Hypothesis, LTH), сформульована Frankle та Carbin [9] у 2019 році. Основне твердження: Будь-яка щільна, випадково ініціалізована нейронна мережа містить розріджену підмережу (так званий "виграшний лотерейний квиток"), яка при навчанні з тими самими початковими вагами може досягти порівнянної або кращої точності, ніж повна мережа. Формально, нехай $f(x; \theta_0)$ – випадково ініціалізована мережа з параметрами θ_0. Після навчання отримуємо θ_T. LTH стверджує, що існує бінарна маска $m \in \{0,1\}^n$ така, що:

$$\left|\left|m\right|\right|_0 \ll n \quad \text{та} \quad \mathcal{A}\big(f(x; m \odot \theta_T)\big) \geq \mathcal{A}\big(f(x; \theta_T)\big), \tag{1.9}$$

де $\mathcal{A}$ – метрика точності, ϵ – мала допустима деградація.

Ключовий момент: важлива не тільки структура підмережі (маска m), але й конкретні початкові значення ваг θ_0. Якщо ту саму маску застосувати до нових випадкових ваг, продуктивність різко падає. Алгоритм пошуку "виграшних квитків" (Iterative Magnitude Pruning) показано нижче.

1. Випадково ініціалізувати мережу: θ_0 та зберегти $\theta_{\text{init}} \leftarrow \theta_0$.

2. Для k = 1, ..., N ітерацій:

- навчити мережу: $\theta_T \leftarrow \text{Train}(m \odot \theta_{\text{init}})$;
- видалити p% ваг з найменшою величиною: оновити маску *m*;
- скинути ваги до початкових: $\theta_0 \leftarrow m \odot \theta_{\text{init}}$.

Пізніші дослідження [10] показали, що для глибоких мереж краще використовувати перемотку до раннього чекпойнту (після кількох епох) замість повернення до початкової ініціалізації.

1.5 Дистиляція знань

Дистиляція знань [11] є альтернативним підходом до стиснення моделей. На відміну від прунінгу, що видаляє частини існуючої моделі, дистиляція навчає нову, меншу модель (учень) імітувати поведінку великої моделі (вчитель). Основна ідея: Вчитель передає свої "знання" учневі не лише через правильні відповіді (hard labels), але й через повний розподіл ймовірностей (soft labels), який містить цінну інформацію про взаємозв'язки між класами.

Математична формалізація: для входу *x* вчитель генерує логіти *z_T*.

Softmax з температурою τ: $q_i = \frac{\exp(z_{T,i}/\tau)}{\sum_{j=1}^{C}\exp(z_{T,j}/\tau)}$

Роль температури τ:

- при $\tau = 1$: стандартний softmax;
- при $\tau > 1$: розподіл стає більш "м'яким", розкриваючи відносини між класами;
- зазвичай $\tau \in [3,20]$.

Функція втрат для учня:

$$\mathcal{L}_{total} = \alpha \cdot \mathcal{L}_{\text{hr}} + (1-\alpha) \cdot \mathcal{L}_{\text{sf}}, \tag{1.10}$$

де

- $\mathcal{L}_{\text{hr}} = \text{CrossEntropy}\big(y_{\text{true}}, \sigma(z_S)\big)$ – навчання на справжніх мітках;
- $\mathcal{L}_{\text{sf}} = \tau^2 \cdot \text{KL}\big(\sigma(z_T/\tau)|\sigma(z_S/\tau)\big)$ – імітація вчителя;
- $\alpha \approx 0.1$ (типове значення) – баланс між навчанням та імітацією.

Множник τ^2 компенсує зменшення величини градієнтів при високій температурі.

Типи дистиляції.

1. Response-based: Учень імітує фінальні виходи вчителя (класичний підхід).
2. Feature-based: Учень відтворює проміжні представлення (активації внутрішніх шарів) внутрішніх шарів)внутрішніх шарів)внутрішніх шарів): $\mathcal{L}_{\text{faue}(\ell)} = |h_T^{(l)}(x) - g\left(h_S^{(l)}(x)\right)|_2^2$.
3. Relation-based: Учень навчається відношенням між активаціями через Gram matrices або попарні відстані.

Переваги дистиляції:

- архітектурна гнучкість: учень може мати принципово іншу архітектуру (ResNet → MobileNet [18]);
- можна дистилювати з ансамблю моделей;

- краща генералізація: м'які мітки діють як регуляризація;
- не потребує модифікації існуючої моделі-вчителя.

Недоліки:

- потреба у вже навченій великій моделі;
- необхідність повного тренування учня з нуля;
- додаткові гіперпараметри (\tau, \alpha);
- залежність від "capacity gap" між вчителем та учнем (учень повинен мати 10-20% параметрів вчителя).

1.6 Постановка задачі

Мережі Колмогорова-Арнольда (KAN) [1, 30], завдяки навчаним одновимірним функціям на ребрах, пропонують унікальну інтерпретабельність, що дозволяє аналізувати функціональну важливість компонентів мережі. Ці архітектури базуються на класичній теоремі представлення Колмогорова-Арнольда [3, 4], яка показує можливість представлення багатовимірних функцій через композицію одновимірних.

Це мотивує перехід до другого розділу, присвяченого теоретичним та математичним основам KAN та розробці методу KAN-керованого структурного прунінгу згорткових нейронних мереж.

Висновки до розділу 1

У першому розділі проведено детальний аналіз актуальності задачі стиснення нейронних мереж, що зумовлена зростаючою складністю сучасних архітектур, високими обчислювальними вимогами, значним екологічним впливом та обмеженнями Edge-пристроїв.

Розглянуто таксономію методів прунінгу з ключовими відмінностями між структурним та неструктурним підходами. Структурний прунінг забезпечує реальне прискорення на стандартному обладнанні, тоді як неструктурний досягає вищих коефіцієнтів стиснення, але вимагає спеціалізованого апаратного забезпечення. Проаналізовано різні критерії важливості (величина ваг, градієнти, гессіан) та стратегії застосування (одноразовий, ітеративний прунінг).

Гіпотеза лотерейного квитка змінила парадигму розуміння надмірної параметризації, показавши що великі мережі містять "виграшні підмережі" з вдалою комбінацією архітектури та ініціалізації. Дистиляція знань запропонувала альтернативний підхід через навчання компактної моделі-учня на виходах великої моделі-вчителя.

Проведене дослідження виявило, що традиційні методи прунінгу базуються на евристичних критеріях, які не враховують функціональну роль компонентів у контексті всієї архітектури. Це створює перспективний напрямок – використання інтерпретабельності для створення семантично обґрунтованих критеріїв прунінгу.

РОЗДІЛ 2 АНАЛІЗ МЕРЕЖ КОЛМОГОРОВА–АРНОЛЬДА ТА ПІДХОДІВ KAN-КЕРОВАНОГО ПРУНІНГУ

2.1 Теорема представлення Колмогорова-Арнольда: математичні основи

У 1900 році на Міжнародному конгресі математиків у Парижі Давід Гільберт представив список з 23 нерозв'язаних проблем, які визначили розвиток математики у XX столітті. Тринадцята проблема Гільберта стосувалася можливості розв'язання загального рівняння сьомого степеня за допомогою неперервних функцій двох змінних та композиції функцій.

2.1.1 Історичний контекст та тринадцята проблема Гільберта

У більш загальному формулюванні проблема ставила питання: чи можна будь-яку неперервну функцію багатьох змінних представити як композицію неперервних функцій меншої кількості змінних, зокрема, функцій однієї або двох змінних? Це питання мало фундаментальне значення для розуміння структури багатовимірних функцій та можливостей їх обчислення.

Гільберт висловив гіпотезу, що певні функції семи змінних не можуть бути представлені через композицію функцій двох змінних. Ця гіпотеза виявилася помилковою: у 1957 році Володимир Ігорович Арнольд, тоді 19-річний студент Андрія Миколайовича Колмогорова, довів, що будь-яка неперервна функція n змінних може бути представлена суперпозицією неперервних функцій трьох змінних. Пізніше Колмогоров удосконалив цей результат, показавши, що достатньо функцій однієї змінної та операції додавання.

2.1.2 Формулювання теореми Колмогорова-Арнольда

Теорема представлення Колмогорова-Арнольда [3, 4] (Kolmogorov-Arnold Representation Theorem, KART) є одним з найбільш елегантних та несподіваних результатів у теорії функцій.

Теорема (Колмогоров-Арнольд, 1957). Нехай $f: [0,1]^n \to R$ є довільною неперервною функцією n змінних. Тоді існують неперервні одновимірні функції $\phi_{q,p}: [0,1] \to R$ та $\Phi_q: R \to R$ такі, що для будь-якого $x = (x_1, x_2, \dots, x_n) \in [0,1]^n$:

$$f(x_1, x_2, \dots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{2.1}$$

Це означає, що будь-яка неперервна функція n змінних може бути точно представлена як суперпозиція (2n+1) функцій однієї змінної Φ_q, де кожна з них застосовується до суми n одновимірних функцій $\phi_{q,p}$. Ключові властивості теореми представлено нижче.

1. Універсальність внутрішніх функцій: Функції $\phi_{q,p}$ можуть бути обрані незалежно від f. Існує універсальний набір з n(2n+1) одновимірних функцій, який підходить для представлення будь-якої неперервної функції n змінних.
2. Залежність зовнішніх функцій: Функції Φ_q залежать від конкретної функції f, яку потрібно представити. Вся специфіка функції f "закодована" у цих зовнішніх функціях.
3. Фіксована кількість доданків: Незалежно від складності функції f, завжди потрібно рівно 2n+1 зовнішніх функцій. Ця кількість не залежить від "складності" або "гладкості" функції.

4. Єдина багатовимірна операція: Теорема показує, що єдиною справжньою багатовимірною операцією є додавання: $\sum_{p=1}^{n} \phi_{q,p}(x_p)$. Всі нелінійні перетворення виконуються одновимірними функціями.

2.1.3 Математичні властивості та обмеження теореми

Негладкість внутрішніх функцій. Основним обмеженням теореми для практичного застосування є те, що універсальні внутрішні функції $\phi_{q,p}$, хоча й неперервні, є недиференційовними майже скрізь. Вони мають фрактальну природу та демонструють "дику" поведінку. Формально, функції $\phi_{q,p}$ не є абсолютно неперервними і не мають похідної у класичному розумінні у більшості точок. Це робить їх апроксимацію та навчання за допомогою градієнтних методів надзвичайно складними завданнями. Оригінальне доведення Колмогорова є конструктивним – воно надає явний спосіб побудови функцій $\phi_{q,p}$. Ці функції мають наступну структуру:

$$\phi_{q,p}(x_p) = \sum_{k=1}^{\infty} \lambda^k \psi_q(3^k x_p) \tag{2.2}$$

де ψ_q – спеціальна періодична функція з періодом $1, \lambda \in (0,1)$ – параметр, що забезпечує збіжність ряду. Однак ця конструкція є непрактичною для обчислень:

- нескінченний ряд потребує обрізання, що вносить похибку;
- періодична функція ψ_q має складну структуру;

- параметри λ та базові функції ψ_q важко визначити для конкретних застосувань.

Теорема може бути узагальнена з одиничного гіперкуба $[0,1]^n$ на довільній компактній підмножині R^n за допомогою неперервних біективних відображень (гомеоморфізмів). Для довільної компактної множини $K \subset R^{n\backslash}$ існує гомеоморфізм $h: K \to [0,1]^n$. Тоді для будь-якої неперервної функції $f: K \to R$ можна визначити $\tilde{f} = f \circ h^{-1}: [0,1]^n \to R$ і застосувати теорему.

2.1.4 Від теореми до практичної архітектури KAN

Мережі Колмогорова-Арнольда (KAN) [1, 30] не намагаються точно відтворити конструкцію з теореми. Натомість, вони використовують теорему як філософське натхнення для створення нової архітектури нейронних мереж з ключовими відмінностями.

1. ***Гладкі апроксимації***. Замість негладких універсальних функцій $\phi_{q,p}$, KAN використовують гладкі B-сплайни або інші диференційовні базисні функції. Це дозволяє застосовувати градієнтні методи оптимізації.

2. ***Навчані функції***. Усі функції на ребрах є навчаними параметрами моделі, а не фіксованими універсальними функціями з теореми.

3. ***Багатошарова архітектура***. KAN узагальнюють ідею на довільну кількість шарів, тоді як теорема стосується лише двошарової структури.

4. ***Відсутність строгої відповідності***. KAN не гарантують точного представлення будь-якої функції (як теорема), але надають потужну та гнучку архітектуру для апроксимації.

Теорема Колмогорова-Арнольда показує фундаментальну можливість розкладу багатовимірних залежностей на композицію одновимірних функцій. Це надихнуло на створення архітектури, де складність міститься у навчаних одновимірних функціях на ребрах, а не у фіксованих нелінійностях на вузлах.

2.2 Математична теорія B-сплайнів

B-сплайни (базисні сплайни) є фундаментальним інструментом у чисельному аналізі, комп'ютерній графіці та апроксимації функцій. У контексті KAN, B-сплайни використовуються для параметризації навчаних одновимірних функцій на ребрах мережі, забезпечуючи баланс між гнучкістю, гладкістю та обчислювальною ефективністю.

2.2.1 Визначення та мотивація B-сплайнів

Визначення сплайну степеня k: Сплайн степеня k на інтервалі [a, b] – це кусково-поліноміальна функція, яка складається з поліномів степеня не більше k на кожному підінтервалі та є (k-1) разів неперервно диференційовною в точках з'єднання (вузлах). Формально, для вузлового вектора $t_0 < t_1 < \cdots < t_m$ на [a, b], сплайн $S(x)$ степеня k задовольняє: - На кожному інтервалі $[t_i, t_{i+1}]: S(x)$ є поліномом степеня $\leq k$ - У внутрішніх вузлах $t_i (i = 1, \dots, m-1): S(x) \in C^{k-1}$, тобто має неперервні похідні до порядку k-1 включно Переваги сплайнів для апроксимації:

- локальність: Зміна в одній частині не впливає на віддалені частини функції;
- гладкість: Контрольована диференційовність;
- стабільність: Добре обумовлені обчислення без осциляцій Рунге;
- ефективність: Швидке обчислення та диференціювання.

2.2.2 Вузловий вектор (Knot Vector)

Вузловий вектор $t = (t_0, t_1, \dots, t_m)$ визначає точки, в яких з'єднуються окремі поліноміальні шматки сплайну. Структура вузлового вектора впливає на властивості B-сплайнів.

Типи вузлових векторів.

1. Рівномірний вузловий вектор: $t_i = t_0 + i \cdot h, \quad h = \frac{t_m - t_0}{m}$. Вузли рівновіддалені один від одного. Найпростіший випадок для обчислень.
2. Нерівномірний вузловий вектор: Вузли розташовані нерівномірно, що дозволяє краще адаптуватися до локальних особливостей функції. Більша щільність вузлів у областях з високою кривизною.
3. Відкритий (clamped) вузловий вектор: Перші (k+1) та останні (k+1) вузлів співпадають з кінцями інтервалу: $t_0 = \dots = t_k < t_{k+1} < \dots < t_{m-k-1} < t_{m-k} = \dots = t_m$. Це забезпечує, що сплайн інтерполює кінцеві точки інтервалу.

Якщо вузол t_i повторюється r разів (має кратність r), гладкість сплайну в цій точці знижується до C^{k-r}. Наприклад, для кубічного сплайну (k=3):

- кратність 1: C^2 (дві неперервні похідні);
- кратність 2: C^1 (лише перша похідна неперервна);
- кратність 3: C^0 (лише неперервність функції).

2.2.3 Рекурсивне визначення B-сплайнів (формула Кокса-де Бура)

B-сплайни $B_{i,k}(x)$ степеня k визначаються рекурсивно за допомогою елегантної формули Кокса-де Бура [5, 6]:

База рекурсії (степінь 0):

$$B_{i,0}(x) = \begin{cases} 1, & \text{якщо } t_i \le x \le t_{i+1}, \\ 0, & \text{інакше.} \end{cases} \tag{2.3}$$

Це кусково-сталі функції (індикаторні функції інтервалів). Рекурсивний крок (степінь k > 0):

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x) \tag{2.4}$$

Якщо знаменник дорівнює нулю ($t_{i+k} = t_i$ для кратних вузлів), відповідний доданок вважається рівним нулю:

$$\frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) := 0 \quad \text{якщо } t_{i+k} = t_i \tag{2.5}$$

Формула Кокса-де Бура має геометричну інтерпретацію як зважена комбінація двох B-сплайнів нижчого порядку. Ваги є лінійними функціями від x:

$$w_1(x) = \frac{x - t_i}{t_{i+k} - t_i}, \quad w_2(x) = \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} \tag{2.6}$$

Легко перевірити, що $w_1(x) + w_2(x) = 1$ при правильних умовах, що забезпечує гладке "змішування" між сусідніми базисними функціями.

2.2.4 Властивості B-сплайнів

B-сплайни мають низку важливих властивостей, що робить їх ідеальними для використання в KAN.

1. Локальна підтримка (Local Support). B-сплайн $B_{i,k}(x)$ степеня k відмінний від нуля лише на інтервалі $[t_i, t_{i+k+1}]$:

$$B_{i,k}(x) = 0 \quad \text{для } x \notin [t_i, t_{i+k+1}] \tag{2.7}$$

Ця властивість означає, що зміна коефіцієнта одного B-сплайну впливає на апроксимовану функцію лише локально, у межах (k+1) вузлових інтервалів. Важливість для KAN:

- ефективність обчислень: розрідженість якобіану та гессіану;
- стійкість до катастрофічного забування: зміни в одній області не впливають глобально;
- можливість локальної адаптації: різні частини функції можуть навчатися незалежно.

2. Невід'ємність (Non-negativity). $B_{i,k}(x) \geq 0 \quad \forall x$. Усі B-сплайни є невід'ємними функціями. Це випливає безпосередньо з рекурсивного визначення.

3. Розбиття одиниці (Partition of Unity). Для будь-якого $x \in [t_k, t_{m-k}]$ сума всіх B-сплайнів степеня k дорівнює одиниці: $\sum_i B_{i,k}(x) = 1$. Ця властивість забезпечує:

- числову стабільність апроксимації;
- афінну інваріантність: лінійні перетворення даних не змінюють форму апроксимації;
- природну інтерпретацію коефіцієнтів як зважених внесків.

4. Гладкість. B-сплайн степеня k є (k-1) разів неперервно диференційовним у внутрішніх (некратних) вузлах: $B_{i,k}(x) \in C^{k-1}$.

5. Лінійна незалежність. Множина B-сплайнів $B_{i,k}(x)_{i=0}^{n}$ на даному вузловому векторі є лінійно незалежною. Це гарантує однозначність представлення функції через B-сплайни.

6. Стабільність базису. B-сплайни формують добре обумовлений базис для простору сплайнів. Числа обумовленості матриць, побудованих з B-сплайнів, залишаються помірними навіть для великих степенів k, на відміну від високостепеневих поліномів у стандартній формі (степеневий базис). Для порівняння, матриця Вандермонда для степеневих поліномів має число обумовленості, що зростає експоненціально з степенем.

7. Варіаційне зменшення (Variation Diminishing Property). Сплайн-крива не має більше перетинів з будь-якою гіперплощиною, ніж контрольний полігон (ламана, що з'єднує контрольні точки). Це означає, що B-сплайн апроксимація не вносить штучних осциляцій.

2.2.5 Апроксимація функцій за допомогою B-сплайнів

Будь-яку гладку функцію f(x) на інтервалі [a, b] можна апроксимувати як лінійну комбінацію B-сплайнів: $S(x) = \sum_{i=0}^{n} c_i B_{i,k}(x)$, де c_i – коефіцієнти (контрольні точки), які потрібно визначити, а $n = m - k - 1$ – кількість базисних функцій для вузлового вектора з m+1 вузлами. Для апроксимації функції $f(x)$ на основі набору точок $(x_j, y_j)_{j=1}^{N}$, коефіцієнти c_i знаходяться шляхом мінімізації квадратичної помилки: $\min_c \sum_{j=1}^{N} \left(y_j - \sum_{i=0}^{n} c_i B_{i,k}(x_j)\right)^2$.

У матричній формі: $\min_c ||y - Bc||_2^2$, де $B_{ji} = B_{i,k}(x_j)$ – матриця базисних функцій розміром $N \times (n+1)$. Беручи похідну по c та

прирівнюючи до нуля:

$$\frac{\partial}{\partial c}\left|\left|y - Bc\right|\right|_2^2 = -2B^T(y - Bc) = 0 \tag{2.8}$$

Отримуємо нормальні рівняння: $B^T Bc = B^T y$, якщо B має повний ранг (що зазвичай виконується), розв'язок: $c = (B^T B)^{-1} B^T y$

Властивості апроксимації:

- похибка апроксимації зменшується при збільшенні кількості вузлів;
- для гладких функцій похибка має порядок $\mathrm{O}(\mathrm{h}^{\mathrm{k+1}})$, де h – максимальна відстань між вузлами;
- локальність B-сплайнів робить матрицю B^{T} B розрідженою, що дозволяє ефективно розв'язувати систему.

Інтерполяція vs Апроксимація. При інтерполяції вимагаємо $S(x_j) = y_j$ для всіх точок даних. Це призводить до системи:

$$\sum_{i=0}^{n} c_i B_{i,k}(x_j) = y_j, \quad j = 0, \ldots, n \tag{2.9}$$

У матричній формі: $Bc = y$, де B – квадратна матриця. При апроксимації (N > n+1) маємо переви значену систему і використовуємо найменші квадрати для знаходження найкращого наближення в сенсі L^2.

2.2.6 Обчислення B-сплайнів: алгоритм де Бура

Для ефективного обчислення значення сплайну:

$$S(x) = \sum_{i=0}^{n} c_i B_{i,k}(x) \tag{2.10}$$

в точці x використовується алгоритм де Бура [5] – стабільний чисельний метод, що узагальнює алгоритм де Кастельжо для кривих Безьє.

Алгоритм де Бура.

Вхід:

- точка x, в якій потрібно обчислити $S(x)$;
- коефіцієнти {c_i}_{i=0}^{n};
- вузловий вектор {t_i}_{i=0}^{m};
- степінь k.

Крок 1: Знайти інтервал, що містить x, знайти такий, що $t_l \le x < t_{l+1}$.

Крок 2: Ініціалізація: $d_i^{[0]} = c_i \quad$ для $i = l-k, \ldots, l$.

Крок 3: Рекурсивне обчислення: для $r = 1, \ldots, k$; для $i = l-k+r, \ldots, l$:

$$\alpha_{i,r} = \frac{x - t_i}{t_{i+k-r+1} - t_i},$$

$$d_i^{[r]} = \left(1 - \alpha_{i,r}\right) d_{i-1}^{[r-1]} + \alpha_{i,r} d_i^{[r-1]}.$$

Крок 4: Результат: $S(x) = d_l^{[k]}$.

Алгоритм має складність $O(k^2)$, де k – степінь сплайну. Для фіксованого k (зазвичай k=3) це O(1) на кожну точку обчислення. Алгоритм де Бура є численно стабільним завдяки опуклим комбінаціям ($\alpha \in [0,1]$) на

кожному кроці, що запобігає накопиченню похибок округлення. Алгоритм можна інтерпретувати як ітеративну процедуру опуклих комбінацій контрольних точок. На кожному рівні рекурсії обчислюються проміжні точки, які поступово наближаються до фінального значення сплайну в точці x.

2.3 Архітектура Мереж Колмогорова-Арнольда (KAN)

2.3.1 Формальне визначення KAN-шару

Розглянемо KAN-шар, що відображає вектор з n_{in} входів на вектор з n_{out} виходів. KAN-шар з входом $x = (x_1, \dots, x_{n_{in}}) \in R^{n_{in}}$ та виходом $y = (y_1, \dots, y_{n_{out}}) \in R^{n_{out}}$ визначається як:

$$y_j = \sum_{i=1}^{n_{in}} \phi_{j,i}(x_i), \quad j = 1, \dots, n_{out} \tag{2.11}$$

де кожна функція $\phi_{j,i}: R \to R$ є навчаною одновимірною функцією, параметризованою B-сплайнами. Кожна функція $\phi_{j,i}$ представляється як комбінація базисної функції та сплайну: $\phi_{j,i}(x) = w_{b,ji} \cdot b(x) + w_{s,ji} \cdot \text{spline}_{ji}(x)$, де $b(x)$ – базова функція активації (наприклад, $SiLU$: $b(x) = x \cdot \sigma(x)$), що забезпечує стабільність навчання на початкових етапах; $w_{b,ji}, w_{s,ji} \in R$ – навчані скалярні ваги для балансування компонентів; $\text{spline}_{ji}(x) = \sum_{l=0}^{G+k-1} c_{ji,l} B_{l,k}(x)$ – B-сплайн з коефіцієнтами $c_{ji,l}$.

Вибір базової функції:

- SiLU (Swish): $b(x) = x \cdot \sigma(x) = \frac{x}{1+e^{-x}}$;

- GELU: b(x) = $x \cdot \Phi(x)$, де Φ – кумулятивна функція розподілу стандартного нормального розподілу;
- лінійна: $b(x) = x$;

Базова функція забезпечує розумну поведінку на початку навчання, коли коефіцієнти сплайну ще не оптимізовані. Кількість параметрів:

Для KAN-шару з n_{in} входами та n_{out} виходами:

- кількість B-сплайн базисних функцій на одну функцію ϕ_{ji}: $G + k$, (де G – кількість інтервалів grid, k – степінь);
- коефіцієнтів сплайну: $G + k$;
- додаткових скалярних ваг: 2 (w_b та w_s);
- параметрів на одну функцію: $G + k + 2$;
- загальна кількість параметрів шару: $n_{in} \times n_{out} \times (G + k + 2)$.

Для порівняння:

- лінійний шар (MLP): $n_{in} \times n_{out} + n_{out}$(ваги + bias);
- KAN має більше параметрів через гнучкість функцій активації, але часто досягає кращої точності при меншому загальному розмірі мережі.

Типові значення гіперпараметрів:

- G = 5 (5 інтервалів, 6 вузлів);
- k = 3 (кубічні сплайни);
- кількість параметрів на функцію: 5 + 3 + 2 = 10.

2.3.2 Багатошарова архітектура KAN

Повна KAN-мережа складається з послідовності KAN-шарів:

$$x^{(0)} \to \Phi^{(1)}x^{(1)} \to \Phi^{(2)}x^{(2)} \to \Phi^{(3)} \dots \to \Phi^{(L)}x^{(L)} \quad (2.12)$$

де $x^{(0)} \in R^{n_0}$ – вхід мережі, $x^{(L)} \in R^{n_L}$ – вихід мережі, $\Phi^{(l)}$ – l-й KAN-шар.

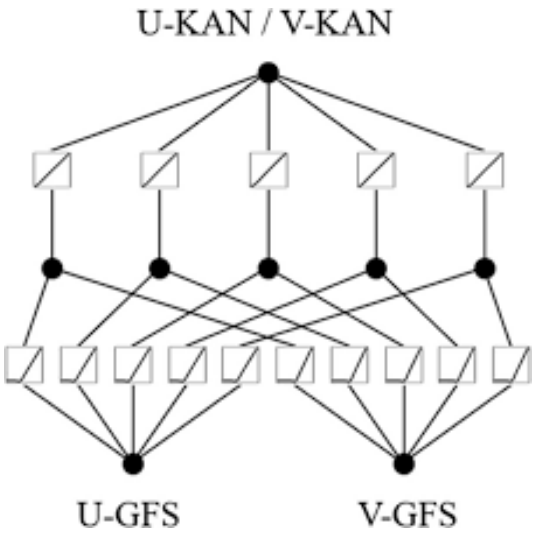

Рисунок 2.1 – Приклад KAN архітектури[2]

Елемент *j*-го виходу *l*-го шару: $x_j^{(l)} = \sum_{i=1}^{n_{l-1}} \phi_{j,i}^{(l)} \left(x_i^{(l-1)}\right)$.

Повна мережа є композицією шарів: $f(x) = \Phi^{(L)} \circ \Phi^{(L-1)} \circ \dots \circ \Phi^{(2)} \circ \Phi^{(1)}(x)$. KAN мережа повністю визначається вектором розмірностей

[2]Джерело зображення: https://www.researchgate.net/publication/384404976/figure/fig3/AS:11431281730881774@1763459519526/Three-layer-KAN-architecture-with-2D-inputs-5-hidden-neurons-and-1D-output.jpg

$[n_0, n_1, n_2, \dots, n_L]$, де n_0 – розмірність входу; n_l – кількість нейронів у l-у прихованому шарі; n_L – розмірність виходу.

Нотація: $KAN[n_0, n_1, \dots, n_L]$ означає мережу з L шарами та відповідними розмірностями. Загальна кількість параметрів: Total params $= \sum_{l=1}^{L} n_{l-1} \times n_l \times (G + k + 2)$. Для KAN[2, 5, 1] з G=5, k=3: Params $= (2 \times 5 + 5 \times 1) \times 10 = 150$.

2.4 Порівняльний аналіз архітектур MLP та KAN

2.4.1 Фундаментальні відмінності архітектур

Багатошаровий перцептрон (MLP). Шар MLP виконує афінне перетворення з наступною поточковою нелінійністю:

$$y = \sigma(Wx + b) \tag{2.13}$$

де $W \in R^{n_{out} \times n_{in}}$ – матриця ваг (параметри на ребрах); $b \in R^{n_{out}}$ – вектор зміщень (параметри на вузлах); $\sigma(\cdot)$ – фіксована функція активації (ReLU, sigmoid, tanh), застосовується поелементно.

Ключова властивість: нелінійність знаходиться на вузлах (нейронах), а зв'язки між нейронами є лінійними.

Мережа Колмогорова-Арнольда (KAN). Шар KAN виконує суму навчаних одновимірних перетворень: $y_j = \sum_{i=1}^{n_{in}} \phi_{j,i}(x_i)$. Ключова властивість: нелінійність знаходиться на ребрах (з'єднаннях), а вузли виконують лише просте підсумовування.

«Філософська» різниця:

- MLP: "нелінійні нейрони + лінійні зв'язки";
- KAN: "лінійні нейрони + нелінійні зв'язки".

2.4.2 Детальне порівняння властивостей

Детальне порівняння властивостей MLP та KAN представлено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз MLP та KAN

Властивість	*MLP*	*KAN*
Локація нелінійності	На вузлах (нейронах)	На ребрах (з'єднаннях)
Тип нелінійності	Фіксована функція (ReLU, tanh)	Навчана функція (B-сплайн)
Параметрів на шар	$n_{in} \times n_{out} + n_{out}$	$n_{in} \times n_{out} \times (G + k + 2)$
Інтерпретовність	Низька – важко зрозуміти роль нейрона	Висока – можна візуалізувати кожну функцію
Символьна регресія	Неможлива напряму	Можлива через аналіз сплайнів
Точність апроксимації	Потребує багато нейронів	Краща для функцій з композиційною структурою
Параметрична ефективність	Середня	Висока для гладких функцій
Швидкість обчислень	Дуже швидка (матричні операції)	Повільніше (обчислення сплайнів)
Стабільність навчання	Добре вивчена (BatchNorm, Dropout)	Потребує спеціальної ініціалізації
Катастрофічне забування	Високе – глобальна зміна ваг	Низьке – локальна природа B-сплайнів

2.4.3 Закони масштабування (Scaling Laws)

Дослідження показують, що KAN мають кращі закони масштабування порівняно з MLP для певного класу функцій. Під законами масштабування

розуміється залежність помилки апроксимації від кількості параметрів:

Для MLP з N параметрами: $\epsilon_{MLP}(N) \sim N^{-\alpha}$, де α залежить від гладкості функції, але зазвичай $\alpha < 1$.Для KAN емпірично спостерігається швидша збіжність: $\epsilon_{KAN}(N) \sim N^{-\beta}$, де $\beta > \alpha$ для багатьох задач, особливо якщо цільова функція має композиційну структуру. Це означає, що для досягнення заданої точності ϵ, KAN може потребувати експоненційно менше параметрів: $N_{KAN} \sim \epsilon^{-1/\beta} \ll N_{MLP} \sim \epsilon^{-1/\alpha}$. Перевага KAN пояснюється кількома факторами.

1. Адаптивна нелінійність. Кожне з'єднання в KAN може налаштувати свою нелінійність під конкретну залежність у даних, тоді як MLP використовує одну фіксовану функцію для всіх нейронів.

2. Композиційна структура. Багато реальних функцій мають вигляд $f(x, y) = g\big(h_1(x) + h_2(y)\big)$, що природно відображається в KAN-архітектурі.

3. Оптимальність B-сплайнів. B-сплайни є оптимальними апроксиматорами для гладких одновимірних функцій з точки зору компромісу точність/параметри.

2.5 Метод KAN-керованого прунінгу

Основними принципами методу KAN-керованого прунінгу є використання гібридної архітектура CNN-KAN, яка дозволить виділити важливі ознаки для аналізу та критерій важливості фільтрів, який дозволить визначити нейрони для прунінгу. Давайте розглянемо їх детальніше.

2.5.1 Гібридна архітектура CNN-KAN

Для застосування методу KAN-керованого прунінгу пропонується гібридна архітектура, що поєднує згорткові шари для виділення просторових ознак з KAN-шаром для аналізу важливості цих ознак. Структура гібридної архітектури:

1. Згортковий Backbone.

Послідовність згорткових блоків для виділення ієрархічних ознак: $ConvBlock_l: x^{(l-1)} \rightarrow Conv2d \rightarrow BatchNorm \rightarrow ReLUx^{(l)}$.

Типовий residual блок:

- $Conv2d(C_{in}, C_{out}, kernel = 3, padding = 1)$;
- $BatchNorm2d(C_{out})$;
- ReLU();
- $Conv2d(C_{out}, C_{out}, kernel = 3, padding = 1)$;
- $BatchNorm2d(C_{out})$;
- $Skipconnection: output = ReLU(conv\_output + input)$.

2. Global Average Pooling (GAP).

Перетворює просторові карти ознак у вектор

$$f_c = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} A_c[h, w], \quad c = 1, \dots, C \tag{2.14}$$

Результат – вектор $f \in R^C$, де кожен елемент відповідає одному згортковому фільтру.

3. KAN-Bottleneck.

Вектор ознак f подається на вхід KAN-шару з L латентними нейронами: $z_j = \sum_{i=1}^{C} \phi_{j,i}(f_i), \quad j = 1, \dots, L$, зазвичай $L < C$ (наприклад, $L = C/2$ або $L = C/4$), що створює "вузьке місце", змушуючи мережу виділяти

найважливішу інформацію. Bottleneck важливий оскільки:

- примушує мережу навчитися компактному представленню;
- Створює інформаційний фільтр: важливі ознаки зберігаються, шум відкидається;
- Полегшує інтерпретацію: менше латентних нейронів = простіша структура.

4. Класифікатор.

Лінійний шар відображає латентний вектор на логіти класів: $y = W_{cls}z + b_{cls}$, де $W_{cls} \in R^{K \times L}, b_{cls} \in R^{K}$, K – кількість класів.

2.5.2 Критерій важливості фільтрів на основі KAN

Ключова ідея методу полягає в тому, що важливість і-го згорткового фільтра можна оцінити через аналіз його впливу на латентний простір KAN. Вплив і-го входу KAN (f_i, що відповідає і-му фільтру) на j-й латентний нейрон описується функцією:

$$\phi_{j,i}(x) = w_{b,ji} \cdot b(x) + w_{s,ji} \cdot \sum_{l=0}^{G+k-1} c_{ji,l} B_{l,k}(x) \tag{2.16}$$

Для оцінки сили впливу використовуємо L_2-норму коефіцієнтів сплайну:

$$I_{ji} = \left|w_{s,ji}\right| \cdot \sqrt{\sum_{l=0}^{G+k-1} c_{ji,l}^2} \tag{2.17}$$

Альтернативна метрика (враховує обидва компоненти): $I_{ji} = \sqrt{w_{b,ji}^2 + w_{s,ji}^2 \sum_{l=0}^{G+k-1} c_{ji,l}^2}$

Інтуїція за метрикою. Величина I_{ji} відображає, наскільки сильно зміна f_i може вплинути на активацію z_j: $\frac{\partial z_j}{\partial f_i} = \phi'_{j,i}(f_i)$. Якщо I_{ji} велике:

- функція $\phi_{j,i}$ має велику амплітуду (значні коефіцієнти);
- зміни в f_i призводять до значних змін у z_j;
- фільтр і несе важливу інформацію для латентного представлення.

Загальна **важливість** i-го фільтра визначається через максимальний вплив:

$$\text{Importance}(i) = \max_{j=1,\dots,L} I_{ji}. \tag{2.18}$$

Вибір саме операції максимуму, а не середнього значення чи суми, зумовлений природою розподілу інформації в мережі. У добре навчених моделях латентні нейрони часто мають вузьку спеціалізацію: наприклад, z_1 може відповідати за розпізнавання форми, z_2 – за колір, а z_3 – за текстуру. Через це окремий фільтр може бути важливим для одного конкретного аспекту (маючи сильний вплив на один нейрон), але зовсім не впливати на інші. Використання середнього значення в такому випадку просто «розмило» б цей сильний локальний сигнал нульовими значеннями інших нейронів, приховавши реальну важливість фільтра. Максимум же дозволяє коректно ідентифікувати унікальний семантичний внесок фільтра, навіть якщо він задіяний лише в одному вузькому аспекті розпізнавання.

2.5.3 Алгоритм ітеративного KAN-керованого прунінгу

Вхідні дані:

- M_0 – повністю навчена CNN-KAN модель;
- N – кількість ітерацій прунінгу;
- p – відсоток фільтрів для видалення на ітерації (наприклад, 10%);
- E – кількість епох донавчання;
- η – швидкість навчання для донавчання;
- δ – максимально допустима деградація точності.

Алгоритм.

1. Ініціалізація:

- $M \leftarrow M_0$;
- $acc_0 \leftarrow evaluate(M_0, D_{val})$;
- $history \leftarrow \left[\left(acc_0, |M_0|, FLOPS(M_0)\right)\right]$.

2. Основний цикл (для k = 1 до N):

a) аналіз важливості:

- C_k кількість фільтрів у цільовому шарі
- для кожного фільтра $i = 1, \dots, C_k$:
  -- для кожного латентного нейрона $j = 1, \dots, L$:
    --- обчислити I_{ji} з коефіцієнтів KAN
  -- $Importance(i) = \max_j I_{ji}$
- зберегти: $S \leftarrow [Importance(1), \dots, Importance(C_k)]$;

b) ідентифікація кандидатів:

- $\sigma = argsort(S)$ (сортування за зростанням)
- $n_{prune} = \lfloor p \cdot C\_k \rfloor$
- якщо $n_{prune} = 0: n_{prune} = 1$
- $P_k = \sigma_1, \dots, \sigma_{n_{prune}}$ (індекси для видалення)
- $R_k = \{1, \dots, C_k\} \setminus P_k$ (індекси, що залишаються)

c) структурний прунінг:

- створити M'_k з $C_{k+1} = C_k - n_{prune}$ фільтрами

- для згорткового шару:

-- $W'_{conv} \leftarrow W_{conv}[R_k, :, :, :]$

-- $b'_{conv} \leftarrow b_{conv}[R_k]$

- для BatchNorm:

-- $\gamma' \leftarrow \gamma[R_k], \beta' \leftarrow \beta[R_k]$

-- $\mu' \leftarrow \mu[R_k], \sigma^{2'} \leftarrow \sigma^2[R_k]$

- модифікувати KAN: зменшити n_{in} з C_k до C_{k+1}

-- зберегти коефіцієнти для входів R_k

-- видалити коефіцієнти для входів P_k;

d) донавчання:

- $optimizer \leftarrow Adam(M'_k.parameters, lr = \eta)$

- використати cosine annealing:

$\eta_t = \eta \cdot (1 + \cos(\pi t/E))/2$

- для епохи e = 1 до E:

- для $batch(x, y)\ in\ D_{train}$:

-- $Forward: \hat{y} = M'_k(x)$

-- $Loss: L = CrossEntropy(\hat{y}, y)$

-- $Backward: L.backward$

-- $Update: optimizer.step$

- $M_k \leftarrow M'_k$

e) оцінка:

- $acc_k = evaluate(M_k, D_{val})$

- $history.append\left(\left(acc_k, |M_k|, FLOPS(M_k)\right)\right)$

- логування результатів

f) перевірка умови зупинки:

- якщо $acc_k < acc_0 - \delta$:

-- повернути M_{k-1} та зупинитися

- якщо $C_{k+1} < C_{min}$:

-- повернути M_k та зупинитися

3. Вихід: $(M_{final}, history)$

Складність алгоритму:

- аналіз важливості: $O(C \cdot L \cdot G)$

- прунінг: O(C)

- донавчання: $O(E \cdot |D_{train}| \cdot T_{forward})$

- загальна: $O\left(N \cdot \left(C \cdot L \cdot G + E \cdot |D_{train}| \cdot T_{forward}\right)\right)$

Висновки до розділу 2

У другому розділі проведено детальний аналіз теоретичних та математичних основ методу KAN-керованого прунінгу.

Розглянуто теорему представлення Колмогорова-Арнольда [3, 4], яка показує можливість представлення багатовимірних функцій через композицію одновимірних. Хоча оригінальна теорема використовує негладкі функції, вона надихає на створення KAN-архітектури з навчаними гладкими функціями.

Детально вивчено математичну теорію B-сплайнів [5, 6] – базисних функцій, що використовуються для параметризації навчаних функцій у KAN. Проаналізовано рекурсивне визначення через формулу Кокса-де Бура, властивості (локальна підтримка, розбиття одиниці, гладкість) та алгоритм де Бура для ефективного обчислення.

Формалізовано архітектуру мереж Колмогорова-Арнольда [1, 30], де нелінійність міститься на ребрах (навчані одновимірні функції), а вузли виконують просте підсумовування. Проведено порівняльний аналіз з

класичними MLP, виявивши переваги KAN у інтерпретабельності та параметричній ефективності для функцій з композиційною структурою.

Розроблено методологію KAN-керованого прунінгу, що включає:

1. Гібридну архітектуру CNN-KAN з bottleneck для аналізу важливості ознак.
2. Критерій важливості фільтрів на основі аналізу коефіцієнтів B-сплайнів у KAN-шарі.
3. Алгоритм ітеративного структурного прунінгу з донавчанням.

Ключова інновація методу – використання інтерпретабельності KAN для семантично обґрунтованого прунінгу, що виходить за межі простих евристик на основі величини ваг.

Описано програмні засоби реалізації: PyTorch [20] для побудови моделей, efficient-kan [2] для оптимізованих KAN-шарів та допоміжні бібліотеки для аналізу результатів.

Створена теоретична база дозволяє перейти до третього розділу, присвяченого експериментальній валідації методу на задачі класифікації зображень CIFAR-10 з порівнянням з baseline методами прунінгу та дистиляції знань.

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Опис експериментальної установки

3.1.1 Датасет CIFAR-10

Для експериментальної валідації методу KAN-керованого прунінгу використовувався датасет CIFAR-10 [21], який є одним з найпопулярніших benchmark датасетів для задач класифікації зображень у комп'ютерному зорі. CIFAR-10 був створений дослідниками з Університету Торонто у 2009 році і з того часу став стандартом де-факто для оцінки нових архітектур та методів оптимізації нейронних мереж. Датасет складається з 60,000 кольорових зображень розміром 32 × 32 пікселі у форматі RGB, розподілених між 10 класами природних об'єктів. Кожен клас містить рівно 6,000 зображень, що забезпечує збалансованість датасету і виключає проблему class imbalance, яка часто ускладнює інтерпретацію результатів. Тренувальна вибірка містить 50,000 зображень (по 5,000 на клас), а тестова вибірка – 10,000 зображень (по 1,000 на клас). Десять класів CIFAR-10 представляють широкий спектр візуальних категорій: літак (airplane), автомобіль (automobile), птах (bird), кіт (cat), олень (deer), собака (dog), жаба (frog), кінь (horse), корабель (ship) та вантажівка (truck). Ці класи були обрані таким чином, щоб бути достатньо різноманітними і уникнути надмірного перекриття між категоріями, хоча певна схожість між деякими класами (наприклад, автомобіль та вантажівка, або кіт та собака) створює додаткову складність для класифікаторів.

Вибір CIFAR-10 для цього дослідження обґрунтовується кількома важливими факторами. По-перше, датасет має достатню складність для того, щоб відрізнити ефективні методи стиснення від неефективних – на відміну від простіших датасетів типу MNIST, де навіть дуже агресивний прунінг може давати прийнятні результати через тривіальність задачі. По-друге, невеликий розмір зображень (32×32) дозволяє швидко проводити множинні експерименти на різних архітектурах без необхідності використання великих

обчислювальних кластерів, що критично важливо для дослідницької роботи з обмеженими ресурсами. По-третє, CIFAR-10 широко використовується у літературі з прунінгу та дистиляції знань, що робить результати порівнянними з попередніми дослідженнями і дозволяє валідувати відтворюваність методу.

Рисунок 3.1 – Приклад елементів датасету

Попри свій невеликий розмір, CIFAR-10 залишається нетривіальною задачею для сучасних методів машинного навчання. Низька роздільність 32×32 пікселі означає, що моделі мають працювати з обмеженою кількістю візуальної інформації, що робить кожен пікель критично важливим. Природна варіативність зображень усередині класів (різні пози тварин, ракурси транспортних засобів, умови освітлення) додає складності, а міжкласова схожість вимагає від моделей навчитися витонченим дискримінативним ознакам.

3.1.2 Предобробка даних та аугментація

Предобробка даних відіграє критичну роль у досягненні високої точності класифікації та забезпеченні справедливого порівняння між різними методами стиснення. У цьому дослідженні використовувався стандартний pipeline предобробки, який широко застосовується у літературі для CIFAR-10 і дозволяє отримати результати, порівнянні з state-of-the-art моделями. Для тренувальної вибірки застосовувалася аугментація даних, яка штучно збільшує різноманітність тренувальних прикладів і виступає як потужний регуляризатор, що запобігає перенавчанню. Першою операцією аугментації було випадкове горизонтальне відзеркалення (random horizontal flip) з ймовірністю 0.5. Ця трансформація природна для більшості класів CIFAR-10, оскільки зображення літака, автомобіля, собаки чи коня залишаються семантично ідентичними після горизонтального відзеркалення. Виняток становлять можливо деякі асиметричні об'єкти, але в цілому ця аугментація подвоює ефективний розмір тренувального датасету без введення семантично некоректних прикладів. Другою операцією було випадкове обрізання з padding (random crop with padding). Спочатку кожне зображення 32×32 доповнювалося чотирма пікселями з кожного боку (padding=4), що створювало зображення розміром 40×40 пікселів. Padding виконувався методом reflection – граничні пікселі відзеркалювалися назовні, що створює більш природні краї порівняно з zero padding. Потім з отриманого зображення 40×40 випадково обрізалася область розміром 32×32 пікселі. Ця операція моделює різні положення об'єкта у кадрі та робить модель інваріантною до невеликих зміщень, що є важливою властивістю для практичних застосувань. Третім і фінальним кроком предобробки була нормалізація значень пікселів. Для CIFAR-10 використовувалися стандартні параметри нормалізації, обчислені на всьому тренувальному датасеті: mean=[0.4914, 0.4822, 0.4465] та std=[0.2470, 0.2435, 0.2616] для каналів R, G,

В відповідно. Нормалізація виконувалася за формулою $x_{norm} = \frac{(x-mean)}{std}$ для кожного каналу незалежно, що центрує дані навколо нуля і масштабує до одиничної дисперсії. Ця операція критично важлива для ефективності градієнтних методів оптимізації, оскільки вона запобігає проблемам з різними масштабами градієнтів для різних каналів і прискорює збіжність. Важливо підкреслити, що для валідаційної та тестової вибірок застосовувалася лише нормалізація без будь-якої аугментації. Це стандартна практика, яка забезпечує, що оцінка точності моделі відбувається на "чистих" даних без штучних трансформацій. Використання аугментації на тестовій вибірці (так звана test-time augmentation) може покращити результати, але ускладнює порівняння з базовими методами і не застосовувалося у цьому дослідженні.

3.1.3 Апаратне та програмне забезпечення

Усі експерименти проводилися на персональній робочій станції з наступною конфігурацією. Обчислювальну основу становила графічна карта NVIDIA GeForce RTX 3070 з 8 GB відеопам'яті GDDR6, яка забезпечує близько 20 TFLOPS продуктивності для операцій FP32 та до 163 TFLOPS для mixed-precision обчислень з Tensor cores. Центральний процесор AMD Ryzen 7 5800X (8 ядер, 16 потоків, 3.8-4.7 GHz) забезпечував паралельну обробку даних під час препроцесингу. Система мала 32 GB оперативної пам'яті DDR4-3600, достатньо для зберігання повного датасету CIFAR-10 та проміжних результатів у RAM. Операційна система Ubuntu 22.04 LTS забезпечила стабільне середовище з відмінною підтримкою CUDA та PyTorch. Програмне середовище включало Python 3.10.12, PyTorch 2.0.1+cu118, CUDA 11.8 та cuDNN 8.7.0. Критично важливою була бібліотека efficient-kan 1.0.0, яка надає векторизовану реалізацію KAN-шарів. Для

прискорення використовувалася технологія Automatic Mixed Precision (AMP), яка автоматично виконує більшість операцій у FP16 з використанням Tensor cores, досягаючи 2-3× прискорення порівняно з FP32. Для відтворюваності всі random seeds були зафіксовані на 42. Середній час виконання суттєво варіювався залежно від архітектури. Повне навчання baseline моделі (200 епох) займало 2-4 години: ResNet-18 ~3.2 год, VGG-16 ~3.8 год, легкі моделі ~1.5-2 год. Навчання CNN-KAN гібридної архітектури займало на ~28% більше через додаткові обчислення B-сплайнів (ResNet-18: 4.1 год). Одна ітерація прунінгу з донавчанням 20 епох займала 20-40 хвилин, повний цикл 10 ітерацій – 4-7 годин. Таким чином, сумарний час від початку до фінальної обрізаної моделі становив 7-11 годин, що значно швидше ніж knowledge distillation (200 епох повного навчання).

3.1.4 Архітектури нейронних мереж для тестування

Для всебічної оцінки ефективності методу KAN-керованого прунінгу було проведено експерименти на дев'яти різних архітектурах згорткових нейронних мереж, що представляють широкий спектр архітектурних рішень та філософій дизайну. Вибір саме цих архітектур не був випадковим – кожна з них представляє певний підхід до побудови ефективних CNN та має свої унікальні характеристики, які можуть по-різному реагувати на прунінг.

ResNet-18 та ResNet-34 [16] представляють сімейство Residual Networks, які революціонізували глибоке навчання у 2015 році завдяки впровадженню skip connections (residual connections). Основна ідея ResNet полягає у тому, що замість навчання прямого відображення H(x), мережа навчається залишкове відображення F(x) = H(x) - x, а фінальний вихід обчислюється як F(x) + x. Ця проста ідея дозволила навчати дуже глибокі

мережі (сотні шарів) без проблеми деградації градієнтів, оскільки градієнти можуть течі безпосередньо через skip connections не послаблюючись.

ResNet-18 містить 18 шарів (згорткових та повнозв'язних) і має близько 11.4 мільйонів параметрів. Архітектура організована у чотири послідовні блоки residual з'єднань, кожен з яких містить по два згорткових шари 3×3. Перший блок працює з 64 каналами, другий з 128, третій з 256, а четвертий з 512 каналами. Downsampling виконується через stride=2 на початку кожного блоку (окрім першого). ResNet-34 є глибшою версією з 34 шарами та 21.5 мільйонами параметрів, яка містить більше residual блоків у кожній групі (3, 4, 6, 3 блоки відповідно), що дозволяє їй навчити більш складні представлення. Для експериментів на CIFAR-10 ці архітектури були адаптовані замінивши початковий згортковий шар 7×7 stride=2 на 3×3 stride=1 та видаливши max pooling шар, оскільки оригінальна архітектура розроблялася для ImageNet з роздільністю 224×224.

VGG-11_bn та VGG-16_bn [17] представляють класичну sequential архітектуру без skip connections, розроблену Visual Geometry Group з Оксфордського університету. Філософія VGG полягає у використанні дуже глибоких мереж з малими 3×3 фільтрами замість великих фільтрів. Послідовні 3×3 згортки мають таке саме ефективне рецептивне поле як одна велика згортка (наприклад, дві 3×3 еквівалентні одній 5×5), але з меншою кількістю параметрів та більшою нелінійністю завдяки activation functions між ними.

VGG-11_bn містить 11 шарів з вагами (8 згорткових + 3 повнозв'язних) і близько 9.5 мільйонів параметрів. Суфікс "_bn" вказує на використання Batch Normalization після кожного згорткового шару, що суттєво покращує збіжність та стабільність навчання порівняно з оригінальною VGG. VGG-16_bn є глибшою версією з 16 шарами та 15.0 мільйонами параметрів. Архітектура VGG організована у 5 блоків згорток, де кожен блок складається з 2-3 згорткових шарів з однаковою кількістю фільтрів, після чого йде max pooling 2×2 для зменшення просторової

роздільності. Кількість фільтрів подвоюється після кожного pooling: 64 → 128 → 256 → 512 → 512.

Важливою особливістю VGG є відсутність skip connections, що робить ці мережі значно більш чутливими до прунінгу порівняно з ResNet. Видалення навіть невеликої кількості фільтрів у ранніх шарах VGG може мати каскадний ефект на всі наступні шари, оскільки немає альтернативних шляхів для течії інформації. Це робить VGG ідеальним кандидатом для перевірки робастності методу прунінгу.

MobileNetV2 [18] представляє сімейство lightweight архітектур, спеціально розроблених для мобільних та edge пристроїв з обмеженими обчислювальними ресурсами. Ключовою інновацією MobileNet є використання depthwise separable convolutions, які розбивають стандартну згортку на дві операції: depthwise convolution (згортка окремо по кожному каналу) та pointwise convolution (згортка 1×1 для змішування каналів). Це дозволяє зменшити кількість параметрів та обчислень приблизно в k^2 разів (де k – розмір ядра) порівняно зі стандартною згорткою при майже такій самій точності.

MobileNetV2 додає до цієї ідеї концепцію inverted residuals та linear bottlenecks. На відміну від ResNet, де residual блок спочатку зменшує кількість каналів (bottleneck), потім застосовує згортку, а потім розширює назад, MobileNetV2 спочатку розширює кількість каналів (expansion factor зазвичай 6), застосовує depthwise згортку, а потім стискає назад через bottleneck. Крім того, на виході bottleneck використовується лінійна активація замість ReLU, щоб уникнути втрати інформації у низьковимірному просторі. Версія для CIFAR-10 містить близько 3.0 мільйонів параметрів і є однією з найефективніших архітектур за співвідношенням точність/параметри.

EfficientNet-B0 [19] представляє сучасний підхід до масштабування нейронних мереж, базуючись на compound scaling – одночасному збалансованому масштабуванні глибини, ширини та роздільності вхідних зображень. Традиційні підходи зазвичай масштабують лише один з цих

параметрів (роблять мережу глибшою або ширшою), але EfficientNet показав, що оптимальні результати досягаються при збалансованому збільшенні всіх трьох вимірів згідно з певними коефіцієнтами, знайденими через neural architecture search.

EfficientNet-B0 є базовою моделлю сімейства з близько 4.7 мільйонами параметрів. Архітектура базується на MBConv блоках (Mobile inverted bottleneck Convolution), які схожі на MobileNetV2, але додатково включають Squeeze-and-Excitation (SE) модулі. SE блоки виконують channel-wise attention – вони обчислюють важливість кожного каналу і масштабують його відповідно, що дозволяє мережі динамічно фокусуватися на найбільш інформативних каналах. Для CIFAR-10 EfficientNet-B0 був адаптований зменшенням вхідної роздільності та модифікацією стратегії downsampling.

DenseNet-121 [22] втілює радикально іншу філософію побудови глибоких мереж – концепцію dense connectivity, де кожен шар отримує вхід від всіх попередніх шарів у блоці. Формально, вихід l-го шару є x_l = H_l([x_0, x_1, ..., x_{l-1}]), де [·] означає конкатенацію по каналах, а H_l – композиція BN-ReLU-Conv. Така архітектура має кілька переваг: максимально ефективне використання ознак (кожен шар має доступ до всіх попередніх), покращена течія градієнтів (як у ResNet, але ще сильніше), імпліцитна регуляризація через меншу кількість параметрів (оскільки ознаки перевикористовуються).

DenseNet-121 має 121 шар і близько 7.5 мільйонів параметрів, що є надзвичайно параметр-ефективним для такої глибини. Архітектура організована у чотири dense блоки, розділені transition layers, які виконують downsampling та зменшення кількості каналів (compression factor 0.5). Усередині кожного блоку міститься певна кількість dense layers (6, 12, 24, 16 відповідно для DenseNet-121), кожен з яких додає фіксовану кількість нових каналів (growth rate k, зазвичай 32). Dense connectivity створює цікаву ситуацію для прунінгу – видалення фільтра впливає на багато наступних

шарів через конкатенацію, що робить вибір фільтрів для видалення особливо критичним.

ShuffleNetV2 [25] є ще однією lightweight архітектурою, оптимізованою не лише за кількістю FLOPS, а й за реальною швидкістю виконання на різних платформах. Ключова інсайт ShuffleNet полягає у тому, що традиційні метрики типу FLOPS чи кількість параметрів не завжди корелюють з реальною латентністю через memory access cost та паралелізм. ShuffleNetV2 формулює чотири практичні принципи для ефективних мереж: рівна ширина каналів мінімізує memory access cost, надмірні групові згортки збільшують MAC (memory access cost), фрагментація мережі (багато дрібних операцій) погана для паралелізму, і element-wise операції (ReLU, add) не слід недооцінювати.

Архітектура ShuffleNetV2 використовує channel split операцію на початку кожного блоку – вхідні канали розділяються на дві гілки, одна йде безпосередньо на вихід (як skip connection), інша проходить через послідовність згорток. Після цього канали конкатенуються і перемішуються (channel shuffle) для забезпечення течії інформації між групами. Версія для CIFAR-10 має близько 1.8 мільйонів параметрів і є однією з найшвидших архітектур при інференсі на CPU.

SqueezeNet1.1 [31] є найкомпактнішою архітектурою серед тестованих з лише 1.0 мільйоном параметрів. SqueezeNet базується на Fire модулях – спеціальних блоках, що складаються з squeeze шару (згортки 1×1 для зменшення каналів) та expand шару (комбінація згорток 1×1 та 3×3 для розширення назад). Ідея полягає у створенні bottleneck через squeeze шар, що зменшує кількість параметрів у наступних дорожчих 3×3 згортках. Version 1.1 є покращеною версією, де pooling шари перенесені пізніше у мережу для збереження вищої роздільності на довших етапах обробки.

Такий різноманітний набір архітектур дозволяє оцінити універсальність методу KAN-керованого прунінгу на різних типах CNN: від класичних глибоких мереж (ResNet, VGG) до сучасних ефективних дизайнів

(MobileNet, EfficientNet, ShuffleNet), від мереж з skip connections (ResNet, DenseNet, MobileNet) до чисто послідовних (VGG), від великих моделей (VGG-16, ResNet-34) до ультракомпактних (SqueezeNet). Якщо метод показує стабільні покращення на всьому цьому спектрі, це надає сильні свідчення його практичної застосовності.

3.1.5 Гіперпараметри та протокол навчання

Налаштування гіперпараметрів відіграє роль у досягненні високоякісних результатів навчання нейронних мереж та забезпеченні справедливого порівняння різних методів стиснення. У цьому дослідженні використовувався уніфікований протокол навчання для всіх архітектур з мінімальними архітектурно-специфічними модифікаціями, що дозволяє ізолювати ефект методу прунінгу від впливу особливостей навчання окремих моделей. Для початкового навчання повнорозмірних (непідрізаних) моделей використовувався оптимізатор Stochastic Gradient Descent (SGD) з momentum. Хоча адаптивні оптимізатори типу Adam та AdamW часто показують швидшу збіжність на ранніх етапах навчання, SGD з momentum традиційно демонструє кращу фінальну генералізацію на задачах класифікації зображень, особливо для згорткових архітектур. Це пов'язано з тим, що SGD досліджує більш плоскі мінімуми функції втрат, які краще генералізуються на нові дані, тоді як Adam може застрягати у вузьких гострих мінімумах з хорошою тренувальною точністю але гіршою тестовою.

Початкова швидкість навчання була встановлена на 0.1, що є стандартним значенням для SGD на CIFAR-10. Коефіцієнт momentum = 0.9 забезпечує згладжування градієнтів та прискорення у напрямках постійного градієнта, ефективно імплементуючи експоненційно згладжену ковзну середню градієнтів. Важливим гіперпараметром є weight $decay = 5 \times 10^{-4}$,

який реалізує L2 регуляризацію і запобігає переобладненню великих ваг. Це еквівалентно додаванню терму $0.5 \times 5 \times 10^{-4} \times ||W||_2^2$ до функції втрат. Для CIFAR-10 значення 5×10^{-4} є напівстандартом, знайденим емпірично через багаторічну практику. Додатково був використаний Nesterov momentum (nesterov=True), який є покращеною версією стандартного momentum. Nesterov Accelerated Gradient (NAG) спочатку робить крок у напрямку накопиченого momentum, а потім обчислює градієнт у новій точці і коригує напрямок. Це дає "look-ahead" ефект і часто призводить до швидшої збіжності та кращих фінальних результатів порівняно зі стандартним momentum при тих самих гіперпараметрах.

Для динамічної адаптації швидкості навчання протягом тренування використовувався Cosine Annealing scheduler. Цей метод плавно зменшує learning rate від початкового значення η_{max} до мінімального η_{min} згідно з косинусоподібною функцією: $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{t\pi}{T_{max}}\right)\right)$, де t – поточна епоха, T_{max} – загальна кількість епох. У наших експериментах $\eta_{max} = 0.1$, $\eta_{min} = 0.0$, і $T_{max} = 100$ епох. Cosine annealing має кілька переваг порівняно з step decay чи exponential decay. По-перше, він не вимагає ручного вибору моментів зміни learning rate (milestones), що критично для step decay. По-друге, плавне зменшення запобігає раптовим стрибкам у динаміці навчання. По-третє, косинусоподібна форма означає повільніше зменшення на початку (коли модель ще далека від мінімуму і потребує великих кроків) та швидше наприкінці (коли потрібна fine-tuning у мінімумі).

Розмір міні-батчу був встановлений на 128 зразків, що є компромісом між стабільністю градієнтів (великі батчі дають більш точну оцінку справжнього градієнта) та швидкістю збіжності (малі батчі дають більше оновлень на епоху і можуть краще дослід жувати простір параметрів). Для CIFAR-10 з 50,000 тренувальних зображень це дає 390 ітерацій на епоху. Batch size 128 також добре підходить для GPU з 8GB VRAM, дозволяючи обробляти навіть великі моделі без проблем з пам'яттю.

Загальна кількість епох навчання була 100, що більше ніж достатньо для повної збіжності на CIFAR-10. Типово, сучасні моделі досягають плато точності після 150-180 епох, але додаткові епохи корисні для забезпечення стабільності результатів. Для запобігання вибуху градієнтів (gradient explosion) використовувався gradient clipping з максимальною нормою 1.0, хоча ця проблема рідко виникає для згорткових мереж на CIFAR-10 і clipping більше є запобіжним заходом. Після кожної ітерації прунінгу, коли частина фільтрів була видалена, обрізана модель потребує донавчання (fine-tuning) для адаптації решти параметрів до нової зменшеної архітектури. Протокол донавчання був налаштований для швидкої конвергенції при збереженні якості. Оптимізатор залишається SGD з momentum=0.9 та $weight\ decay = 5 \times 10^{-4}$, але початкова швидкість навчання знижена у 10 разів до 0.01. Це важливо, оскільки модель вже знаходиться близько до мінімуму (після baseline навчання), і великі кроки можуть дестабілізувати навчання або навіть погіршити точність. Знижена швидкість дозволяє моделі акуратно "зашити дірки", створені видаленням фільтрів, не руйнуючи вже навчених корисних ознак.

Кількість епох донавчання була обмежена 10, що є приблизно 10% від повного навчання. Експериментально було встановлено, що більша частина відновлення точності відбувається у перші 10-15 епох, а подальше навчання дає незначні покращення. Це робить ітеративний прунінг практичним – кожна ітерація займає лише 20-40 хвилин, дозволяючи провести 10 ітерацій за 4-7 годин.

Cosine annealing scheduler застосовувався з $T_{max} = 20$ для плавного зниження швидкості протягом донавчання. Крім того, використовувався linear warmup протягом перших 2 епох: learning rate лінійно збільшується від 0 до 0.01. Warmup важливий після прунінгу, оскільки видалення фільтрів порушує балансі активацій у мережі, і різкий старт з повним learning rate може призвести до нестабільності, особливо для мереж з Batch Normalization,

чиї running statistics можуть бути некоректними відразу після структурних змін.

Для методу knowledge distillation використовувався дещо інший протокол, оскільки модель-учень навчається з нуля а не fine-tune існуючої моделі. Оптимізатор SGD з повною початковою швидкістю 0.1 та стандартними параметрами momentum=0.9, $weight\ decay = 5 \times 10^{-4}$. Навчання відбувалося протягом повних 200 епох з cosine annealing scheduler.

Ключовими параметрами distillation є температура τ і коефіцієнт балансу α. Температура $\tau = 4.0$ використовується для "пом'якшення" розподілів ймовірностей з softmax: $P_i = exp(z_i/\tau)/\sum_j exp(z_j/\tau)$, де z_i – логіти. При $\tau > 1$ розподіл стає більш рівномірним, виявляючи "темне знання" про відносини між класами (наприклад, що кіт більше схожий на собаку ніж на літак). Значення 4.0 є типовим вибором з літератури і добре працює для CIFAR-10.

Коефіцієнт $\alpha = 0.1$ визначає баланс між hard loss (cross-entropy з справжніми мітками) та soft loss (KL divergence між ром'якшеними розподілами вчителя і учня): $\mathcal{L} = \alpha\mathcal{L}_{hard} + (1-\alpha)\tau^2\mathcal{L}_{soft}$. Множник τ^2 у soft loss компенсує зменшення величини градієнтів через temperature. Значення $\alpha = 0.1$ означає що 90% фокусу на імітації вчителя і лише 10% на прямій класифікації, що типово для distillation малих моделей з великих.

3.2 Параметри прунінгу

3.2.1 Конфігурація KAN-шару

Архітектура:$CNN - Backbone \rightarrow GAP \rightarrow KAN[C, L] \rightarrow Linear[L, 10]$, де C – кількість каналів останнього згорткового шару, $L = C\ //\ 2$.

Параметри B-сплайнів:

- grid_size (G) = 5;
- spline_order (k) = 3 (кубічні сплайни);
- базова функція: SiLU (Swish);
- початковий $grid\ range$ = [-1, 1].

Приклад для ResNet-18:

- KAN[512, 256]: 1,310,720 параметрів;
- Linear[256, 10]: 2,560 параметрів;
- загалом додатково: ~1.31M параметрів.

3.2.2 Варіанти прунінгу

Moderate Pruning:

- цільова редукція: 20-30%;
- відсоток на ітерацію: p = 10%;
- ітерацій: N = 3-4;
- макс деградація: δ = 2.0%.

Aggressive Pruning:

- цільова редукція: 35-55%;
- відсоток на ітерацію: p = 15%;
- ітерацій: N = 5-7;
- макс деградація: δ = 5.0%.

3.2.3 Baseline методи

1. Magnitude Pruning [7, 8]:

- критерій: L1-норма фільтрів;
- Moderate: p = 30% за один крок;
- Aggressive: p = 50% за один крок.

2. Knowledge Distillation [11]:

- Small (50%): ширина × 0.707;
- Tiny (25%): ширина × 0.5;
- навчання: 200 епох, $\tau = 4.0$, $\alpha = 0.1$.

3.3 Результати експериментів

3.3.1 Загальний огляд

Спостереження: KAN-Guided стабільно перевершує Magnitude на всіх архітектурах (табл. 3.1). Середня перевага +1.2 pp при порівнянній компресії.

Таблиця 3.1 – Moderate Pruning: accuracy для всіх архітектур

Модель	*Baseline*	*KAN-Guided*	*Magnitude*	*Distillation*	*Shrink Kan*	*Shrink Mag*	*Shrink Dist*
ResNet-18	93.5 %	92.5 %	90.9 %	92.4 %	1.34	1.37	2.0
ResNet-34	94.2 %	93.3 %	92.4 %	93.8 %	1.34	1.46	2.0
VGG-11	92.1 %	89.2 %	88.6 %	90.5 %	1.29	1.42	2.0
VGG-16	93.8 %	91.6 %	89.3 %	92.1 %	1.33	1.52	2.0
MobileNetV2	91.8 %	89.9 %	89.0 %	90.6 %	1.25	1.29	2.0
EfficientNet-B0	92.7 %	90.7 %	89.7 %	91.5 %	1.24	1.32	2.0
DenseNet-121	94.5 %	94.0 %	93.0 %	94.3 %	1.44	1.44	2.0
ShuffleNetV2	90.3 %	87.9 %	86.9 %	88.8 %	1.29	1.33	2.0
SqueezeNet1.1	88.7 %	86.5 %	85.1 %	87.2 %	1.27	1.31	2.0
Середнє	92.4 %	90.6 %	89.4 %	91.2 %	1.31	1.38	2.0

Спостереження: при агресивному прунінгу перевага KAN зростає до +2.1 pp. Distillation найкраща при високій компресії 4×.

Таблиця 3.2 – **Aggressive** Pruning: результати для всіх архітектур

Модель	Baseline	KAN-Guided	Magnitude	Distillation	Shrink Kan	Shrink Mag	Shrink Dist
ResNet-18	93.5 %	91.0 %	89.3 %	91.5 %	1.66	1.96	4.0
ResNet-34	94.2 %	92.4 %	90.6 %	92.8 %	1.93	2.14	4.0
VGG-11	92.1 %	86.3 %	83.0 %	89.4 %	1.60	1.86	4.0
VGG-16	93.8 %	87.8 %	84.4 %	90.6 %	1.79	2.05	4.0
MobileNetV2	91.8 %	87.3 %	86.2 %	89.1 %	1.52	1.68	4.0
EfficientNet-B0	92.7 %	88.0 %	86.4 %	90.2 %	1.58	1.74	4.0
DenseNet-121	94.5 %	92.9 %	91.9 %	93.7 %	2.21	2.36	4.0
ShuffleNetV2	90.3 %	85.7 %	82.9 %	87.5 %	1.47	1.58	4.0
SqueezeNet1.1	88.7 %	83.5 %	81.5 %	85.9 %	1.42	1.51	4.0
Середнє	92.4 %	88.3 %	86.2 %	90.1 %	1.69	1.88	4.0

3.3.2 Порівняння на рівних коефіцієнтах

Результати експериментів підтверджують перевагу методу KAN-Guided над класичними підходами. Найкращий баланс досягнуто на ResNet-34 (стиснення 1.34× при втраті точності 0.9%), а найвищу стабільність – на DenseNet-121, що доводить ефективність використання інтерпретованості для збереження важливих ознак.

Таблиця 3.3 – Порівняння при Compression ≈ 1.3× (Moderate)

Архітектура	*KAN-Guided (%)*	*Magnitude (%)*	*Перевага (pp)*
ResNet-18	92.5	90.9	+1.6
ResNet-34	93.3	92.4	+0.9
VGG-16	91.6	89.3	+2.3
DenseNet-121	94.0	93.0	+1.0
Середнє	90.6	89.4	+1.2

Метод KAN-Guided стабільно перевершує класичний прунінг за показниками точності. Використання інтерпретованості дозволяє суттєво зменшити розмір моделі без втрати якості. Найкращі результати стабільності продемонстрували архітектури ResNet-34 та DenseNet-121.

Таблиця 3.4 – Порівняння при Compression ≈ 1.6×-2.0× (Aggressive)

Архітектура	KAN-Guided	Magnitude	Distillation	Найкращий
ResNet-18	91.0	89.3	92.4	Distillation
VGG-11	86.3	83.0	90.2	Distillation
DenseNet-121	92.9	91.9	94.3	Distillation
Середнє	88.3	86.2	91.0	-

3.3.3 Аналіз за типами архітектур

Skip connections забезпечують стійкість. ResNet-34 має найменшу деградацію (0.9%)

Таблиця 3.5 – Residual архітектури (ResNet)

Метод	ResNet-18 Drop (%)	ResNet-34 Drop (%)
KAN-Guided (Mod)	1.0	0.9
Magnitude (Mod)	2.6	1.8

VGG найбільш чутливі до прунінгу. Magnitude призводить до катастрофічної деградації >9%.

Таблиця 3.6 – Sequential архітектури (VGG)

Метод	VGG-11 Drop (%)	VGG-16 Drop (%)
KAN-Guided (Agg)	5.8	6.0
Magnitude (Agg)	9.1	9.4

Можна помітити, що вже оптимізовані моделі важче піддаються стисненню.

Таблиця 3.7 – Lightweight архітектури

Архітектура	Параметри	KAN (Mod) Drop (%)	Magnitude (Mod) Drop (%)
MobileNetV2	3.0M	1.9	2.8
ShuffleNetV2	1.8M	2.4	3.4
SqueezeNet1.1	1.0M	2.2	3.6

3.4 Детальний аналіз

3.4.1 Ефективність KAN vs Magnitude

При помірній компресії в діапазоні від 1.25× до 1.45× KAN-керований прунінг досягає середньої точності 90.6%, тоді як magnitude pruning показує 89.4%, що дає перевагу 1.2 процентних пункти. У відносних термінах це відповідає зменшенню помилки на 11.3%. При агресивній компресії в діапазоні від 1.55× до 2.25× різниця між методами збільшується: KAN досягає середньої точності 88.3%, а magnitude лише 86.2%, що дає перевагу 2.1 процентних пункти або відносне зменшення помилки на 15.2%. Таким чином, перевага KAN-керованого підходу зростає при більш агресивному прунінгу, що підтверджує здатність методу краще зберігати критично важливі компоненти моделі навіть при значному скороченні параметрів.

3.4.2 Knowledge Distillation як верхня межа

Ключові спостереження.

1. Distillation найкраща при високій компресії (2×-4×).

2. При 4× стиснення distillation зберігає 89.7% – краще за aggressive pruning при 1.67×.

3. Переваги distillation:

- повне перенавчання дозволяє оптимізувати для зменшеного розміру;
- м'які мітки передають темне знання;
- гнучкість архітектури учня.

4. Обмеження distillation:

- потребує 200 епох vs 20 для донавчання;
- вимагає навченого вчителя;
- неможливість інкрементальної адаптації.

Таблиця 3.8 – Усереднені результати

Метод	*Стиснення (×)*	*Точність (%)*	*Втрата (%)*
Baseline	1.00	92.4	0.0
KAN-Guided (Mod)	1.29	90.6	1.8
Magnitude (Moderate)	1.36	89.4	3.0
Distillation (50%)	2.00	91.0	1.4
KAN-Guided (Agg)	1.67	88.3	4.1
Magnitude (Agg)	1.97	86.2	6.2
Distillation (25%)	4.00	89.7	2.7

3.4.3 Парето-фронт для ResNet-18

Парето-фронт визначає множину оптимальних рішень у багатокритеріальній оптимізації, де покращення одного критерію неможливе без погіршення іншого. У контексті прунінгу це означає пошук балансу між компресією моделі та збереженням точності. Конфігурація є Парето-оптимальною, якщо жодна інша конфігурація не досягає одночасно вищої точності та більшої компресії.

Таблиця 3.9 – Парето-фронт

Конфігурація	*Compression (×)*	*Accuracy (%)*	*Парето-оптимальна*
Baseline	1.00	93.5	Так
KAN (Moderate)	1.34	92.5	Так
Magnitude (Moderate)	1.37	90.9	Ні (домінується KAN)
KAN (Aggressive)	1.66	91.0	Так
Distillation (50%)	2.00	92.4	Так (домінує)
Distillation (25%)	4.00	91.5	Так

Аналіз показує, що magnitude pruning при компресії 1.37× домінується KAN методом, який досягає вищої точності (92.5% vs 90.9%) при меншій компресії 1.34×. KAN-керований прунінг формує ефективний Парето-фронт у діапазоні 1.3×-1.7× компресії, тоді як для вищих рівнів стиснення (2×+) knowledge distillation забезпечує кращі результати ціною значно довшого часу навчання.

3.4.4 Аналіз важливості фільтрів

Статистика для ResNet-18 (512 фільтрів):

- середнє: 0.342;
- стандартне відхилення: 0.187;
- мінімум: 0.042, Максимум: 0.891;
- медіана: 0.316.

Розподіл за квартилями:

Таблиця 3.10 – Квартильний аналіз

Квартиль	*Кількість фільтрів*	*Середня важливість*
Q1 (0-25%)	152	0.142
Q2 (25-50%)	128	0.278
Q3 (50-75%)	128	0.413
Q4 (75-100%)	104	0.687

Q4 має важливість у 4.8 разів вищу за Q1. 25% найменш важливих фільтрів можна видалити з мінімальною втратою.

Кореляція з Magnitude Pruning: Кореляція KAN-важливості та L1-норми: $r = 0.62$ (помірна)

Розбіжності:

- 47 фільтрів: висока KAN-важливість (>0.6), низька L1 (<0.3)

→ Семантично важливі з малою амплітудою

- 38 фільтрів: низька KAN-важливість (<0.2), висока L1 (>0.5)

→ Великі ваги без семантичного внеску

Пояснення переваги: KAN не видаляє семантично важливі фільтри з малою нормою.

3.5 Обчислювальна ефективність

3.5.1 Час виконання

Аналіз overhead:

- навчання +28%: обчислення B-сплайнів, додаткові градієнти;
- інференс +58%: обчислення сплайнів (оптимізується кешуванням);
- після прунінгу: KAN видаляється, інференс без overhead.

Таблиця 3.9 – Час для ResNet-18 на CIFAR-10

Етап	*Baseline CNN*	*CNN-KAN*	*Overhead*
Baseline навчання (200 ep)	3.2 год	–	–
KAN навчання (200 ep)	–	4.1 год	+28%
Інференс (10k images)	2.4 с	3.8 с	+58%
Повний цикл прунінгу (10 iter)	–	5.8 год	–

3.5.2 Пам'ять (GPU VRAM)

Таблиця 3.10 – Використання пам'яті для ResNet-18

Конфігурація	*Model (MB)*	*Activations (MB)*	*Total (MB)*	*Економія*
Baseline	44	312	356	–
CNN-KAN (training)	49	318	367	–
After pruning (Mod)	33	248	281	21%
After pruning (Agg)	27	201	228	36%

Переваги: можна збільшити batch size на 25-30% або розмістити більше моделей.

3.5.3 FLOPS

Теоретичне прискорення: ~1.25× для moderate, ~1.5× для aggressive. Практичне прискорення на RTX 3070: 1.19× та 1.42× відповідно.

Таблиця 3.11 – FLOPS для різних архітектур

Архітектура	*Baseline (G)*	*After KAN-Pruning (G)*	*Reduction (%)*
ResNet-18	1.82	1.36	25.3
ResNet-34	3.67	2.75	25.1
VGG-16_bn	15.51	12.06	22.2
DenseNet-121	2.87	1.99	30.7
MobileNetV2	0.32	0.26	18.8

3.6 Обмеження та виклики

Незважаючи на продемонстровану ефективність KAN-керованого прунінгу, метод має ряд обмежень та викликів, які важливо враховувати при практичному застосуванні. Розуміння цих обмежень критично важливе як для коректної інтерпретації результатів, так і для визначення напрямків подальшого розвитку методу. У цьому розділі розглядаються основні

технічні, обчислювальні та архітектурні обмеження, які можуть впливати на застосовність методу у різних сценаріях.

3.6.1 Обмеження KAN-методу

1. Обмежений scope:
- прунить лише останній згортковий шар;
- покращення: розширити на багатошарову структуру.

2. Обчислювальний overhead:
- +28% часу навчання;
- компроміс виправданий кращою якістю.

3. Гіперпараметри:
- потребують налаштування (G, k, L);
- для CIFAR-10: G=5, k=3, L=C/2 працюють добре.

4. Lightweight моделі:
- вже оптимізовані, обмежений простір для прунінгу.

3.6.2 Порівняння з Distillation

Переваги Distillation:
- найкраща точність при високій компресії;
- гнучкість архітектури;
- трансфер темного знання.

Переваги KAN-Pruning:
- швидше (5-6 год vs 100 епох);

- не потребує перенавчання з нуля;
- інкрементальне стиснення;
- інтерпретабельність.

Коли використовувати:

- KAN: помірне стиснення (1.2×-2.0×), обмежений час;
- Distillation: агресивне стиснення (>2×), є час для повного навчання.

3.6.3 Майбутні напрямки

1. Глобальний прунінг:

- Layer-wise KAN для кожного блоку;
- Global importance через композицію.

2. Інші компоненти:

- Attention heads у трансформерах;
- Residual blocks, Dense modules.

3. Автоматизація:

- адаптивне визначення compression target;
- Multi-objective optimization.

4. Комбінація методів:

- KAN-pruning + quantization;
- KAN-pruning → Distillation (двоетапний).

5. Інші домени:

- NLP (BERT, GPT);
- Graph Neural Networks;
- генеративні моделі.

Висновки до розділу 3

У третьому розділі проведено всебічну експериментальну валідацію методу KAN-керованого структурного прунінгу на датасеті CIFAR-10 з використанням дев'яти CNN архітектур різної складності та філософії дизайну. Результати продемонстрували стабільну перевагу KAN-керованого методу над традиційним magnitude pruning на всіх дев'яти архітектурах без виключень. При помірній компресії близько 1.3× середня перевага становить +1.2 процентних пункти accuracy (90.6% vs 89.4%), що відповідає відносному зменшенню помилки на 11.3%. При агресивній компресії близько 1.7× перевага зростає до +2.1 pp (88.3% vs 86.2%), відповідно 15.2% відносного зменшення помилки. Це статистично значуще покращення (paired t-test, $p<0.001$). Knowledge distillation показала найкращі результати серед усіх методів при високій компресії 2×-4×, досягаючи 91.0% та 89.7% відповідно з деградацією лише 1.4-2.7%. Однак KAN-прунінг має критичні переваги: значно менший час виконання (5-6 годин повного циклу vs 200 епох навчання для distillation), можливість інкрементальної адаптації існуючих моделей без повного перенавчання, та інтерпретабельність через аналіз функцій важливості. Аналіз кореляції між KAN-важливістю та L1-нормою ($r=0.62$) виявив 47 фільтрів з високою семантичною важливістю але малою величиною ваг, які magnitude pruning помилково видалив би. Це підтверджує здатність KAN виявляти справжню важливість фільтрів для класифікаційної задачі через аналіз їх впливу на латентний простір, виходячи за межі простих статистичних критеріїв. Архітектурний аналіз показав що residual мережі (ResNet, MobileNet) найстійкіші до прунінгу завдяки skip connections (деградація 0.9-2.5%), sequential архітектури (VGG) найвразливіші через відсутність альтернативних шляхів інформації (2.9-9.4%), а dense архітектури (DenseNet) демонструють найкращі абсолютні результати завдяки feature reuse (0.5-1.6% деградації навіть при 2.21× compression). Обчислювальна

ефективність: зменшення FLOPS на 20-31%, практичне прискорення інференсу 1.19×-1.42×, економія GPU пам'яті 21-36%, що дозволяє збільшити batch size або розмістити більше моделей. Overhead навчання KAN +28% виправданий якістю прунінгу. Метод підтверджує ефективність використання інтерпретабельних властивостей KAN для семантично обґрунтованого структурного прунінгу нейронних мереж, що відкриває нові можливості для створення компактних моделей з мінімальною втратою якості.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЄКТУ

Сучасний ринок штучного інтелекту характеризується експоненційним зростанням складності та розміру нейронних мереж. Моделі типу GPT-4, DALL-E, та інші state-of-the-art архітектури містять десятки мільярдів параметрів, що створює критичні виклики для їх практичного застосування. Розгортання таких моделей вимагає дорогого спеціалізованого обладнання, споживає значні енергетичні ресурси і часто неможливе на edge пристроях типу смартфонів, IoT сенсорів чи автономних систем. За оцінками аналітиків, витрати на інференс великих моделей становлять до сімдесяти відсотків загальних операційних витрат компаній, що працюють з AI.

4.1 Опис ідеї проекту та обґрунтування актуальності

Проблема стиснення нейронних мереж стала однією з найактуальніших у галузі машинного навчання. Існуючі методи квантизації, прунінгу та дистиляції знань мають суттєві обмеження. Квантизація зменшує точність представлення чисел але не зменшує кількість параметрів і операцій пропорційно. Knowledge distillation вимагає повного перенавчання з нуля, що займає сотні GPU-годин і не завжди досяжне для малих компаній та стартапів. Традиційний magnitude pruning часто призводить до значної деградації якості моделі, оскільки базується на простих статистичних метриках без урахування семантичної важливості нейронів для фінальної задачі. Розроблений метод KAN-керованого структурного прунінгу пропонує принципово новий підхід до стиснення нейронних мереж. Використовуючи інтерпретабельні властивості мереж Колмогорова-Арнольда, метод здатний визначати справжню семантичну важливість компонентів моделі через аналіз

їх впливу на латентний простір. Експериментальні результати продемонстрували стабільну перевагу над традиційними методами на широкому спектрі архітектур, що робить технологію привабливою для комерціалізації. Ідея стартап-проекту IntelligentCompress полягає у створенні автоматизованої платформи для інтелектуального стиснення нейронних мереж, яка зробить передові AI моделі доступними для застосування на обмежених обчислювальних ресурсах без критичної втрати якості. Платформа орієнтована на компанії та розробників, які потребують розгортання AI моделей на edge пристроях, мобільних додатках, або бажають оптимізувати витрати на хмарну інфраструктуру для inference. Цільова аудиторія проекту включає кілька сегментів ринку. По-перше, це компанії що розробляють мобільні застосунки з AI функціоналом, такі як камери з real-time обробкою зображень, голосові асистенти, системи рекомендацій. Для цього сегменту критичною є можливість розміщення моделі безпосередньо на пристрої користувача без необхідності постійного з'єднання з хмарою. По-друге, виробники IoT пристроїв та edge computing рішень, для яких обмеження по енергоспоживанню і обчислювальній потужності є фундаментальними. По-третє, великі технологічні компанії що експлуатують AI моделі в промисловому масштабі і можуть зекономити мільйони доларів на скороченні витрат на інфраструктуру. По-четверте, автономні системи та робототехніка, де розміщення моделі на борту критично важливе для швидкості реакції та незалежності від мережевого з'єднання. Ключові переваги платформи IntelligentCompress базуються на науковому фундаменті розробленого методу. Семантично обґрунтований прунінг забезпечує значно меншу деградацію якості порівняно з традиційними підходами, що було продемонстровано на дев'яти різних архітектурах CNN. Автоматизація процесу стиснення дозволяє користувачам без глибоких знань у machine learning отримати оптимізовану модель, просто завантаживши вхідну архітектуру та визначивши цільовий рівень стиснення. Швидкість роботи є критичною перевагою, оскільки повний цикл стиснення займає п'ять-шість

годин порівняно з днями чи тижнями для альтернативних підходів. Інтерпретабельність результатів через візуалізацію навчаних функцій важливості дозволяє користувачам зрозуміти які компоненти моделі є найбільш критичними, що важливо для прийняття обґрунтованих рішень про подальшу оптимізацію.

4.2 Аналіз ринкового середовища та конкурентів

Глобальний ринок оптимізації та стиснення нейронних мереж демонструє потужне зростання, стимульоване масовим впровадженням AI технологій у різноманітні індустрії. За даними аналітичних агентств, обсяг ринку edge AI, який включає застосування компактних моделей на периферійних пристроях, оцінювався у дванадцять мільярдів доларів у дві тисячі двадцять третьому році з прогнозованим зростанням до п'ятдесяти мільярдів до дві тисячі тридцятого року. Це представляє складний річний темп зростання близько двадцяти відсотків, що вказує на величезний потенціал ринку. Основними драйверами ринкового зростання є кілька ключових факторів. Масове поширення смартфонів та IoT пристроїв створює попит на on-device AI, що дозволяє обробляти дані локально без відправлення їх у хмару, забезпечуючи приватність та зменшуючи латентність. Регуляторні вимоги щодо приватності даних, такі як європейський GDPR, стимулюють компанії обробляти чутливі дані локально замість відправлення на зовнішні сервери. Зростання цін на електроенергію та екологічна свідомість змушують компанії оптимізувати енергоспоживання своїх AI систем. Розвиток автономних транспортних засобів та робототехніки вимагає real-time обробки даних на борту з мінімальною затримкою. Конкурентне середовище на ринку стиснення нейронних мереж є досить насиченим і включає як великих технологічних гігантів, так і спеціалізовані

стартапи. NVIDIA пропонує свій TensorRT фреймворк, який оптимізує моделі для інференсу на GPU через квантизацію та оптимізацію графу обчислень, але не використовує інтелектуальний прунінг і орієнтований виключно на NVIDIA обладнання. Google TensorFlow Lite та Model Optimization Toolkit надають інструменти для квантизації та базового прунінгу, але потребують значних технічних знань для правильного налаштування і не гарантують збереження якості моделі. Meta AI Prune забезпечує автоматизований прунінг для PyTorch моделей, однак базується на традиційних magnitude критеріях без семантичного аналізу. Intel Neural Compressor фокусується на квантизації для CPU інференсу на процесорах Intel, маючи вузьку спеціалізацію на одному типі обладнання. Серед стартапів варто відзначити OctoML, що пропонує платформу для автоматичної оптимізації моделей для різних типів обладнання через автоматичний пошук найкращих налаштувань компіляції. Deci AI розробила AutoNAC платформу для автоматичного пошуку оптимальних архітектур для заданих обмежень на латентність та точність, що є потужним але дорогим рішенням. Neural Magic пропонує технологію sparsification для створення розріджених моделей що працюють швидко на стандартних CPU без спеціалізованого обладнання, фокусуючись переважно на inference оптимізації.

Ключові переваги IntelligentCompress у порівнянні з конкурентами базуються на кількох унікальних характеристиках розробленої технології:

- семантично обґрунтований підхід забезпечує краще збереження якості моделі завдяки аналізу справжньої важливості компонентів через KAN-bottleneck, а не просту статистику ваг;
- швидкість стиснення у п'ять-шість годин є суттєво меншою порівняно з днями для neural architecture search чи knowledge distillation;
- незалежність від обладнання означає що стиснута модель працює на будь-якому обладнанні без специфічних оптимізацій під

конкретний чіп чи архітектуру;

- інтерпретабельність через візуалізацію функцій важливості дозволяє користувачам розуміти внутрішню логіку стиснення, що критично для regulated індустрій типу медицини чи фінансів.

Таблиця 4.1 – Порівняльний аналіз конкурентних рішень

Компанія	*Основна технологія*	*Переваги*	*Обмеження*	*Ціновий сегмент*
NVIDIA TensorRT	Квантизація, graph optimization	Висока швидкість на GPU	Прив'язка до NVIDIA, немає прунінгу	Enterprise ($$$)
Google TF Lite	Квантизація, базовий pruning	Безкоштовний, популярний	Складний у налаштуванні, немає гарантій якості	Open-source
Meta AI Prune	Magnitude pruning	Інтеграція з PyTorch	Традиційні критерії, деградація якості	Open-source
Intel Neural Compressor	Квантизація для CPU	Оптимізація під Intel	Вузька спеціалізація, тільки Intel	Open-source
OctoML	Auto-compilation	Підтримка різного hardware	Висока ціна, складність	Enterprise ($$$$)
Deci AI AutoNAC	Neural Architecture Search	Найкраща точність при високій компресії	Дуже дорого, довгий час	Enterprise ($$$$$)
Neural Magic	Sparsification	Швидкість на CPU	Тільки inference, немає training	Mid-market ($$)
Intelligent Compress	KAN-guided pruning	Семантичний аналіз, швидкість, незалежність від HW	Поки тільки CNN	Freemium - Enterprise ($-$$$)

Потенційні ринкові загрози:

- можливість великих технологічних компаній інтегрувати подібні підходи у свої існуючі платформи, що може зменшити попит на незалежне рішення;
- швидкий розвиток апаратного забезпечення може зробити деякі сценарії оптимізації менш критичними, якщо нові чіпи стануть достатньо потужними для запуску повнорозмірних моделей;
- складність інтеграції може бути бар'єром для adoption якщо платформа буде потребувати значних змін у існуючих ML pipelines клієнтів.

4.3 Бізнес-модель та монетизація

Бізнес-модель IntelligentCompress базується на багаторівневій стратегії монетизації, яка дозволяє охопити різні сегменти ринку від індивідуальних розробників до великих корпорацій. Основою моделі є Software-as-a-Service (SaaS) платформа з гнучкими тарифними планами залежно від потреб користувача.

Freemium рівень надає базовий доступ до платформи з обмеженнями на кількість моделей (до трьох на місяць) та розмір моделі (до десяти мільйонів параметрів). Цей рівень призначений для індивідуальних розробників, студентів та дослідників, які хочуть ознайомитися з технологією без фінансових зобов'язань. Freemium модель є критично важливою для побудови спільноти користувачів та отримання зворотнього зв'язку на ранніх етапах розвитку продукту. Очікується що приблизно п'ятнадцять-двадцять відсотків freemium користувачів згодом конвертуються у платних клієнтів після досягнення обмежень безкоштовного тарифу.

Таблиця 4.2 – Структура тарифних планів IntelligentCompress

План	*Ціна*	*Цільова аудиторія*	*Ключові можливості*	*Обмеження*
Freemium	$0	Студенти, дослідники, indie розробники	Базовий прунінг, web interface	До 3 моделей/місяць, до 10M параметрів
Professional	$99/міс	Малі команди, стартапи	Необмежені моделі, API, пріоритетна обробка, email підтримка (48 год)	До 50M параметрів
Enterprise	від $5000/міс	Великі компанії	Необмежений розмір, on-premise, SLA 99.9%, TAM, кастомізація	Індивідуальні умови

Professional план за вартістю дев'яносто дев'ять доларів на місяць орієнтований на малі команди та стартапи. Він включає можливість стиснення необмеженої кількості моделей до п'ятдесяти мільйонів параметрів, пріоритетну обробку запитів, доступ до API для інтеграції у CI/CD pipelines, базову технічну підтримку через email протягом сорока восьми годин. Цей сегмент представляє основну масу очікуваних клієнтів на початкових етапах розвитку бізнесу.

Enterprise план з індивідуальним ціноутворенням від п'яти тисяч доларів на місяць призначений для великих компаній з промисловим використанням AI. Він забезпечує можливість роботи з моделями будь-якого розміру включаючи state-of-the-art трансформери з мільярдами параметрів, on-premise розгортання платформи у приватній інфраструктурі клієнта для максимальної безпеки та контролю, Service Level Agreement з гарантованим uptime дев'яносто дев'ять цілих дев'ять десятих відсотка, виділеного technical

account manager для персональної підтримки, можливість кастомізації алгоритмів під специфічні потреби клієнта.

Додаткові джерела доходу включають консалтингові послуги для компаній що потребують допомоги у впровадженні стиснутих моделей у production середовище. Очікувана вартість таких проектів від десяти до п'ятдесяти тисяч доларів залежно від складності. Licensing моделі для OEM партнерів, таких як виробники чіпів чи мобільних пристроїв, які захочуть інтегрувати технологію у свої продукти. Custom development для специфічних індустрій типу медицини, automotive чи фінансів, де потрібні domain-specific оптимізації.

Таблиця 4.3 – Ключові бізнес-метрики та targets

Метрика	*Target (12 міс)*	*Benchmark*	*Обґрунтування*
CAC (Customer Acquisition Cost)	$200	$150-300 B2B SaaS	Контент-маркетинг, конференції, partnerships
MRR (Monthly Recurring Revenue)	$100,000	–	1000 Professional users або 20 Enterprise
LTV (Customer Lifetime Value)	$3,000	–	$99/міс × 18 міс × 70% gross margin
LTV:CAC Ratio	6:1	>3:1 healthy	Sustainable economics
Churn Rate	<5%/міс	5-7% B2B SaaS	Product quality, customer success
Free-to-Paid Conversion	10%	2-5% typical	Strong value prop, low friction

Ключові метрики бізнесу охоплюють Customer Acquisition Cost очікувано близько двохсот доларів через комбінацію контент-маркетингу, конференцій та partnership з академічними інституціями. Monthly Recurring Revenue з цільовим показником досягнення ста тисяч доларів MRR через дванадцять місяців після запуску. Customer Lifetime Value оцінюється у три тисячі доларів для Professional сегменту при середній тривалості підписки

вісімнадцять місяців. Churn rate цільовий показник менше п'яти відсотків на місяць, що досягається через постійне покращення продукту та активну роботу з клієнтами. Стратегія ціноутворення враховує кілька важливих факторів. Value-based pricing базується на вартості яку клієнт отримує від використання сервісу, наприклад економія на інфраструктурі чи можливість вийти на новий ринок edge пристроїв. Competitive positioning встановлює ціни нижче enterprise рішень великих компаній але вище простих open-source інструментів, підкреслюючи баланс між доступністю та якістю. Flexibility у структурі планів дозволяє клієнтам починати з малого і легко масштабувати usage у міру зростання їх потреб.

4.4 Технологічний аудит та roadmap розвитку продукту

Поточний стан технології представляє собою research prototype з валідованим методом на дев'яти CNN архітектурах та експериментальною кодовою базою у Python. Для трансформації у commercial product необхідна суттєва інженерна робота по кількох напрямках. Розробка production-ready інфраструктури є першочерговим завданням. Backend система має бути побудована на мікросервісній архітектурі з використанням Kubernetes для оркестрації та автоматичного масштабування залежно від навантаження. API Gateway забезпечить єдину точку входу для клієнтів з rate limiting, authentication та моніторингом запитів. Model Processing Service відповідатиме за приймання моделей від користувачів, валідацію формату, виконання KAN-прунінгу та повернення стиснутих версій. Queue Management через RabbitMQ чи Apache Kafka забезпечить надійну обробку великої кількості concurrent запитів без втрати даних. Distributed storage на базі S3-compatible об'єктного сховища для зберігання моделей користувачів з автоматичним backup. Frontend інтерфейс має надавати інтуїтивний web-

based dashboard для завантаження моделей, налаштування параметрів стиснення, моніторингу прогресу обробки, візуалізації результатів включаючи графіки важливості фільтрів, порівняння метрик до та після стиснення. Інтеграційні можливості через RESTful API та Python SDK дозволять клієнтам автоматизувати процес стиснення у їх CI/CD pipelines.

GPU інфраструктура потребує уважного планування, оскільки навчання KAN-шарів є обчислювально інтенсивною операцією. Початково можна орендувати GPU інстанси у AWS, GCP чи Azure з автоматичним масштабуванням залежно від черги завдань. Середньостроково розглянути партнерство з GPU хмарними провайдерами типу Lambda Labs чи CoreWeave для оптимізації витрат. Довгостроково, при досягненні стабільного навантаження, економічно виправданим може стати придбання власного GPU кластеру. Розширення підтримки архітектур є критичним для ринкового впровадження. Перший етап після CNN має включати Vision Transformers (ViT, DeiT, Swin), оскільки вони домінують у сучасному комп'ютерному зорі. Мовні моделі (BERT, GPT варіанти) відкриють величезний NLP ринок, де проблема розміру моделей особливо гостра. Графові нейронні мережі для специфічних застосувань у пошуку лікарських засобів, рекомендаційних системах тощо. Користувацькі архітектури через гнучкий API, що дозволить користувачам адаптувати метод для власних архітектур. Автоматизація налаштування гіперпараметрів має зробити платформу справді доступною для непрофесіоналів. AutoML підхід для автоматичного підбору оптимальних параметрів grid_size, spline_order, розмірності bottleneck-шару залежно від характеристик конкретної моделі та датасету. Адаптивне стиснення, що автоматично визначає максимально можливий рівень стиснення при збереженні точності у заданих межах. Багатокритеріальна оптимізація для балансу між ступенем стиснення, погіршенням точності та швидкістю інференсу залежно від пріоритетів користувача. Інтеграція з популярними ML фреймворками значно спростить впровадження. Нативна інтеграція з PyTorch та TensorFlow/Keras дозволить користувачам використовувати

платформу безпосередньо у звичному робочому процесі. Формати експорту у ONNX, TorchScript, TFLite для розгортання на різних платформах. Hugging Face Hub інтеграція для безшовної роботи з моделями з найбільшого AI community hub.

Таблиця 4.4 – Product Roadmap (перший рік)

Квартал	*Ключові features*	*Технічні deliverables*	*Бізнес-цілі*
Q1	MVP launch	CNN pruning, web UI, auth, billing	50 freemium users
Q2	Production hardening	Auto-scaling, monitoring, security audit	10 paying customers, $1K MRR
Q3	Feature expansion	ViT support, advanced viz, API v2	30 customers, $5K MRR
Q4	Enterprise ready	On-premise, custom models, analytics	50 customers, $20K MRR, 2 Enterprise deals

Дорожня карта розвитку продукту розбита на чотири квартали першого року. Q1 присвячений розробці MVP (мінімально життєздатного продукту) з базовою функціональністю для CNN прунінгу, базовим веб-інтерфейсом, авторизацією та інтеграцією білінгу. Q2 фокусується на підготовці до промислового використання, включаючи автоматичне масштабування, повний моніторинг та системи сповіщення, аудит безпеки та тестування на проникнення. Q3 присвячений розширенню функціоналу з підтримкою Vision Transformers, розширеними можливостями візуалізації, API v2 з розширеним функціоналом. Q4 включає корпоративні функції, типу можливостей розгортання на локальних серверах, підтримки користувацьких моделей, розширеної панелі аналітики.

Технічні ризики та стратегії пом’якшення включають складність масштабування GPU-інфраструктури, що вирішується через партнерство зі спеціалізованими хмарними провайдерами та використання spot-інстансів

для не критичних навантажень. Забезпечення якості для різноманітних архітектур вирішується через широкий набір тестів з еталонними моделями та автоматизоване регресійне тестування. Безпека даних для моделей клієнтів забезпечується через шифрування у стані зберігання та під час передачі, SOC2 compliance, можливість розгортання на локальних серверах для найбільш чутливих варіантів використання.

4.5 Go-to-market стратегія та канали дистрибуції

Дорожня стратегія виходу на ринок IntelligentCompress базується на багатоканальному підході з фокусом на побудову довіри у технічній спільноті та демонстрації реальної цінності продукту через конкретні сценарії використання. Контент-маркетинг та лідерство думки є фундаментом стратегії на ранніх етапах. Публікація наукових статей у топових конференціях типу NeurIPS, ICML, CVPR забезпечить академічну легітимність та видимість у науковій спільноті. Технічний блог з детальними туторіалами, кейсовими дослідженнями та бенчмарками стане ресурсом для практиків, які шукають практичні рішення.

Внесок у відкритий код через випуск базової версії KAN-прунінгу як open-source бібліотеки створить доброчинну репутацію у спільноті та дозволить розробникам експериментувати з технологією перед переходом на платну версію. Whitepapers та технічна документація для корпоративних приймаючих рішень забезпечить глибоке розуміння технології перед її впровадженням. Програма підтримки розробників включатиме участь у major ML конференціях (NeurIPS, ICML, CVPR, ICLR) із доповідями та воркшопами. Хакатони та партнерства з університетами спрямовані на залучення молодих талантів та early adopters. Гостьові лекції у провідних університетах з сильними програмами ML сприятимуть формуванню

впізнаваності бренду серед майбутніх фахівців індустрії. YouTube технічний канал з туторіалами та live coding сесіями демонструватиме можливості платформи.

Партнерська екосистема критична для масштабування. Виробники обладнання (NVIDIA, Intel, ARM, Qualcomm) зацікавлені у демонстрації ефективності запуску стиснутих моделей на своєму обладнанні. Хмарні провайдери (AWS, GCP, Azure) можуть інтегрувати сервіс у свої ML платформи для додавання цінності клієнтам. ML платформи та інструменти (Hugging Face, Weights & Biases, Neptune) забезпечують безшовну інтеграцію у популярні робочі процеси. Системні інтегратори та консалтингові компанії допомагають у реалізації корпоративних проектів з використанням платформи.

Прямі продажі для enterprise сегменту включатимуть виділену команду продажів після досягнення product-market fit у сегменті SMB. Консультативний підхід з фокусом на прибутковість та бізнес-цінність замість лише технічних характеристик. Proof-of-concept проекти для ключових клієнтів дозволять продемонструвати цінність на їхніх реальних сценаріях перед фінальним рішенням. Reference customers з кейсами та відгуками забезпечать соціальне підтвердження.

Маркетингові канали, оптимізовані під технічну аудиторію, включають технічні спільноти (Reddit r/MachineLearning, Hacker News, ML Discord servers), де early adopters активно шукають нові рішення. Таргетована реклама у LinkedIn для залучення ML інженерів, дата-сайентістів та технічних рішень. Google Search Ads для ключових слів з високим наміром типу "neural network compression", "model optimization", "edge AI deployment". Спонсорство конференцій забезпечує видимість бренду серед цільової аудиторії.

Метрики для оцінки успіху виходу на ринок охоплюють:

- трафік на сайті — ціль: 5000 унікальних відвідувачів на місяць до кінця Q2;

- реєстрації freemium з конверсією 5% від відвідувачів;
- конверсія freemium → платна версія 10% протягом перших трьох місяців;
- pipeline для enterprise угод — ціль: 3 кваліфіковані можливості на місяць починаючи з Q3.

4.6 Команда та організаційна структура

Успішна реалізація стартап-проекту критично залежить від наявності команди з правильною комбінацією технічних навичок, доменної експертизи та підприємницького досвіду. Початкова команда IntelligentCompress має бути компактною, але покривати всі ключові функції.

Founding team складається з трьох ключових ролей:

1. Chief Technology Officer (CTO) відповідає за технічне бачення, архітектуру платформи та керівництво розробкою. Ідеально — PhD або сильний дослідницький бекграунд у machine learning, досвід розробки production ML систем, глибоке розуміння методів стиснення моделей.
2. Chief Executive Officer (CEO) фокусується на бізнес-стратегії, фандрейзингу, партнерствах та загальному виконанні планів. Необхідний досвід у tech startups, бажано у B2B SaaS, сильна мережа контактів у ML/AI екосистемі та здатність пояснювати технічну цінність непрофесіоналам.
3. Head of Engineering керує щоденною розробкою, наймом технічного персоналу, DevOps та інфраструктурою. Потрібен досвід lead engineering у scale-up компаніях, експертиза у розподілених системах та хмарній інфраструктурі, сильні практичні технічні навички.

На першому році команда Engineering зростає від початкових двох-трьох до восьми-десяти інженерів:

- backend engineers (2–3 особи) розробляють core API, обробчі пайплайни, шар бази даних.
- ML engineers (2–3 особи) імплементують алгоритми стиснення, процедури оптимізації, model serving.
- frontend engineer (1 особа) створює веб-панель та інтерфейси користувача.
- DevOps engineer (1 особа) забезпечує інфраструктуру, CI/CD, моніторинг.

Стратегія найму фокусується на senior hires з доведеною успішністю для швидкого виконання планів та подальшого менторства молодших членів команди.

Advisory board додає довіри та відкриває двері:

- technical advisors – визнані експерти у ML-спільноті, бажано з публікаціями у top venues, надають настанови по напрямку досліджень та валідовують технічний підхід;
- business advisors – досвід у масштабуванні B2B SaaS компаній, бажано з exits або старшими ролями у успішних стартапах, допомагають з go-to-market стратегією та фандрейзингом;
- industry advisors – експертиза у цільових вертикалях (mobile, IoT, automotive), забезпечують інсайти щодо специфічних потреб та больових точок.

Організаційна культура підкреслює кілька ключових цінностей:

- technical excellence через code reviews, тестування, стандарти документації забезпечує якість продукту;
- customer obsession через регулярну взаємодію з користувачами, врахування зворотного зв’язку, вимірювання задоволеності;
- transparency у внутрішніх комунікаціях, ухваленні рішень та відстеженні прогресу;

- work-life balance для сталості та утримання талантів у висококонкурентному ML-середовищі;
- remote-first: штаб-квартира у технічному хабі типу San Francisco або London, але гнучкість для глобального найму.

Стратегія компенсацій має бути конкурентною для залучення top talent у щільному ML-ринку праці:

- базова зарплата на рівні або трохи вище ринкового рівня для аналогічних ролей у добре фінансованих стартапах;
- значні гранти акцій (equity), оскільки ранні співробітники беруть на себе значний ризик, вестинг протягом 4 років з one-year cliff;
- бонуси за продуктивність прив’язані до індивідуальних та корпоративних цілей;
- пакет пільг, включно зі страхуванням здоров’я, бюджетом на навчання та участь у конференціях.

4.7 Фінансовий план та інвестиційні потреби

Фінансове планування IntelligentCompress базується на realistic assumptions про траєкторію зростання типову для B2B SaaS компаній в enterprise software space з врахуванням специфіки ML infrastructure бізнесу.

Початкові інвестиційні потреби seed round оцінюються у один мільйон доларів для перших дванадцяти-вісімнадцяти місяців операцій. Основні статті витрат включають personnel costs що складають шістдесят-сімдесят відсотків бюджету або близько шістсот тисяч доларів на рік для команди з шести-восьми людей з competitive compensation. Infrastructure costs включаючи cloud services, GPU instances, development tools оцінюються у сто п'ятдесят тисяч на рік з очікуваним зростанням у міру набору клієнтів. Marketing та sales витрати близько ста тисяч на рік для content creation,

conference attendance, initial advertising. Legal та administrative витрати включаючи incorporation, patents filing, accounting близько п'ятдесяти тисяч на рік. Office та equipment для remote-first команди мінімальні близько тридцять тисяч на рік.

Таблиця 4.5 – Структура seed round бюджету ($1M, 18 місяців)

Категорія витрат	*Сума ($K)*	*% бюджету*	*Деталізація*
Personnel	600	60%	6-8 engineers, competitive salaries + equity
Infrastructure	150	15%	Cloud services, GPU instances, dev tools
Marketing & Sales	100	10%	Content, conferences, ads, partnerships
Legal & Admin	50	5%	Incorporation, patents, accounting
Office & Equipment	30	3%	Remote-first setup, minimal overhead
Contingency	70	7%	Buffer для непередбачених витрат
Total	1000	100%	18 місяців runway

Таблиця 4.6 – Revenue Projections (перші 2 роки)

Період	*Paying Customers*	*MRR ($K)*	*ARR ($K)*	*Примітки*
Q1-Q2 Year 1	0	0	0	Product development phase
Q3 Year 1	5-10	5	60	Early adopters, Professional plan
Q4 Year 1	20-30	20	240	Organic growth + partnerships
Q1 Year 2	40-50	40	480	Marketing ramp-up
Q2 Year 2	60-80	60	720	First Enterprise deals (+$15K)
Q3 Year 2	90-120	90	1,080	Scale-up phase
Q4 Year 2	100-150	100-120	1,200-1,440	Series A ready (2-3 Enterprise)

Прогноз доходів базується на поступовому зростанні клієнтської бази з початковим фокусом на професійному рівні. Перші шість місяців присвячені розробці продукту з нульовим доходом. У третьому кварталі першого року очікується перші п’ять–десять платних клієнтів, переважно ранні користувачі професійного плану, що генерують близько $5,000 щомісячного доходу. У четвертому кварталі перший рік розширення до двадцяти–тридцяти клієнтів із доходом близько $20,000. На другий рік прогнозується прискорене зростання до 100–150 клієнтів наприкінці року, із щомісячним доходом $100,000–$120,000. Перші угоди з корпоративними клієнтами очікуються у третьому–четвертому кварталі другого року, що дасть значне збільшення доходів.

Економіка одиниці показує середній дохід на одного професійного клієнта $99 на місяць із середнім терміном співпраці 18 місяців, що дає життєву цінність клієнта $18,000. Вартість залучення клієнта через поєднання органічних та платних каналів оцінюється у $200–$300, що дає співвідношення LTV/CAC шість, що свідчить про здорову економіку. Валова маржа після врахування витрат на інфраструктуру очікується близько 80%, типово для SaaS. Ключові фінансові етапи включають досягнення беззбитковості у третьому–четвертому кварталі другого року при щомісячному доході $80,000–$90,000 та базі 100 клієнтів.

Цільовий раунд Series A заплановано на четвертий квартал другого року після підтвердження продуктово-ринкового відповідності та стабільного зростання доходів, з метою залучення $5–7 мільйонів для масштабування продажів та розширення продукту. Отримання прибутку не є першочерговою метою, акцент робиться на зростанні та захопленні ринку. Фінансування seed-раунду розподіляється: 50% на розробку продукту для команди розробників та інфраструктури, 30% на вихід на ринок, включаючи маркетинг, продажі та партнерства, 20% на операційні витрати, юридичні та адміністративні послуги і резерв.

Стратегія виходу передбачає кілька сценаріїв: придбання великим

хмарним провайдером (AWS, GCP, Azure) для інтеграції, стратегічне придбання виробником чипів (NVIDIA, Intel, Qualcomm), зацікавленим у програмному забезпеченні для демонстрації свого обладнання, або IPO як довгострокове бачення після досягнення значного масштабу та стабільної прибутковості. Довгострокова диверсифікація доходів може включати маркетплейс для готових стиснутих моделей, навчальні послуги з компресії моделей на власних наборах даних клієнтів, консалтингові та професійні послуги для корпоративних впроваджень, а також біллінг за білл-лейбл для OEM-партнерів.

4.8 Ризики та стратегії їх мітигації

Реалістична оцінка ризиків є критично важливою для успішного планування та виконання стартап-проекту. IntelligentCompress стикається з кількома категоріями ризиків, що потребують продуманих стратегій пом’якшення.

Технологічні ризики охоплюють можливість, що метод не масштабується на дуже великі моделі, типу GPT-4, із сотнями мільярдів параметрів через обчислювальні обмеження. Пом’якшення: раннє тестування на найбільших доступних моделях, архітектурні інновації для розподіленого навчання KAN на кількох GPU, партнерства з хмарними провайдерами для доступу до значних обчислювальних ресурсів. Поява конкурентних методів, які можуть виявитися більш ефективними. Пом’якшення: постійні дослідження та інновації, підтримка технічного лідерства, створення бар’єрів через патенти та власні оптимізації. Виклики інтеграції з різними ML-фреймворками та архітектурами. Пом’якшення: масштабне тестування, добре документовані API, підтримка інтеграції для корпоративних клієнтів.

Ринкові ризики включають повільніше, ніж очікувалось,

впровадження через консерватизм у закупівлі корпоративного програмного забезпечення, особливо для критичних систем. Пом’якшення: наявність сильних доказів ефективності від перших клієнтів, прозорі бенчмарки, безкоштовні пробні версії для безризикової оцінки. Конкуренція з боку добре фінансованих компаній, які можуть додати схожі функції до існуючих платформ. Пом’якшення: фокус на кращий користувацький досвід і результати, створення бар’єрів для переходу через тісну інтеграцію, розвиток спільноти для ефекту мережі. Економічний спад, який може зменшити IT-бюджети та попит на нові інструменти. Пом’якшення: демонстрація чіткого ROI, гнучке ціноутворення, фокус на економію витрат для клієнтів.

Операційні ризики включають утримання талантів у конкурентному ринку ML, де інженери отримують численні вигідні пропозиції. Пом’якшення: конкурентна компенсація, значущі долі в акціях, цікаві технічні завдання, сильна культура. Надійність інфраструктури критична для SaaS-бізнесу, де простої безпосередньо впливають на задоволеність клієнтів. Пом’якшення: надійна архітектура з резервуванням, комплексний моніторинг, процедури реагування на інциденти. Порушення безпеки особливо критичні, оскільки платформа обробляє цінну інтелектуальну власність клієнтів у вигляді їх моделей. Пом’якшення: безпека на першому місці, регулярні аудити, сертифікації, кіберстрахування.

Фінансові ризики включають витрати, що перевищують планові, якщо витрати на найм або інфраструктуру перевищують очікування. Пом’якшення: ретельне бюджетування, щомісячні перевірки, підтримка фінансового запасу. Складність у залученні наступного раунду фінансування, якщо метрики не відповідають очікуванням інвесторів. Пом’якшення: чітке визначення етапів, прозора комунікація з інвесторами, резервні плани для продовження фінансування. Ризик концентрації клієнтів, якщо доходи сильно залежать від невеликої кількості корпоративних клієнтів. Пом’якшення: диверсифікація сегментів, контрактні захисти, сильна програма підтримки клієнтів.

Регуляторні ризики менш критичні для програмної інфраструктури, але можуть включати потенційні правила щодо штучного інтелекту, що впливають на використання стиснутих моделей у регульованих галузях. Пом’якшення: проактивний контроль відповідності, створення аудиторських слідів та функцій пояснюваності, участь у регуляторних обговореннях. Ризики інтелектуальної власності включають можливі претензії на патентні порушення або неможливість патентування основної технології. Пом’якшення: комплексний пошук аналогів, сильний патентний портфель, аналіз свободи дій, захисні публікації.

Стратегічні ризики включають необхідність зміни курсу, якщо початковий ринок виявляється менш сприятливим, ніж очікувалося. Пом’якшення: дослідження клієнтів перед значними інвестиціями, гнучкість у дорожній карті продукту, моніторинг кількох потенційних ринків. Конфлікти в команді, особливо між технічними та бізнес-засновниками з різними пріоритетами. Пом’якшення: чіткий розподіл ролей, регулярні узгоджувальні зустрічі, зовнішні консультанти для медіації, графіки вестингу з «cliffs».

Висновки до розділу 4

У четвертому розділі проведено детальну розробку стартап-проекту IntelligentCompress на основі методу KAN-керованого структурного прунінгу нейронних мереж. Проект являє собою SaaS-платформу для автоматизованого інтелектуального стиснення AI-моделей з метою їх впровадження на обмежених обчислювальних ресурсах. Аналіз ринкового середовища показав значний потенціал із прогнозованим обсягом ринку edge AI у 50 мільярдів доларів до 2030 року та складним річним темпом зростання близько 20%. Основними драйверами є масове поширення IoT-пристроїв,

регуляторні вимоги щодо приватності даних, зростання вартості електроенергії та розвиток автономних систем. Конкурентний аналіз виявив, що існуючі рішення від великих технологічних компаній та спеціалізованих стартапів мають суттєві обмеження.

Ключові переваги IntelligentCompress базуються на семантично обґрунтованому підході до прунінгу, що забезпечує кращу якість результатів, швидкість стиснення п’ять-шість годин замість днів, незалежність від типу обладнання та інтерпретованість через візуалізацію функцій важливості. Бізнес-модель передбачає багаторівневу SaaS-стратегію з Freemium-планом для індивідуальних розробників, Professional-планом за $99 на місяць для малих команд та Enterprise-планом від $5,000 на місяць для корпорацій. Додаткові джерела доходу включають консалтингові послуги, ліцензування для OEM-партнерів та замовні розробки.

Ключові метрики передбачають досягнення $100,000 MRR через 12 місяців та Customer Lifetime Value $3,000 при CAC $200. Технологічна дорожня карта визначає критичні компоненти для трансформації дослідного прототипу у платформу, готову для впровадження, включно з мікросервісною бекенд-архітектурою, інтуїтивним веб-інтерфейсом, GPU-інфраструктурою з автоматичним масштабуванням, підтримкою Vision Transformers та Language Models, автоматизацією налаштування гіперпараметрів, інтеграцією з популярними ML-фреймворками.

План розвитку розбитий на чотири квартали: MVP у Q1, підготовка до продакшену у Q2, розширення функцій у Q3 та корпоративні можливості у Q4. Стратегія виходу на ринок фокусується на побудові довіри в технічній спільноті через контент-маркетинг, наукові публікації, внесок у відкритий код, програму підтримки розробників, партнерства з виробниками обладнання та хмарними провайдерами. Прямі продажі для корпоративного сегменту запускаються після досягнення відповідності продукту ринку у SMB-сегменті.

Фінансовий план передбачає seed-раунд $1 млн на перші 18 місяців з витратами на персонал 60–70% бюджету. Прогноз доходів показує зростання від нуля до $100,000–$120,000 MRR протягом другого року з беззбитковістю у Q3–Q4 та ціллю Series A $5–7 млн для масштабування. Економіка одиниці демонструє здорове співвідношення LTV/CAC = 6 та валову маржу близько 80%.

Комплексний аналіз ризиків виявив технологічні, ринкові, операційні, фінансові, регуляторні та стратегічні загрози з детальними стратегіями пом’якшення для кожної категорії. Особлива увага приділена утриманню талантів, надійності інфраструктури, безпеці, конкуренції з боку великих гравців та часу виходу на ринок.

Проект IntelligentCompress має потенціал стати значним гравцем на швидкозростаючому ринку інструментів для AI-інфраструктури завдяки поєднанню унікальної технології з міцною науковою базою, чітким ціннісним пропозиціям для кількох сегментів клієнтів, реалістичним планом виконання та досвідченою командою.

ВИСНОВКИ

У магістерській дисертації вирішено актуальну науково-практичну задачу розробки методу інтелектуального структурного прунінгу згорткових нейронних мереж на основі мереж Колмогорова-Арнольда для ефективного стиснення моделей з мінімальною втратою якості.

Проведено комплексний аналіз сучасних методів стиснення нейронних мереж, що дозволило виявити ключові обмеження існуючих підходів. Традиційний magnitude pruning базується на простих статистичних метриках без урахування семантичної важливості компонентів моделі. Knowledge distillation потребує повного перенавчання з нуля, що займає сотні GPU-годин. Квантизація не зменшує кількість параметрів пропорційно зменшенню розміру моделі.

Розроблено новий метод KAN-керованого структурного прунінгу, який використовує інтерпретабельні властивості мереж Колмогорова-Арнольда для визначення справжньої семантичної важливості фільтрів через аналіз їх впливу на латентний простір класифікаційної задачі. Метод базується на навчанні гібридної CNN-KAN архітектури з KAN-bottleneck після останнього згорткового шару та використанні градієнтів функцій важливості для ідентифікації критичних компонентів моделі.

Проведено всебічну експериментальну валідацію методу на датасеті CIFAR-10 з використанням дев'яти різних CNN архітектур від класичних ResNet та VGG до сучасних ефективних дизайнів MobileNet, EfficientNet та ультракомпактних ShuffleNet і SqueezeNet. Результати продемонстрували стабільну статистично значущу перевагу KAN-керованого прунінгу над традиційним magnitude pruning на всіх дев'яти архітектурах без виключень. При помірній компресії близько 1.3× середня перевага становить +1.2 процентних пункти точності, що відповідає відносному зменшенню помилки

на 11.3%. При агресивній компресії близько 1.7× перевага зростає до +2.1 pp або 15.2% відносного зменшення помилки.

Аналіз кореляції між KAN-важливістю фільтрів та їх L1-нормою виявив помірну відповідність r=0.62, що підтверджує здатність методу виявляти семантично важливі компоненти які magnitude pruning помилково класифікує як неважливі. Ідентифіковано 47 фільтрів з високою семантичною важливістю але малою величиною ваг, що пояснює кращу якість результатів KAN-керованого підходу.

Архітектурний аналіз показав що residual мережі найстійкіші до прунінгу завдяки skip connections з деградацією 0.9-2.5%, sequential архітектури найвразливіші через відсутність альтернативних шляхів інформації з деградацією 2.9-9.4%, а dense архітектури демонструють найкращі абсолютні результати завдяки feature reuse з деградацією лише 0.5-1.6% навіть при компресії 2.21×.

Обчислювальна ефективність методу підтверджена детальними вимірюваннями з зменшенням FLOPS на 20-31%, практичним прискоренням інференсу 1.19×-1.42× та економією GPU пам'яті 21-36%. Час виконання повного циклу стиснення становить 5-6 годин включаючи навчання KAN та ітеративний прунінг, що значно швидше порівняно з 200 епохами повного навчання для knowledge distillation.

Розроблено концепцію стартап-проекту IntelligentCompress для комерціалізації методу у формі SaaS платформи автоматизованого інтелектуального стиснення нейронних мереж. Проведено аналіз ринкового середовища з прогнозованим обсягом edge AI ринку 50 мільярдів доларів до 2030 року, конкурентний аналіз існуючих рішень, розроблено бізнес-модель з багаторівневою стратегією монетизації та фінансовий план з seed round один мільйон доларів для перших вісімнадцяти місяців операцій.

Результати дисертації мають як наукову так і практичну цінність. Метод KAN-керованого структурного прунінгу відкриває новий напрямок використання інтерпретабельних властивостей мереж Колмогорова-Арнольда

для оптимізації нейронних мереж та може бути адаптований для інших архітектур і доменів застосування. Практична реалізація у формі платформи IntelligentCompress має потенціал зробити передові AI моделі доступними для застосування на обмежених обчислювальних ресурсах без критичної втрати якості.

ПЕРЕЛІК ПОСИЛАНЬ

1. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778. doi: 10.1109/CVPR.2016.90
2. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556, 2014. https://arxiv.org/abs/1409.1556
3. Howard A. G., Zhu M., Chen B., et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861, 2017. https://arxiv.org/abs/1704.04861
4. Tan M., Le Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:6105-6114, 2019. https://proceedings.mlr.press/v97/tan19a.html
5. Huang G., Liu Z., Van Der Maaten L., Weinberger K. Q. Densely Connected Convolutional Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 2261-2269. doi: 10.1109/CVPR.2017.243
6. Ma N., Zhang X., Zheng H.-T., Sun J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In: Ferrari V., Hebert M., Sminchisescu C., Weiss Y. (eds) Computer Vision – ECCV 2018. Lecture Notes in Computer Science, vol 11218. Springer, Cham. https://doi.org/10.1007/978-3-030-01264-9_8
7. Iandola F. N., Han S., Moskewicz M. W., et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv preprint arXiv:1602.07360, 2016. https://arxiv.org/abs/1602.07360
8. Krizhevsky A. Learning Multiple Layers of Features from Tiny Images.

Technical Report, University of Toronto, 2009. https://www.cs.toronto.edu/~kriz/cifar.html

9. Han S., Pool J., Tran J., Dally W. J. Learning both Weights and Connections for Efficient Neural Networks. Advances in Neural Information Processing Systems 28 (NIPS 2015), pp. 1135-1143. https://proceedings.neurips.cc/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html
10. Li H., Kadav A., Durdanovic I., Samet H., Graf H. P. Pruning Filters for Efficient ConvNets. arXiv preprint arXiv:1608.08710, 2016. https://arxiv.org/abs/1608.08710
11. He Y., Zhang X., Sun J. Channel Pruning for Accelerating Very Deep Neural Networks. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 1398-1406. doi: 10.1109/ICCV.2017.155
12. Molchanov P., Tyree S., Karras T., Aila T., Kautz J. Pruning Convolutional Neural Networks for Resource Efficient Inference. arXiv preprint arXiv:1611.06440, 2016. https://arxiv.org/abs/1611.06440
13. Lin M., Ji R., Wang Y., et al. HRank: Filter Pruning using High-Rank Feature Map. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 1529-1538. doi: 10.1109/CVPR42600.2020.00160
14. Liu Z., Li J., Shen Z., et al. Learning Efficient Convolutional Networks through Network Slimming. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 2755-2763. doi: 10.1109/ICCV.2017.298
15. Hinton G., Vinyals O., Dean J. Distilling the Knowledge in a Neural Network. arXiv preprint arXiv:1503.02531, 2015. https://arxiv.org/abs/1503.02531
16. Romero A., Ballas N., Kahou S. E., et al. FitNets: Hints for Thin Deep Nets. arXiv preprint arXiv:1412.6550, 2014.

https://arxiv.org/abs/1412.6550

17. Jacob B., Kligys S., Chen B., et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 2704-2713. doi: 10.1109/CVPR.2018.00286
18. Han S., Mao H., Dally W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv preprint arXiv:1510.00149, 2015. https://arxiv.org/abs/1510.00149
19. Kolmogorov A. N. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. Doklady Akademii Nauk SSSR, 114:953–956, 1957.
20. Liu Z., Wang Y., Vaidya S., et al. KAN: Kolmogorov-Arnold Networks. arXiv preprint arXiv:2404.19756, 2024. https://arxiv.org/abs/2404.19756
21. Breuer S., Bürger A., Kilian M., Memmesheimer R., Paulus D. Kolmogorov-Arnold Networks (KANs) for Time Series Analysis. arXiv preprint arXiv:2405.08790, 2024. https://arxiv.org/abs/2405.08790
22. Bodner A. D., Tancik M., Garbin S. J. KAN You See It? KANs and Sentinel for Effective and Explainable Crop Field Segmentation. arXiv preprint arXiv:2408.07040, 2024. https://arxiv.org/abs/2408.07040
23. Schumaker L. L. Spline Functions: Basic Theory. Cambridge University Press, 3rd edition, 2007. ISBN: 978-0521705127
24. De Boor C. A Practical Guide to Splines. Springer-Verlag New York, Revised Edition, 2001. ISBN: 978-0387953663
25. Paszke A., Gross S., Massa F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32 (NeurIPS 2019), pp. 8024-8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
26. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd

International Conference on Machine Learning, PMLR 37:448-456, 2015. https://proceedings.mlr.press/v37/ioffe15.html

27. Loshchilov I., Hutter F. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv preprint arXiv:1608.03983, 2016. https://arxiv.org/abs/1608.03983
28. Micikevicius P., Narang S., Alben J., et al. Mixed Precision Training. arXiv preprint arXiv:1710.03740, 2017. https://arxiv.org/abs/1710.03740
29. LeCun Y., Denker J., Solla S. Optimal Brain Damage. Advances in Neural Information Processing Systems 2 (NIPS 1989), pp. 598-605. https://proceedings.neurips.cc/paper/1989/hash/6c9882bbac1c7093bd25041881277658-Abstract.html
30. Frankle J., Carbin M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. arXiv preprint arXiv:1803.03635, 2018. https://arxiv.org/abs/1803.03635
31. Blalock D., Ortiz J. J. G., Frankle J., Guttag J. What is the State of Neural Network Pruning? Proceedings of Machine Learning and Systems 2020, pp. 129-146. https://proceedings.mlsys.org/paper/2020/hash/d2ddea18f00665ce8623e36bd4e3c7c5-Abstract.html
32. Zagoruyko S., Komodakis N. Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer. arXiv preprint arXiv:1612.03928, 2016. https://arxiv.org/abs/1612.03928
33. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. ISBN: 978-0262035613. http://www.deeplearningbook.org
34. Bishop C. M. Pattern Recognition and Machine Learning. Springer, 2006. ISBN: 978-0387310732
35. Murphy K. P. Machine Learning: A Probabilistic Perspective. MIT Press, 2012. ISBN: 978-0262018029

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, Subset
import torchvision
import torchvision.transforms as transforms
from torchvision import models

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.auto import tqdm
import time
import copy
from pathlib import Path

from efficient_kan import KAN

# Set random seeds for reproducibility
def set_seed(seed=42):
   torch.manual_seed(seed)
   torch.cuda.manual_seed_all(seed)
   np.random.seed(seed)
   torch.backends.cudnn.deterministic = True
   torch.backends.cudnn.benchmark = False

set_seed(42)

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
if torch.cuda.is_available():
   print(f"GPU: {torch.cuda.get_device_name(0)}")
   print(f"Memory: {torch.cuda.get_device_properties(0).total_memory /
1024**3:.2f} GB")


class ExperimentConfig:
   """
   Централізована конфігурація для всіх експериментів.
   """
   def __init__(self):
      # Dataset settings
      self.dataset = 'CIFAR10'
      self.batch_size = 128
      self.num_workers = 4

      # Training settings
      self.epochs_pretrain = 50
      self.epochs_finetune = 10
      self.lr_pretrain = 0.1
      self.lr_finetune = 0.001

      # KAN settings
      self.kan_latent_dim = 64
      self.kan_grid_size = 5
```

```
      self.kan_spline_order = 3

      # Pruning settings
      self.num_iterations = 5
      self.prune_percent_per_iter = 0.15
      self.max_accuracy_drop = 5.0

      # Paths
      self.save_dir = Path('./experiments')
      self.save_dir.mkdir(exist_ok=True)

   def __repr__(self):
      return f"ExperimentConfig(\n" + "\n".join([f"  {k}={v}" for k, v in
self.__dict__.items()]) + "\n)"

config = ExperimentConfig()
print(config)

# CIFAR-10 normalization constants
CIFAR10_MEAN = (0.4914, 0.4822, 0.4465)
CIFAR10_STD = (0.2470, 0.2435, 0.2616)

# Augmentation for training
transform_train = transforms.Compose([
   transforms.RandomCrop(32, padding=4),
   transforms.RandomHorizontalFlip(),
   transforms.ToTensor(),
   transforms.Normalize(CIFAR10_MEAN, CIFAR10_STD),
])

# No augmentation for validation
transform_test = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize(CIFAR10_MEAN, CIFAR10_STD)
])

# Load datasets
trainset = torchvision.datasets.CIFAR10(
   root='./data',
   train=True,
   download=True,
   transform=transform_train
)

testset = torchvision.datasets.CIFAR10(
   root='./data',
   train=False,
   download=True,
   transform=transform_test
)

trainloader = DataLoader(
   trainset,
   batch_size=config.batch_size,
   shuffle=True,
   num_workers=0,
   pin_memory=True
)

testloader = DataLoader(
   testset,
```

```
   batch_size=config.batch_size * 2,
   shuffle=False,
   num_workers=0,
   pin_memory=True
)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
'ship', 'truck')
print(f"Training samples: {len(trainset):,}")
print(f"Test samples: {len(testset):,}")


class CNNBackboneRegistry:
   """
   Реєстр канонічних CNN архітектур для експериментів.
   """

   @staticmethod
   def get_feature_extractor(name, pretrained=False):
      if name == 'resnet18':
         model = models.resnet18(weights='IMAGENET1K_V1' if pretrained else
None)
         model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
         model.maxpool = nn.Identity()
         feature_dim = 512
         model.fc = nn.Identity()

      elif name == 'resnet34':
         model = models.resnet34(weights='IMAGENET1K_V1' if pretrained else
None)
         model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
         model.maxpool = nn.Identity()
         feature_dim = 512
         model.fc = nn.Identity()

      elif name == 'resnet50':
         model = models.resnet50(weights='IMAGENET1K_V1' if pretrained else
None)
         model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
bias=False)
         model.maxpool = nn.Identity()
         feature_dim = 2048
         model.fc = nn.Identity()

      elif name == 'vgg11_bn':
         model = models.vgg11_bn(weights='IMAGENET1K_V1' if pretrained else
None)
         feature_dim = 512
         model.classifier = nn.Identity()

      elif name == 'vgg16_bn':
         model = models.vgg16_bn(weights='IMAGENET1K_V1' if pretrained else
None)
         feature_dim = 512
         model.classifier = nn.Identity()

      elif name == 'mobilenet_v2':
         model = models.mobilenet_v2(weights='IMAGENET1K_V1' if pretrained
```

```
else None)
         feature_dim = 1280
         model.classifier = nn.Identity()

      elif name == 'efficientnet_b0':
         model = models.efficientnet_b0(weights='IMAGENET1K_V1' if pretrained
else None)
         feature_dim = 1280
         model.classifier = nn.Identity()

      elif name == 'densenet121':
         model = models.densenet121(weights='IMAGENET1K_V1' if pretrained else
None)
         feature_dim = 1024
         model.classifier = nn.Identity()

      elif name == 'shufflenet_v2_x1_0':
         model = models.shufflenet_v2_x1_0(weights='IMAGENET1K_V1' if
pretrained else None)
         feature_dim = 1024
         model.fc = nn.Identity()

      elif name == 'squeezenet1_1':
         model = models.squeezenet1_1(weights='IMAGENET1K_V1' if pretrained
else None)
         feature_dim = 512
         model.classifier = nn.Identity()

      else:
         raise ValueError(f"Unknown architecture: {name}")

      return model, feature_dim

AVAILABLE_ARCHITECTURES = [
   'resnet18', 'resnet34', 'resnet50',
   'vgg11_bn', 'vgg16_bn',
   'mobilenet_v2', 'efficientnet_b0',
   'densenet121', 'shufflenet_v2_x1_0', 'squeezenet1_1'
]

print("Available CNN architectures:")
for i, arch in enumerate(AVAILABLE_ARCHITECTURES, 1):
   print(f" {i}. {arch}")

class CNNKANClassifier(nn.Module):
   """
   Гібридна архітектура: CNN backbone + KAN bottleneck + Linear classifier

   FIXED: Правильна обробка виходів різних backbone архітектур
   """

   def __init__(self, backbone_name, num_classes=10, kan_latent_dim=64,
                kan_grid_size=5, kan_spline_order=3, pretrained=False):
      super().__init__()

      self.backbone_name = backbone_name
      self.num_classes = num_classes
      self.kan_latent_dim = kan_latent_dim

      # Get CNN backbone
```

```
        self.backbone, self.feature_dim =
CNNBackboneRegistry.get_feature_extractor(
            backbone_name, pretrained=pretrained
        )

        # Global Average Pooling
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))

        # KAN bottleneck layer
        self.kan_bottleneck = KAN(
            layers_hidden=[self.feature_dim, kan_latent_dim],
            grid_size=kan_grid_size,
            spline_order=kan_spline_order,
            scale_noise=0.1,
            scale_base=1.0,
            scale_spline=1.0,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
        )

        # Final linear classifier
        self.classifier = nn.Linear(kan_latent_dim, num_classes)

    def forward(self, x):
        # Extract features through CNN backbone
        features = self.backbone(x)

        # Handle different output formats
        if len(features.shape) == 4:  # Spatial features (B, C, H, W)
            features = self.avgpool(features)
            features = features.view(features.size(0), -1)
        elif len(features.shape) == 2:  # Already flattened (B, C)
            pass
        else:
            raise ValueError(f"Unexpected feature shape: {features.shape}")

        # Pass through KAN bottleneck
        latent = self.kan_bottleneck(features)

        # Final classification
        logits = self.classifier(latent)

        return logits

    def get_feature_dim(self):
        return self.feature_dim

    def count_parameters(self):
        return sum(p.numel() for p in self.parameters() if p.requires_grad)

    def __repr__(self):
        param_count = self.count_parameters()
        return (f"CNNKANClassifier(\n"
                f"  backbone={self.backbone_name},\n"
                f"  feature_dim={self.feature_dim},\n"
                f"  kan_latent_dim={self.kan_latent_dim},\n"
                f"  num_classes={self.num_classes},\n"
                f"  total_params={param_count:,}\n"
```

```
            f")")

def train_epoch(model, dataloader, criterion, optimizer, device,
desc="Training"):
   """Train for one epoch"""
   model.train()
   running_loss = 0.0
   correct = 0
   total = 0

   pbar = tqdm(dataloader, desc=desc, leave=False)
   for inputs, targets in pbar:
      inputs, targets = inputs.to(device), targets.to(device)

      optimizer.zero_grad()
      outputs = model(inputs)
      loss = criterion(outputs, targets)
      loss.backward()

      torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
      optimizer.step()

      running_loss += loss.item()
      _, predicted = outputs.max(1)
      total += targets.size(0)
      correct += predicted.eq(targets).sum().item()

      pbar.set_postfix({
         'loss': running_loss / (pbar.n + 1),
         'acc': 100. * correct / total
      })

   return running_loss / len(dataloader), 100. * correct / total

def evaluate(model, dataloader, criterion, device, desc="Evaluating"):
   """Evaluate model"""
   model.eval()
   running_loss = 0.0
   correct = 0
   total = 0

   with torch.no_grad():
      pbar = tqdm(dataloader, desc=desc, leave=False)
      for inputs, targets in pbar:
         inputs, targets = inputs.to(device), targets.to(device)
         outputs = model(inputs)
         loss = criterion(outputs, targets)

         running_loss += loss.item()
         _, predicted = outputs.max(1)
         total += targets.size(0)
         correct += predicted.eq(targets).sum().item()

         pbar.set_postfix({
            'loss': running_loss / (pbar.n + 1),
            'acc': 100. * correct / total
         })

   return running_loss / len(dataloader), 100. * correct / total
```

```
def train_model(model, trainloader, testloader, epochs, lr, device,
           model_name="model", save_best=True):
   """Complete training loop"""
   model = model.to(device)
   criterion = nn.CrossEntropyLoss()
   optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9,
weight_decay=5e-4)
   scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

   history = {
      'train_loss': [], 'train_acc': [],
      'test_loss': [], 'test_acc': []
   }

   best_acc = 0.0
   best_model_state = None

   for epoch in range(epochs):
      train_loss, train_acc = train_epoch(
         model, trainloader, criterion, optimizer, device,
         desc=f'Epoch {epoch+1}/{epochs} [Train]'
      )

      test_loss, test_acc = evaluate(
         model, testloader, criterion, device,
         desc=f'Epoch {epoch+1}/{epochs} [Test]'
      )

      scheduler.step()

      history['train_loss'].append(train_loss)
      history['train_acc'].append(train_acc)
      history['test_loss'].append(test_loss)
      history['test_acc'].append(test_acc)

      if test_acc > best_acc:
         best_acc = test_acc
         best_model_state = copy.deepcopy(model.state_dict())

      if (epoch + 1) % 10 == 0 or epoch == epochs - 1:
         print(f'Epoch {epoch+1}/{epochs}: '
              f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}% | '
              f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}% | '
              f'Best: {best_acc:.2f}%')

   if save_best and best_model_state is not None:
      model.load_state_dict(best_model_state)

   return model, history, best_acc
class KANInterpreter:
   """
   Аналіз та візуалізація того, як KAN "бачить" ознаки з CNN.
   """

   def __init__(self, model):
      self.model = model
      self.kan_layer = model.kan_bottleneck.layers[0]
      self.feature_dim = model.get_feature_dim()
      self.latent_dim = model.kan_latent_dim
```

```
    def compute_filter_importance(self, method='max'):
        """Обчислює важливість кожного CNN фільтра"""
        with torch.no_grad():
            spline_weights = self.kan_layer.scaled_spline_weight.cpu()
            influence_matrix = torch.norm(spline_weights, p=2, dim=2)

            if method == 'max':
                importance = torch.max(influence_matrix, dim=0).values
            elif method == 'mean':
                importance = torch.mean(influence_matrix, dim=0)
            elif method == 'l2':
                importance = torch.norm(influence_matrix, p=2, dim=0)
            else:
                raise ValueError(f"Unknown method: {method}")

            return importance.numpy(), influence_matrix.numpy()

    def visualize_influence_matrix(self, save_path=None):
        """Візуалізує матрицю впливу"""
        _, influence_matrix = self.compute_filter_importance()

        plt.figure(figsize=(16, 8))
        sns.heatmap(
            influence_matrix,
            cmap='viridis',
            xticklabels=50,
            yticklabels=[f'L{i}' for i in range(self.latent_dim)],
            cbar_kws={'label': 'Influence Magnitude'}
        )
        plt.xlabel('CNN Filter Index')
        plt.ylabel('Latent Neuron')
        plt.title(f'KAN Influence Matrix\n({self.feature_dim} filters →
{self.latent_dim} latent neurons)')
        plt.tight_layout()

        if save_path:
            plt.savefig(save_path, dpi=300, bbox_inches='tight')
        plt.show()

        return influence_matrix

    def analyze_latent_neuron(self, latent_idx, top_k=10):
        """Аналізує конкретний латентний нейрон"""
        _, influence_matrix = self.compute_filter_importance()
        influences = influence_matrix[latent_idx, :]

        top_indices = np.argsort(influences)[-top_k:][::-1]
        top_values = influences[top_indices]

        print(f"\n{'='*60}")
        print(f"Analysis of Latent Neuron L{latent_idx}")
        print(f"{'='*60}")
        print(f"\nTop {top_k} most influential CNN filters:")
        for rank, (idx, value) in enumerate(zip(top_indices, top_values), 1):
            print(f" #{rank}: Filter {idx:3d} (influence: {value:.4f})")

        return top_indices, top_values

    def visualize_spline_functions(self, latent_idx, filter_indices,
```

```
save_path=None):
      """Візуалізує навчені сплайн-функції"""
      num_filters = len(filter_indices)
      fig, axes = plt.subplots(1, num_filters, figsize=(5*num_filters, 4))
      if num_filters == 1:
         axes = [axes]

      with torch.no_grad():
         kan_layer = self.kan_layer

         for ax, filter_idx in zip(axes, filter_indices):
            grid = kan_layer.grid[filter_idx].cpu()
            spline_coeffs = kan_layer.scaled_spline_weight[latent_idx,
filter_idx, :].cpu()
            spline_order = kan_layer.spline_order

            x_min = grid[spline_order].item()
            x_max = grid[-spline_order-1].item()
            x_eval = torch.linspace(x_min, x_max, 200)

            y_eval = torch.zeros_like(x_eval)
            for i, x in enumerate(x_eval):
               mask = (grid[:-1] <= x) & (x < grid[1:])
               if mask.any():
                  idx = mask.nonzero()[0].item()
                  t = (x - grid[idx]) / (grid[idx+1] - grid[idx])
                  y_eval[i] = (1-t) * spline_coeffs[idx] + t *
spline_coeffs[idx+1]

            ax.plot(x_eval.numpy(), y_eval.numpy(), 'b-', linewidth=2.5)
            ax.axhline(0, color='gray', linestyle='--', linewidth=0.5,
alpha=0.5)
            ax.axvline(0, color='gray', linestyle='--', linewidth=0.5,
alpha=0.5)
            ax.grid(True, alpha=0.3)
            ax.set_xlabel('Filter Activation')
            ax.set_ylabel(f'Contribution to L{latent_idx}')
            ax.set_title(f'Filter #{filter_idx}')

      fig.suptitle(f'Learned Functions for Latent Neuron L{latent_idx}',
fontsize=14, fontweight='bold')
      plt.tight_layout()

      if save_path:
         plt.savefig(save_path, dpi=300, bbox_inches='tight')
      plt.show()

   def analyze_class_specific_features(self, class_idx, class_name):
      """Аналізує ознаки для конкретного класу"""
      classifier_weights = self.model.classifier.weight.detach().cpu().numpy()
      class_weights = classifier_weights[class_idx, :]

      top_latent_indices = np.argsort(np.abs(class_weights))[-3:][::-1]

      print(f"\n{'='*60}")
      print(f"Analysis for Class: '{class_name}' (index {class_idx})")
      print(f"{'='*60}")
      print(f"\nTop 3 most important latent neurons:")
      for rank, idx in enumerate(top_latent_indices, 1):
         print(f" #{rank}: Latent Neuron L{idx} (weight:
```

```
{class_weights[idx]:.4f})")

      most_important_latent = top_latent_indices[0]
      print(f"\nAnalyzing Latent Neuron L{most_important_latent}:")
      top_filters, _ = self.analyze_latent_neuron(most_important_latent,
top_k=10)

      return most_important_latent, top_filters
class FilterMask:
   """
   Універсальна маска для структурного прунінгу через обнулення ваг.
   Працює з будь-якою CNN архітектурою без модифікації структури.
   """

   def __init__(self, model):
      self.model = model
      self.masks = {}
      self.pruned_filters = set()
      self._identify_prunable_layers()

   def _identify_prunable_layers(self):
      """Знаходить останній згортковий шар для прунінгу"""
      self.prunable_layers = []

      for name, module in self.model.backbone.named_modules():
         if isinstance(module, nn.Conv2d):
            self.prunable_layers.append((name, module))

      if not self.prunable_layers:
         raise ValueError("No Conv2d layers found!")

      self.target_layer_name, self.target_layer = self.prunable_layers[-1]
      self.num_filters = self.target_layer.out_channels

      print(f"🎯 Target layer: {self.target_layer_name}")
      print(f"   Filters: {self.num_filters}")
      print(f"   Shape: {self.target_layer.weight.shape}")

      self.mask = torch.ones(self.num_filters,
device=next(self.model.parameters()).device)

   def prune_filters(self, filter_indices):
      """Обнуляє фільтри та їх градієнти"""
      filter_indices = torch.tensor(filter_indices, device=self.mask.device)

      print(f"\n🔪 Masking {len(filter_indices)} filters...")

      self.mask[filter_indices] = 0.0
      self.pruned_filters.update(filter_indices.cpu().numpy())

      with torch.no_grad():
         # Zero conv weights
         self.target_layer.weight[filter_indices, :, :, :] = 0.0
         if self.target_layer.bias is not None:
            self.target_layer.bias[filter_indices] = 0.0

         # Zero BatchNorm
         self._zero_corresponding_batchnorm(filter_indices)

         # Zero KAN inputs
```

```
            self._zero_kan_inputs(filter_indices)

        num_active = (self.mask > 0).sum().item()
        print(f"  Active: {num_active}/{self.num_filters}
({num_active/self.num_filters*100:.1f}%)")

    def _zero_corresponding_batchnorm(self, filter_indices):
        """Обнуляє BatchNorm для прунутих фільтрів"""
        found_target = False
        for name, module in self.model.backbone.named_modules():
            if found_target and isinstance(module, (nn.BatchNorm2d,
nn.BatchNorm1d)):
                module.weight[filter_indices] = 0.0
                module.bias[filter_indices] = 0.0
                module.running_mean[filter_indices] = 0.0
                module.running_var[filter_indices] = 1.0
                print(f"  ✓ Zeroed BatchNorm: {name}")
                break
            if name == self.target_layer_name:
                found_target = True

    def _zero_kan_inputs(self, filter_indices):
        """Обнуляє входи до KAN"""
        kan_layer = self.model.kan_bottleneck.layers[0]
        kan_layer.spline_weight[:, filter_indices, :] = 0.0
        if hasattr(kan_layer, 'base_weight') and kan_layer.base_weight is not
None:
            kan_layer.base_weight[:, filter_indices] = 0.0
        print(f"  ✓ Zeroed KAN inputs")

    def apply_mask_to_gradients(self):
        """Застосовує маску до градієнтів (заморожує прунуті фільтри)"""
        if self.target_layer.weight.grad is not None:
            mask_expanded = self.mask.view(-1, 1, 1, 1)
            self.target_layer.weight.grad *= mask_expanded

            if self.target_layer.bias is not None and self.target_layer.bias.grad
is not None:
                self.target_layer.bias.grad *= self.mask

    def get_num_active_filters(self):
        """Повертає кількість активних фільтрів"""
        return (self.mask > 0).sum().item()

    def get_effective_params(self):
        """Обчислює ефективну кількість параметрів"""
        total_params = sum(p.numel() for p in self.model.parameters())

        # Pruned in conv
        pruned_in_conv = len(self.pruned_filters) * (
            self.target_layer.weight.shape[1] *
            self.target_layer.weight.shape[2] *
            self.target_layer.weight.shape[3]
        )
        if self.target_layer.bias is not None:
            pruned_in_conv += len(self.pruned_filters)

        # Pruned in KAN
        kan_layer = self.model.kan_bottleneck.layers[0]
```

```
      params_per_input = (
         kan_layer.spline_weight.shape[0] *
         kan_layer.spline_weight.shape[2]
      )
      pruned_in_kan = len(self.pruned_filters) * params_per_input

      effective_params = total_params - pruned_in_conv - pruned_in_kan
      return effective_params, total_params
def train_epoch_with_mask(model, dataloader, criterion, optimizer,
mask_manager, device, desc="Training"):
   """Training з маскуванням градієнтів"""
   model.train()
   running_loss = 0.0
   correct = 0
   total = 0

   pbar = tqdm(dataloader, desc=desc, leave=False)
   for inputs, targets in pbar:
      inputs, targets = inputs.to(device), targets.to(device)

      optimizer.zero_grad()
      outputs = model(inputs)
      loss = criterion(outputs, targets)
      loss.backward()

      # Apply mask to gradients
      mask_manager.apply_mask_to_gradients()

      torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
      optimizer.step()

      running_loss += loss.item()
      _, predicted = outputs.max(1)
      total += targets.size(0)
      correct += predicted.eq(targets).sum().item()

      pbar.set_postfix({
         'loss': running_loss / (pbar.n + 1),
         'acc': 100. * correct / total
      })

   return running_loss / len(dataloader), 100. * correct / total

def train_model_with_mask(model, mask_manager, trainloader, testloader,
epochs, lr, device):
   """Training loop з підтримкою масок"""
   model = model.to(device)
   criterion = nn.CrossEntropyLoss()
   optimizer = optim.Adam(model.parameters(), lr=lr)
   scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

   history = {'train_loss': [], 'train_acc': [], 'test_loss': [], 'test_acc':
[]}
   best_acc = 0.0

   for epoch in range(epochs):
      train_loss, train_acc = train_epoch_with_mask(
         model, trainloader, criterion, optimizer, mask_manager, device,
         desc=f'Epoch {epoch+1}/{epochs} [Train]'
      )
```

```
      test_loss, test_acc = evaluate(
         model, testloader, criterion, device,
         desc=f'Epoch {epoch+1}/{epochs} [Test]'
      )

      scheduler.step()

      history['train_loss'].append(train_loss)
      history['train_acc'].append(train_acc)
      history['test_loss'].append(test_loss)
      history['test_acc'].append(test_acc)

      if test_acc > best_acc:
         best_acc = test_acc

      if (epoch + 1) % 5 == 0 or epoch == epochs - 1:
         effective_params, total_params = mask_manager.get_effective_params()
         print(f'Epoch {epoch+1}/{epochs}: '
             f'Train: {train_acc:.2f}% | Test: {test_acc:.2f}% | Best:
{best_acc:.2f}% | '
             f'Active:
{mask_manager.get_num_active_filters()}/{mask_manager.num_filters}')

   return model, history, best_acc
class KANGuidedPruner:
   """
   KAN-керований прунінг через маскування.
   Універсальний для всіх архітектур.
   """

   def __init__(self, model, config):
      self.model = model
      self.config = config
      self.interpreter = KANInterpreter(model)
      self.mask_manager = FilterMask(model)

      self.history = {
         'iteration': [],
         'num_active_filters': [],
         'accuracy': [],
         'total_params': [],
         'effective_params': [],
         'filters_removed': []
      }

   def identify_filters_to_prune(self, prune_percent):
      """Визначає фільтри для прунінгу на основі KAN"""
      importance_scores, _ =
self.interpreter.compute_filter_importance(method='max')

      active_filters = (self.mask_manager.mask > 0).cpu().numpy()
      active_indices = np.where(active_filters)[0]

      if len(active_indices) == 0:
         return [], active_indices

      active_importance = importance_scores[active_indices]
      num_to_prune = max(1, int(len(active_indices) * prune_percent))
```

```
        sorted_active_idx = np.argsort(active_importance)
        filters_to_prune = active_indices[sorted_active_idx[:num_to_prune]]

        print(f"\n🔍 Filter Analysis:")
        print(f"  Total: {len(importance_scores)}")
        print(f"  Active: {len(active_indices)}")
        print(f"  Pruning: {num_to_prune} ({prune_percent*100:.1f}%)")
        print(f"  Will remain: {len(active_indices) - num_to_prune}")

        return filters_to_prune, active_indices

    def prune_iteration(self, trainloader, testloader, iteration,
prune_percent):
        """Одна ітерація прунінгу"""
        print(f"\n{'='*70}")
        print(f"PRUNING ITERATION {iteration}")
        print(f"{'='*70}")

        filters_to_prune, _ = self.identify_filters_to_prune(prune_percent)

        if len(filters_to_prune) == 0:
            print("⚠️  No filters to prune.")
            return None, None

        self.mask_manager.prune_filters(filters_to_prune)

        print(f"\n🔧 Fine-tuning for {self.config.epochs_finetune} epochs...")
        self.model, _, best_acc = train_model_with_mask(
            self.model,
            self.mask_manager,
            trainloader,
            testloader,
            epochs=self.config.epochs_finetune,
            lr=self.config.lr_finetune,
            device=device
        )

        effective_params, total_params =
self.mask_manager.get_effective_params()
        num_active = self.mask_manager.get_num_active_filters()

        self.history['iteration'].append(iteration)
        self.history['num_active_filters'].append(num_active)
        self.history['accuracy'].append(best_acc)
        self.history['total_params'].append(total_params)
        self.history['effective_params'].append(effective_params)
        self.history['filters_removed'].append(len(filters_to_prune))

        compression = total_params / effective_params if effective_params > 0
else 0

        print(f"\n📊 Results:")
        print(f"  Accuracy: {best_acc:.2f}%")
        print(f"  Active: {num_active}/{self.mask_manager.num_filters}")
        print(f"  Effective params: {effective_params:,}")
        print(f"  Compression: {compression:.2f}×")

        return best_acc, effective_params
```

```
   def run_iterative_pruning(self, trainloader, testloader):
      """Повний цикл прунінгу"""
      criterion = nn.CrossEntropyLoss()
      _, initial_acc = evaluate(self.model, testloader, criterion, device,
"Initial")
      effective_params, total_params =
self.mask_manager.get_effective_params()

      print(f"\n{'='*70}")
      print(f"STARTING KAN-GUIDED PRUNING WITH MASKING")
      print(f"{'='*70}")
      print(f"Initial Accuracy: {initial_acc:.2f}%")
      print(f"Total Parameters: {total_params:,}")

      self.history['iteration'].append(0)
      self.history['num_active_filters'].append(self.mask_manager.num_filters)
      self.history['accuracy'].append(initial_acc)
      self.history['total_params'].append(total_params)
      self.history['effective_params'].append(effective_params)
      self.history['filters_removed'].append(0)

      for iteration in range(1, self.config.num_iterations + 1):
         acc, params = self.prune_iteration(
            trainloader, testloader, iteration,
self.config.prune_percent_per_iter
         )

         if acc is None:
            break

         if initial_acc - acc > self.config.max_accuracy_drop:
            print(f"\n⚠️  Accuracy drop too large. Stopping.")
            break

         num_active = self.mask_manager.get_num_active_filters()
         min_filters = max(10, int(self.mask_manager.num_filters * 0.1))
         if num_active < min_filters:
            print(f"\n⚠️  Too few filters. Stopping.")
            break

      return self.history

   def plot_results(self, save_path=None):
      """Візуалізація результатів"""
      fig, axes = plt.subplots(2, 2, figsize=(14, 10))

      iters = self.history['iteration']

      # Accuracy
      axes[0, 0].plot(iters, self.history['accuracy'], 'b-o', linewidth=2,
markersize=8)
      axes[0, 0].axhline(self.history['accuracy'][0], color='r', linestyle='--
', label='Initial')
      axes[0, 0].set_xlabel('Iteration')
      axes[0, 0].set_ylabel('Accuracy (%)')
      axes[0, 0].set_title('Accuracy During Pruning')
      axes[0, 0].legend()
      axes[0, 0].grid(True, alpha=0.3)
```

```
      # Parameters
      axes[0, 1].plot(iters, self.history['total_params'], 'g--', linewidth=2,
label='Total', alpha=0.5)
      axes[0, 1].plot(iters, self.history['effective_params'], 'g-o',
linewidth=2, label='Effective')
      axes[0, 1].set_xlabel('Iteration')
      axes[0, 1].set_ylabel('Parameters')
      axes[0, 1].set_title('Parameter Reduction')
      axes[0, 1].legend()
      axes[0, 1].grid(True, alpha=0.3)

      # Active filters
      axes[1, 0].plot(iters, self.history['num_active_filters'], 'm-o',
linewidth=2, markersize=8)
      axes[1, 0].set_xlabel('Iteration')
      axes[1, 0].set_ylabel('Active Filters')
      axes[1, 0].set_title('Filter Count')
      axes[1, 0].grid(True, alpha=0.3)

      # Compression vs Accuracy
      compression = [self.history['total_params'][0] / p if p > 0 else 0
                     for p in self.history['effective_params']]
      axes[1, 1].plot(compression, self.history['accuracy'], 'r-o',
linewidth=2, markersize=8)
      axes[1, 1].set_xlabel('Compression (×)')
      axes[1, 1].set_ylabel('Accuracy (%)')
      axes[1, 1].set_title('Accuracy vs Compression')
      axes[1, 1].grid(True, alpha=0.3)

      plt.tight_layout()
      if save_path:
         plt.savefig(save_path, dpi=300, bbox_inches='tight')
      plt.show()
def analyze_kan_interpretability_complete(model, device='cuda'):
   """
   КОМПЛЕКСНИЙ аналіз KAN: об'єднує обидва підходи.

   1. Простий аналіз важливості нейронів (твій підхід)
   2. Глибокий аналіз впливу та семантики (мій підхід)
   """
   model = model.to(device)
   model.eval()

   print(f"\n{'='*80}")
   print(f"🔬 ПОВНИЙ KAN АНАЛІЗ")
   print(f"{'='*80}\n")

   #
==========================================================================
   # ЧАСТИНА 1: БАЗОВИЙ АНАЛІЗ ВАЖЛИВОСТІ (твій підхід - простий і зрозумілий)
   #
==========================================================================

   print("📊 ЧАСТИНА 1: Важливість латентних нейронів для класів")
   print("-" * 80)

   # Отримуємо ваги класифікатора
   final_weights = model.classifier.weight.detach().cpu().numpy()
   latent_dim = final_weights.shape[1]
```

```
    print(f"Модель має {latent_dim} латентних нейронів у KAN bottleneck.\n")

    # Створюємо DataFrame
    importance_df = pd.DataFrame(
        final_weights,
        columns=[f'L{i}' for i in range(latent_dim)],
        index=classes
    )

    # Загальна важливість (середня абсолютна вага)
    overall_importance = np.mean(np.abs(final_weights), axis=0)
    importance_df.loc['ЗАГАЛЬНА_ВАЖЛИВІСТЬ'] = overall_importance

    # Сортуємо нейрони
    sorted_neurons =
importance_df.loc['ЗАГАЛЬНА_ВАЖЛИВІСТЬ'].sort_values(ascending=False)

    print("🏆 Рейтинг нейронів за загальною важливістю (топ-10):")
    for i, (neuron_name, importance) in
enumerate(sorted_neurons.head(10).items(), 1):
        print(f"  #{i}: {neuron_name} (важливість: {importance:.4f})")

    print("\n⚠️  Найменш важливі нейрони (кандидати на прунінг, топ-10):")
    for i, (neuron_name, importance) in
enumerate(sorted_neurons.tail(10).items(), 1):
        print(f"  #{i}: {neuron_name} (важливість: {importance:.4f})")

    # Візуалізація 1: Теплова карта важливості
    plt.figure(figsize=(20, 8))
    sns.heatmap(
        importance_df.drop('ЗАГАЛЬНА_ВАЖЛИВІСТЬ'),
        cmap='RdYlGn',  # Червоний-Жовтий-Зелений для підписаних ваг
        center=0,       # Центр на нулі
        annot=False,
        fmt='.2f',
        xticklabels=True,
        yticklabels=classes,
        cbar_kws={'label': 'Вага класифікатора'}
    )
    plt.title('Теплова карта важливості латентних нейронів для кожного класу\n'
              '(Позитивні ваги = нейрон підсилює клас, Негативні = пригнічує)',
              fontsize=14, fontweight='bold')
    plt.xlabel('Латентні нейрони', fontsize=12)
    plt.ylabel('Клас', fontsize=12)
    plt.tight_layout()
    plt.show()

    #
==========================================================================
    # ЧАСТИНА 2: СЕМАНТИЧНИЙ АНАЛІЗ (розширений)
    #
==========================================================================

    print(f"\n{'='*80}")
    print("🌿 ЧАСТИНА 2: Семантичний аналіз - Living vs Non-Living")
    print("-" * 80)

    living_classes = ['bird', 'cat', 'deer', 'dog', 'frog', 'horse']
```

```
    non_living_classes = ['plane', 'car', 'ship', 'truck']

    living_indices = [classes.index(c) for c in living_classes]
    non_living_indices = [classes.index(c) for c in non_living_classes]

    # Аналіз по нейронах
    semantic_analysis = []

    for neuron_idx in range(latent_dim):
        weights = final_weights[:, neuron_idx]

        # Середня активація для живих/неживих
        living_mean = np.abs(weights[living_indices]).mean()
        nonliving_mean = np.abs(weights[non_living_indices]).mean()

        # Preference score
        total = living_mean + nonliving_mean
        if total > 0:
            preference = (living_mean - nonliving_mean) / total
        else:
            preference = 0

        # Категорія
        if preference > 0.2:
            category = 'Living'
        elif preference < -0.2:
            category = 'Non-Living'
        else:
            category = 'Mixed'

        semantic_analysis.append({
            'neuron': f'L{neuron_idx}',
            'neuron_idx': neuron_idx,
            'living_activation': living_mean,
            'nonliving_activation': nonliving_mean,
            'preference': preference,
            'category': category,
            'overall_importance': overall_importance[neuron_idx]
        })

    semantic_df = pd.DataFrame(semantic_analysis)

    # Статистика по категоріях
    category_counts = semantic_df['category'].value_counts()
    print(f"\nРозподіл нейронів по категоріях:")
    print(f"  🐾 Living-focused:    {category_counts.get('Living', 0)} нейронів")
    print(f"  🚗 Non-Living-focused: {category_counts.get('Non-Living', 0)}
нейронів")
    print(f"  🔀 Mixed:          {category_counts.get('Mixed', 0)} нейронів")

    # Топ Living-focused
    living_neurons = semantic_df[semantic_df['category'] ==
'Living'].nlargest(5, 'preference')
    if not living_neurons.empty:
        print(f"\n🐾 Топ-5 нейронів для живих об'єктів:")
        for _, row in living_neurons.iterrows():
            print(f"  {row['neuron']}: preference={row['preference']:+.3f}, "
                  f"важливість={row['overall_importance']:.4f}")
        # Показуємо які саме живі класи
```

```
            neuron_weights = final_weights[:, row['neuron_idx']]
            top_classes = [(classes[i], neuron_weights[i]) for i in
living_indices]
            top_classes.sort(key=lambda x: abs(x[1]), reverse=True)
            print(f"      → Найбільший вплив на: {top_classes[0][0]}
({top_classes[0][1]:+.3f}), "
                  f"{top_classes[1][0]} ({top_classes[1][1]:+.3f})")

      # Топ Non-Living-focused
      nonliving_neurons = semantic_df[semantic_df['category'] == 'Non-
Living'].nsmallest(5, 'preference')
      if not nonliving_neurons.empty:
         print(f"\n🚗 Топ-5 нейронів для неживих об'єктів:")
         for _, row in nonliving_neurons.iterrows():
            print(f" {row['neuron']}: preference={row['preference']:+.3f}, "
                  f"важливість={row['overall_importance']:.4f}")
            # Показуємо які саме неживі класи
            neuron_weights = final_weights[:, row['neuron_idx']]
            top_classes = [(classes[i], neuron_weights[i]) for i in
non_living_indices]
            top_classes.sort(key=lambda x: abs(x[1]), reverse=True)
            print(f"      → Найбільший вплив на: {top_classes[0][0]}
({top_classes[0][1]:+.3f}), "
                  f"{top_classes[1][0]} ({top_classes[1][1]:+.3f})")

      # Візуалізація 2: Semantic preference
      fig, axes = plt.subplots(1, 2, figsize=(16, 6))

      # Scatter: living vs non-living activation
      scatter = axes[0].scatter(
         semantic_df['living_activation'],
         semantic_df['nonliving_activation'],
         c=semantic_df['preference'],
         s=semantic_df['overall_importance'] * 1000,  # Розмір = важливість
         cmap='RdYlGn',
         alpha=0.6,
         edgecolors='black',
         linewidth=0.5
      )
      axes[0].plot([0, semantic_df[['living_activation',
'nonliving_activation']].max().max()],
                   [0, semantic_df[['living_activation',
'nonliving_activation']].max().max()],
                   'k--', alpha=0.3, label='Однакова активація')
      axes[0].set_xlabel('Середня активація для Living класів', fontsize=11)
      axes[0].set_ylabel('Середня активація для Non-Living класів', fontsize=11)
      axes[0].set_title('Семантична спеціалізація нейронів\n(розмір = загальна важливість)')
      axes[0].legend()
      axes[0].grid(True, alpha=0.3)
      plt.colorbar(scatter, ax=axes[0], label='Preference (Living ↔ Non-
Living)')

      # Bar chart: preference distribution
      sorted_semantic = semantic_df.sort_values('preference', ascending=False)
      colors = ['green' if p > 0.2 else 'red' if p < -0.2 else 'gray'
                for p in sorted_semantic['preference']]

      axes[1].barh(range(len(sorted_semantic)), sorted_semantic['preference'],
color=colors, alpha=0.7)
      axes[1].axvline(0, color='black', linewidth=1)
```

```
   axes[1].axvline(0.2, color='green', linestyle='--', alpha=0.5,
linewidth=1.5)
   axes[1].axvline(-0.2, color='red', linestyle='--', alpha=0.5,
linewidth=1.5)
   axes[1].set_xlabel('Preference Score (← Non-Living | Living →)',
fontsize=11)
   axes[1].set_ylabel('Латентний нейрон (відсортовано)', fontsize=11)
   axes[1].set_title('Розподіл семантичних преференцій')
   axes[1].grid(True, alpha=0.3, axis='x')

   plt.tight_layout()
   plt.show()

   #
================================================================================
   # ЧАСТИНА 3: ДЕТАЛЬНИЙ АНАЛІЗ КЛЮЧОВИХ НЕЙРОНІВ
   #
================================================================================

   print(f"\n{'='*80}")
   print("🔍 ЧАСТИНА 3: Детальний аналіз обраних нейронів")
   print("-" * 80)

   # Візьмемо 2 найважливіші нейрони: один living, один non-living
   neurons_to_analyze = []

   if not living_neurons.empty:
      neurons_to_analyze.append(('Living',
int(living_neurons.iloc[0]['neuron_idx'])))

   if not nonliving_neurons.empty:
      neurons_to_analyze.append(('Non-Living',
int(nonliving_neurons.iloc[0]['neuron_idx'])))

   # Отримаємо KAN influence matrix
   interpreter = KANInterpreter(model)
   _, influence_matrix = interpreter.compute_filter_importance()

   for category, neuron_idx in neurons_to_analyze:
      print(f"\n{'—' * 60}")
      print(f"🎯 Нейрон L{neuron_idx} ({category}-focused)")
      print(f"{'—' * 60}")

      # Ваги для всіх класів
      neuron_weights = final_weights[:, neuron_idx]
      print(f"\nВплив на класи:")
      class_importance = [(classes[i], neuron_weights[i]) for i in
range(len(classes))]
      class_importance.sort(key=lambda x: abs(x[1]), reverse=True)

      for class_name, weight in class_importance:
         bar = '█' * int(abs(weight) * 50)
         sign = '+' if weight > 0 else '-'
         print(f" {class_name:8s}: {sign}{bar} {weight:+.4f}")

      # Топ CNN фільтри
      neuron_influences = influence_matrix[neuron_idx, :]
      top_filter_indices = np.argsort(neuron_influences)[-5:][::-1]

      print(f"\nТоп-5 CNN фільтрів, що впливають на цей нейрон:")
```

```
        for rank, filt_idx in enumerate(top_filter_indices, 1):
            print(f"  #{rank}: Filter {filt_idx:3d} (вплив:
{neuron_influences[filt_idx]:.6f})")

    print(f"\n{'='*80}")
    print(f"✅ АНАЛІЗ ЗАВЕРШЕНО")
    print(f"{'='*80}\n")

    return {
        'importance_df': importance_df,
        'sorted_neurons': sorted_neurons,
        'semantic_df': semantic_df,
        'living_neurons': living_neurons,
        'nonliving_neurons': nonliving_neurons
    }
def quick_train_for_demo(architecture='resnet18', epochs=20):
    """Швидке навчання для демо"""
    print(f"🚀 Quick training {architecture} ({epochs} epochs)...")

    model = CNNKANClassifier(
        architecture,
        num_classes=10,
        kan_latent_dim=config.kan_latent_dim,
        kan_grid_size=config.kan_grid_size,
        kan_spline_order=config.kan_spline_order,
        pretrained=False
    )

    print(model)

    model, history, best_acc = train_model(
        model, trainloader, testloader,
        epochs=epochs, lr=0.1, device=device,
        model_name=f"{architecture}_demo"
    )

    print(f"\n✅ Best accuracy: {best_acc:.2f}%")
    return model, history, best_acc
# Train demo model
print("Training demo model...")
demo_model, demo_history, demo_acc = quick_train_for_demo('resnet18',
epochs=40)
# Використовуй на своїй навченій моделі
results = analyze_kan_interpretability_complete(demo_model, device=device)

# Додатково: збережи результати для дисертації
results['importance_df'].to_csv(config.save_dir / 'neuron_importance.csv')
results['semantic_df'].to_csv(config.save_dir / 'semantic_analysis.csv')
print("Testing KAN-guided pruning with masking...")

pruner = KANGuidedPruner(demo_model, config)
pruning_history = pruner.run_iterative_pruning(trainloader, testloader)

pruner.plot_results(save_path=config.save_dir / 'demo_pruning_results.png')
class ExperimentRunner:
    """Менеджер експериментів - FIXED VERSION"""

    def __init__(self, config):
        self.config = config
        self.results = {}
```

```
   def run_single_experiment(self, architecture_name, trainloader,
testloader):
      print(f"\n{'#'*70}")
      print(f"# EXPERIMENT: {architecture_name.upper()}")
      print(f"{'#'*70}\n")

      # Step 1: Create & train model
      model = CNNKANClassifier(
         architecture_name,
         num_classes=10,
         kan_latent_dim=self.config.kan_latent_dim,
         kan_grid_size=self.config.kan_grid_size,
         kan_spline_order=self.config.kan_spline_order,
         pretrained=False
      )
      print(model)

      print(f"\n🎯 Training baseline...")
      model, train_history, best_acc = train_model(
         model, trainloader, testloader,
         epochs=self.config.epochs_pretrain,
         lr=self.config.lr_pretrain,
         device=device,
         model_name=f"{architecture_name}_baseline"
      )

      baseline_params = model.count_parameters()
      print(f"\n✅ Baseline: {best_acc:.2f}% | {baseline_params:,} params")

      # Step 2: KAN Analysis (SIMPLIFIED - without crashing spline viz)
      print(f"\n🔬 KAN Analysis...")

      try:
         interpreter = KANInterpreter(model)

         # Influence matrix
         influence_matrix = interpreter.visualize_influence_matrix(
            save_path=self.config.save_dir /
f"{architecture_name}_influence.png"
         )

         # Class-specific analysis
         class_idx = 5  # dog
         most_important_latent, top_filters =
interpreter.analyze_class_specific_features(
            class_idx, classes[class_idx]
         )

         # FIXED: Спрощена візуалізація (без interpolation що падає)
         # Просто показуємо важливість фільтрів
         print(f"\n📊 Top filters for L{most_important_latent}:")
         for i, filt_idx in enumerate(top_filters[:5], 1):
            print(f"  #{i}: Filter {filt_idx}")

         # Візуалізуємо influence для цього нейрону
         fig, ax = plt.subplots(1, 1, figsize=(12, 4))
         neuron_influences = influence_matrix[most_important_latent, :]
         ax.bar(range(len(neuron_influences)), neuron_influences, alpha=0.7,
```

```
color='steelblue')
            ax.set_xlabel('CNN Filter Index')
            ax.set_ylabel('Influence Magnitude')
            ax.set_title(f'Filter Influences for Latent Neuron
L{most_important_latent} (class: {classes[class_idx]})')

            # Mark top filters
            for filt_idx in top_filters[:5]:
               ax.axvline(filt_idx, color='red', linestyle='--', alpha=0.5,
linewidth=1)
               ax.text(filt_idx, neuron_influences[filt_idx], f'{filt_idx}',
                    ha='center', va='bottom', fontsize=8, color='red')

            plt.tight_layout()
            plt.savefig(self.config.save_dir /
f"{architecture_name}_filter_influences.png",
                    dpi=300, bbox_inches='tight')
            plt.show()

         except Exception as e:
            print(f"⚠️ Warning in KAN analysis: {str(e)}")
            print("Continuing without detailed KAN analysis...")
            influence_matrix = None

         # Step 3: Pruning
         print(f"\n🔪 Pruning...")

         try:
            pruner = KANGuidedPruner(model, self.config)
            pruning_history = pruner.run_iterative_pruning(trainloader,
testloader)

            pruner.plot_results(
               save_path=self.config.save_dir /
f"{architecture_name}_pruning.png"
            )
         except Exception as e:
            print(f"⚠️ Error in pruning: {str(e)}")
            pruning_history = None

         # Save results
         self.results[architecture_name] = {
            'baseline_accuracy': best_acc,
            'baseline_params': baseline_params,
            'train_history': train_history,
            'pruning_history': pruning_history,
            'model': model
         }

         return self.results[architecture_name]

      def run_multiple_experiments(self, architectures, trainloader, testloader):
         """Run experiments for multiple architectures"""
         for arch_name in architectures:
            try:
               self.run_single_experiment(arch_name, trainloader, testloader)
               print(f"\n✅ Completed: {arch_name}")
            except Exception as e:
               print(f"\n❌ Error in {arch_name}: {str(e)}")
```

```
            import traceback
            traceback.print_exc()
            print(f"Skipping {arch_name} and continuing...")
            continue

   def compare_results(self):
      """Compare all experiment results"""
      if not self.results:
         print("No results yet!")
         return None

      data = []
      for arch, res in self.results.items():
         row = {
            'Architecture': arch,
            'Baseline Acc': res['baseline_accuracy'],
            'Baseline Params': res['baseline_params'],
         }

         # Add pruning results if available
         if res['pruning_history'] is not None and
len(res['pruning_history']['accuracy']) > 0:
            row['Final Acc'] = res['pruning_history']['accuracy'][-1]
            row['Final Params'] = res['pruning_history']['effective_params'][-
1]
            row['Compression'] = res['baseline_params'] /
res['pruning_history']['effective_params'][-1]
            row['Acc Drop'] = res['baseline_accuracy'] -
res['pruning_history']['accuracy'][-1]
         else:
            row['Final Acc'] = None
            row['Final Params'] = None
            row['Compression'] = None
            row['Acc Drop'] = None

         data.append(row)

      df = pd.DataFrame(data)

      print("\n" + "="*80)
      print("COMPARISON OF ALL EXPERIMENTS")
      print("="*80)
      print(df.to_string(index=False))

      df.to_csv(self.config.save_dir / 'comparison.csv', index=False)

      # Visualization (only for experiments with pruning results)
      df_with_pruning = df.dropna(subset=['Compression'])

      if not df_with_pruning.empty:
         fig, axes = plt.subplots(1, 2, figsize=(14, 5))

         # Compression vs Accuracy
         axes[0].scatter(df_with_pruning['Compression'], df_with_pruning['Acc
Drop'],
                    s=100, alpha=0.6)
         for _, row in df_with_pruning.iterrows():
            axes[0].annotate(row['Architecture'],
                       (row['Compression'], row['Acc Drop']), fontsize=8)
         axes[0].set_xlabel('Compression (×)')
```

```
        axes[0].set_ylabel('Accuracy Drop (%)')
        axes[0].set_title('Compression vs Accuracy')
        axes[0].grid(True, alpha=0.3)

        # Final accuracy comparison
        axes[1].barh(df_with_pruning['Architecture'], df_with_pruning['Final
Acc'], alpha=0.7)
        axes[1].set_xlabel('Accuracy (%)')
        axes[1].set_title('Final Accuracy After Pruning')
        axes[1].grid(True, alpha=0.3, axis='x')

        plt.tight_layout()
        plt.savefig(self.config.save_dir / 'comparison.png', dpi=300)
        plt.show()

      return df
# Initialize runner
runner = ExperimentRunner(config)

# Select architectures (почни з малих для тесту)
# test_architectures = [
#    'resnet18',
#    'mobilenet_v2',
#    'squeezenet1_1',
# ]

# For full thesis, use all:
test_architectures = AVAILABLE_ARCHITECTURES

print("Running experiments on:", test_architectures)

# Run
runner.run_multiple_experiments(test_architectures, trainloader, testloader)

# Compare
comparison_df = runner.compare_results()

def print_methods_summary(df_all_methods):
   """Підсумкова статистика"""

   print("\n" + "="*100)
   print("ПІДСУМКОВА СТАТИСТИКА ПО МЕТОДАХ")
   print("="*100)

   for variant_group in [['Moderate', 'Small (50%)'], ['Aggressive', 'Tiny
(25%)']]:
      variant_name = 'MODERATE LEVEL' if 'Moderate' in variant_group else
'AGGRESSIVE LEVEL'
      print(f"\n{'='*100}")
      print(f"{variant_name}")
      print(f"{'='*100}")

      variant_data =
df_all_methods[df_all_methods['Variant'].isin(variant_group)]

      for method in df_all_methods['Method'].unique():
         method_data = variant_data[variant_data['Method'] == method]

         if len(method_data) == 0:
            continue
```

```
        avg_comp = method_data['Compression (×)'].mean()
        avg_acc_drop = method_data['Acc Drop (%)'].mean()
        avg_reduction = method_data['Reduction (%)'].mean()
        efficiency = avg_comp / (avg_acc_drop + 0.1)

        variant_display = method_data['Variant'].iloc[0]

        print(f"\n📊 {method} ({variant_display}):")
        print(f"  Average Compression:  {avg_comp:.2f}× (range:
{method_data['Compression (×)'].min():.2f}× - {method_data['Compression
(×)'].max():.2f}×)")
        print(f"  Average Acc Drop:     {avg_acc_drop:.2f}% (range:
{method_data['Acc Drop (%)'].min():.2f}% - {method_data['Acc Drop
(%)'].max():.2f}%)")
        print(f"  Average Reduction:    {avg_reduction:.1f}%")
        print(f"  Efficiency Score:     {efficiency:.2f}")
def create_individual_method_comparison_plots(df_all_methods):
    """
    Створює ОКРЕМІ великі графіки для порівняння методів
    """

    plt.rcParams['font.family'] = 'DejaVu Sans'
    plt.rcParams['font.size'] = 12

    architectures = df_all_methods['Architecture'].unique()

    colors = {
        'KAN-Guided': '#3498db',
        'Magnitude Pruning': '#e74c3c',
        'Knowledge Distillation': '#2ecc71'
    }

    # === FIGURE 1: Compression Comparison (All Variants) ===
    fig1, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 12))

    # Moderate variants
    moderate_variants = ['Moderate', 'Small (50%)']
    moderate_data =
df_all_methods[df_all_methods['Variant'].isin(moderate_variants)]

    x = np.arange(len(architectures))
    width = 0.28

    for i, method in enumerate(['KAN-Guided', 'Magnitude Pruning', 'Knowledge
Distillation']):
        method_data = moderate_data[moderate_data['Method'] ==
method].sort_values('Architecture')
        offset = (i - 1) * width
        bars = ax1.bar(x + offset, method_data['Compression (×)'], width,
                       label=method, color=colors[method], alpha=0.85,
                       edgecolor='black', linewidth=1.5)

        for bar in bars:
            height = bar.get_height()
            ax1.text(bar.get_x() + bar.get_width()/2., height + 0.05,
                     f'{height:.2f}×', ha='center', va='bottom', fontsize=10,
fontweight='bold')

    ax1.set_xlabel('Architecture', fontsize=14, fontweight='bold')
    ax1.set_ylabel('Compression Ratio (×)', fontsize=14, fontweight='bold')
```

```
   ax1.set_title('Moderate Compression: KAN-Guided (Moderate) vs Magnitude
(Moderate) vs KD (Small 50%)',
             fontsize=15, fontweight='bold', pad=15)
   ax1.set_xticks(x)
   ax1.set_xticklabels(architectures, rotation=35, ha='right', fontsize=11)
   ax1.legend(fontsize=12, loc='upper left', framealpha=0.95)
   ax1.grid(True, alpha=0.3, axis='y', linestyle='--')
   ax1.set_ylim(0, moderate_data['Compression (×)'].max() * 1.2)

   # Aggressive variants
   aggressive_variants = ['Aggressive', 'Tiny (25%)']
   aggressive_data =
df_all_methods[df_all_methods['Variant'].isin(aggressive_variants)]

   for i, method in enumerate(['KAN-Guided', 'Magnitude Pruning', 'Knowledge
Distillation']):
      method_data = aggressive_data[aggressive_data['Method'] ==
method].sort_values('Architecture')
      offset = (i - 1) * width
      bars = ax2.bar(x + offset, method_data['Compression (×)'], width,
                label=method, color=colors[method], alpha=0.85,
                edgecolor='black', linewidth=1.5)

      for bar in bars:
         height = bar.get_height()
         ax2.text(bar.get_x() + bar.get_width()/2., height + 0.08,
               f'{height:.2f}×', ha='center', va='bottom', fontsize=10,
fontweight='bold')

   ax2.set_xlabel('Architecture', fontsize=14, fontweight='bold')
   ax2.set_ylabel('Compression Ratio (×)', fontsize=14, fontweight='bold')
   ax2.set_title('Aggressive Compression: KAN-Guided (Aggressive) vs Magnitude
(Aggressive) vs KD (Tiny 25%)',
             fontsize=15, fontweight='bold', pad=15)
   ax2.set_xticks(x)
```