МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ХРИПЛИВИЙ ІВАН ІГОРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2025 р.

РЕДУКЦІЯ ПАРАМЕТРІВ НЕЙРОННИХ МЕРЕЖ НА ПІДСТАВІ МАШИННОГО НАВЧАННЯ

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (магістерська) робота

Науковий керівник:

Олександр РОТШТЕЙН, професор кафедри
інформаційних технологій,
д. т. н., професор

_____
(підпис)

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2025

АНОТАЦІЯ

Хрипливий І. І. Редукція параметрів нейронних мереж на підставі машинного навчання. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У кваліфікаційній роботі досліджено методи оптимізації використання пам'яті у великих мовних моделях (LLM) та розроблено підходи до редукції параметрів нейронних мереж. Проаналізовано сучасні архітектури Mixture-of-Experts (MoE), методи квантизації, оптимізації KV-кеша та факторизації ембедингів.

Запропоновано метод shared MoE layer, який дозволяє використовувати один набір експертів на всіх шарах моделі замість окремих FFN блоків. Експериментально підтверджено, що модель з 50M параметрів досягає якості щільної моделі з 125M параметрів.

Розроблено методологію конвертації існуючих MoE моделей (Qwen 3 30B-A3B) шляхом об'єднання експертів у глобальний пул з наступним застосуванням REAP pruning для видалення надлишкових експертів. Очікується редукція 80-90% параметрів при збереженні якості.

Ключові слова: великі мовні моделі, Mixture-of-Experts, редукція параметрів, shared experts, REAP pruning, KV-кеш, квантизація.

69 с., 12 табл., 7 рис., 50 джерел.

ABSTRACT

Khryplyvyi I. I. Reduction of Neural Network Parameters Based on Machine Learning. Specialty 122 "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2025.

The qualification work investigates methods for optimizing memory usage in Large Language Models (LLM) and develops approaches to neural network parameter reduction. Modern Mixture-of-Experts (MoE) architectures, quantization methods, KV-cache optimization, and embedding factorization are analyzed.

A shared MoE layer method is proposed, which allows using a single set of experts across all model layers instead of separate FFN blocks. It is experimentally confirmed that a 50M parameter model achieves the quality of a 125M щільна модель.

A methodology for converting existing MoE models (Qwen 3 30B-A3B) by combining experts into a global pool with subsequent application of REAP pruning for removing redundant experts is developed. A reduction of 80-90% of parameters while maintaining quality is expected.

Keywords: large language models, Mixture-of-Experts, parameter reduction, shared experts, REAP pruning, KV-cache, quantization.

69 p., 12 tables, 7 fig., 50 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

LLM (Large Language Model) — велика мовна модель

MoE (Mixture-of-Experts) — архітектура суміші експертів

FFN (Feed-Forward Network) — повнозв'язна нейронна мережа

KV-кеш (Key-Value cache) — кеш ключів та значень механізму уваги

MHA (Multi-Head Attention) — багатоголова увага

MQA (Multi-Query Attention) — багатозапитова увага

GQA (Grouped-Query Attention) — групова увага

MLA (Multi-Head Latent Attention) — багатоголова латентна увага

REAP (Routing-based Expert Assignment Pruning) — метод прунінгу на основі маршрутизації

BI (Block Influence) — метрика впливу блоку на якість моделі

RoPE (Rotary Position Embedding) — ротаційне позиційне кодування

GPTQ — метод квантизації ваг нейронних мереж

FP8/FP16/FP32 — формати чисел з плаваючою комою (8/16/32 біти)

INT4/INT8 — цілочисельні формати (4/8 біт)

VRAM — відеопам'ять графічного прискорювача

GPU (Graphics Processing Unit) — графічний процесор

TPU (Tensor Processing Unit) — тензорний процесор Google

ВСТУП

Актуальність теми дослідження. Великі мовні моделі (Large Language Models, LLM) демонструють широкі можливості в обробці природної мови, генерації тексту, програмуванні та міркуванні. Проте їх розгортання обмежується значними вимогами до пам'яті: найпотужніші моделі (GPT-4, DeepSeek V3, Kimi K2) потребують серверної інфраструктури вартістю мільйони доларів.

Розрив між можливостями споживчого та серверного обладнання сягає трьох порядків. NVIDIA RTX 5090 з 32 ГБ пам'яті коштує $2000, тоді як серверна стійка GB200 NVL72 з 30+ ТБ пам'яті — близько $3 мільйонів. Це означає, що найпотужніші відкриті моделі недоступні для локального запуску індивідуальними дослідниками та розробниками.

Архітектура Mixture-of-Experts (MoE) частково вирішує цю проблему, активуючи лише частину параметрів на кожен токен. Проте всі параметри потрібно тримати в пам'яті — модель DeepSeek V3 з 671B параметрів займає понад 670 ГБ у FP8 навіть при 37B активних. Квантизація дозволяє стиснути ваги до 4-8 біт, але не змінює принципово вимоги до обладнання.

Мета і завдання дослідження. Метою роботи є розробка методів редукції параметрів нейронних мереж для наближення передових моделей до можливостей споживчого обладнання. Для досягнення мети поставлено наступні завдання: проаналізувати існуючі методи оптимізації пам'яті LLM; дослідити архітектури Mixture-of-Experts та механізми перевикористання експертів; розробити метод shared MoE layer для редукції параметрів; експериментально валідувати підхід на малому масштабі; застосувати метод до великої MoE моделі (Qwen 3 30B-A3B).

Об'єкт дослідження — процеси оптимізації використання пам'яті у великих мовних моделях.

Предмет дослідження — методи редукції параметрів архітектур Mixture-of-Experts через перевикористання експертів між шарами.

Методи дослідження включають: аналіз та синтез для вивчення існуючих підходів; експериментальне дослідження для валідації гіпотез; порівняльний аналіз для оцінки ефективності методів.

Наукова новизна одержаних результатів полягає в: обґрунтуванні можливості використання shared MoE layer для заміни окремих FFN блоків на всіх шарах моделі; розробці методології конвертації існуючих MoE моделей з локальними пулами експертів у архітектуру з глобальним пулом; застосуванні REAP pruning для виявлення та видалення надлишкових експертів при глобальній конкуренції.

Практичне значення одержаних результатів полягає в можливості значного зменшення вимог до пам'яті для розгортання великих MoE моделей, що робить їх доступними для ширшого кола користувачів та дослідників.

РОЗДІЛ 1

МОВНІ МОДЕЛІ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1. Принцип роботи великих мовних моделей

Велика мовна модель (Large Language Model, LLM) — це нейронна мережа, натренована передбачати наступне слово в тексті. Коли ви пишете «Київ — столиця...», модель оцінює ймовірність кожного можливого продовження і вибирає найімовірніше: «України». Потім додає це слово до контексту і повторює процес для наступного слова. Токен за токеном народжується текст.

Ця проста задача — передбачення наступного токена — виявилася ефективною. Щоб добре передбачати текст, модель змушена засвоїти граматику, факти про світ, логічні зв'язки, стиль письма, навіть елементи міркування. Ніхто не програмував ChatGPT відповідати на запитання чи писати код — ці здібності виникли як побічний ефект тренування на терабайтах тексту з інтернету, книг, статей, репозиторіїв коду.

Сучасні LLM базуються на архітектурі Transformer. Її ключовий компонент — механізм уваги (attention): модель аналізує всі слова в контексті одночасно і вчиться, які з них важливі для передбачення наступного. Це дозволяє враховувати зв'язки на великих відстанях — між початком і кінцем довгого документа.

«Великими» ці моделі називають через кількість параметрів — числових ваг, що визначають поведінку мережі. GPT-2 (2019) мав 1.5 мільярда параметрів. GPT-3 (2020) — 175 мільярдів. Сучасні моделі сягають трильйонів. Кожен параметр — це число, яке потрібно зберігати в пам'яті та використовувати при обчисленнях.

1.2. Масштабування моделей

Історія розвитку LLM — це історія зростання. BERT (2018) мав 340 мільйонів параметрів. GPT-2 (2019) — 1.5 мільярда. GPT-3 (2020) — 175

мільярдів. PaLM (2022) — 540 мільярдів. GPT-4 (2023) — 1.8 трильйона. Grok 4 (2025) — 3 трильйони. Grok 5 анонсовано на 6 трильйонів.

Ранні закони масштабування (Kaplan et al., 2020) стверджували, що продуктивність моделі залежить насамперед від кількості параметрів. Це спровокувало гонку: компанії нарощували розмір моделей, тримаючи обсяг тренувальних даних відносно сталим. GPT-3 тренували на 300 мільярдах токенів — приблизно 1.7 токена на параметр.

У 2022 році DeepMind опублікували дослідження Chinchilla (Hoffmann et al.). Автори натренували понад 400 моделей різного розміру і виявили: GPT-3 та подібні моделі були суттєво недотреновані. Оптимальне співвідношення — близько 20 токенів на параметр. Модель Chinchilla (70B параметрів, 1.4 трильйона токенів) перевершила 280B Gopher, 175B GPT-3 та 530B Megatron-Turing NLG на більшості бенчмарків, використовуючи той самий обчислювальний бюджет.

Висновок Chinchilla: замість нарощування параметрів вигідніше тренувати менші моделі на більшій кількості даних. LLaMA 3 70B (2024) тренували на 15 трильйонах токенів — 200 токенів на параметр, у 10 разів більше за Chinchilla-оптимум. Такі моделі потребують більше compute для тренування, але значно менше пам'яті для інференсу.

Є фізична межа того, скільки знань можна «впакувати» в параметри моделі. Allen-Zhu та Li у серії робіт «Physics of Language Models» (2024) експериментально встановили: трансформер здатний зберігати максимум 2 біти знань на параметр, навіть при квантизації до int8. Це означає, що 7B-модель може зберігати близько 14 мільярдів біт фактичних знань — більше, ніж англійська Вікіпедія разом з підручниками, але все одно скінченний обсяг.

Це накладає обмеження на «перетренування» малих моделей. Qwen 3 0.6B (2025) — приклад моделі, що наближається до цієї межі: 600 мільйонів параметрів, натренованих на 36 трильйонах токенів — близько 60,000 токенів на параметр. Модель перевершує попередні моделі з 3B параметрів на

багатьох бенчмарках. Але подальше збільшення тренувальних даних вже не покращить якість суттєво — досягнуто межі ємності.

1.3. Архітектура Mixture-of-Experts

Архітектурна відповідь на обмеження щільних моделей — Mixture-of-Experts (MoE). У щільної моделі кожен токен проходить через усі параметри мережі. У MoE-моделі є набір «експертів» (окремих FFN-блоків), і для кожного токена маршрутизатор вибирає лише кілька з них.

DeepSeek V3 (грудень 2024) популяризував цей підхід для відкритих моделей: 671 мільярд загальних параметрів, 37 мільярдів активних на токен, співвідношення 18:1. Модель використовує Multi-Head Latent Attention (MLA) для стиснення KV-кеша та тренування у форматі FP8.

Kimi K2 (липень 2025) пішов далі: 1 трильйон загальних параметрів, 32 мільярди активних, 384 експерти з яких 8 активуються на кожен токен. Модель тренували на 15.5 трильйонах токенів з використанням оптимізатора MuonClip. На бенчмарках Kimi K2 перевершує DeepSeek V3, LLaMA 3.1 405B та GPT-4.1 на задачах кодування (SWE-bench Verified: 65.8%) та математики (MATH-500: 97.4%).

Таблиця 1.1 – Порівняння архітектур LLM

Модель	Загальні параметри	Активні параметри	Тренувальні токени
GPT-3 (2020)	175B	175B (dense)	300B
LLaMA 3 70B (2024)	70B	70B (dense)	15T
DeepSeek V3 (2024)	671B	37B	14.8T
Kimi K2 (2025)	1T	32B	15.5T

MoE вирішує проблему ємності знань: 1T параметрів дають значно більший «резервуар» для зберігання інформації, ніж 70B. Але всі експерти потрібно тримати в пам'яті, навіть якщо активні лише деякі. Kimi K2 у нативному INT4 потребує близько 600 ГБ RAM, у FP8 — понад 1 ТБ.

1.4. Внутрішня будова Transformer

Для розуміння методів оптимізації пам'яті необхідно розглянути внутрішню будову архітектури Transformer, на якій базуються всі сучасні мовні моделі.

Блок Transformer

Transformer складається з послідовності однакових блоків. Кожен блок містить два основні компоненти: механізм уваги (Multi-Head Attention) та повнозв'язну мережу (Feed-Forward Network). Між ними розташовані шари нормалізації та залишкові з'єднання (residual connections).

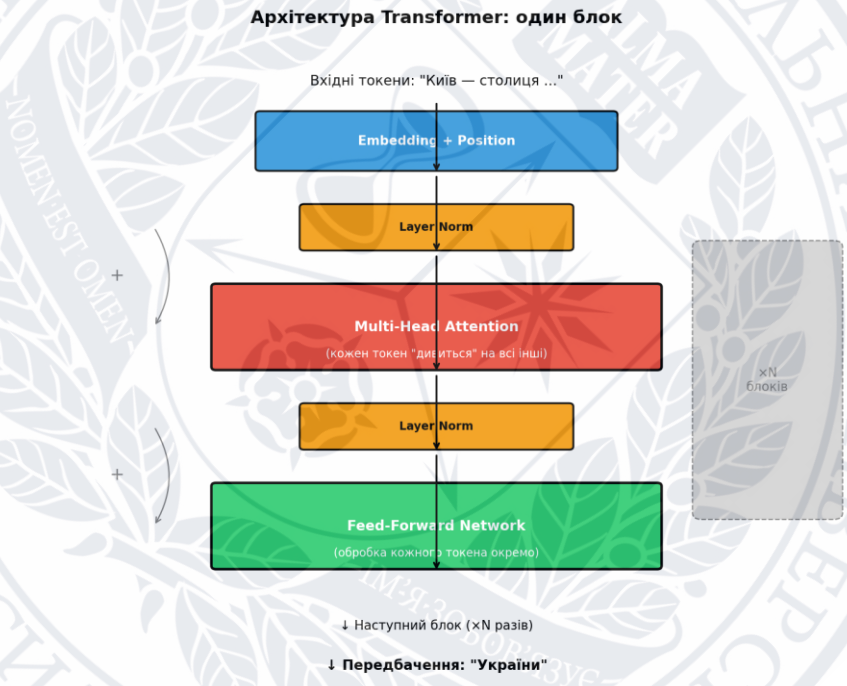

Рисунок 1.1 – Архітектура одного блоку Transformer

Вхідний текст спочатку перетворюється на послідовність векторів через таблицю ембедингів. До кожного вектора додається позиційне кодування, яке дозволяє моделі розрізняти порядок токенів. Потім послідовність проходить через N блоків Transformer (типово 32-80 для великих моделей), після чого фінальний прихований стан проєктується на словник для передбачення наступного токена.

Механізм уваги

Механізм уваги — основна особливість Transformer [1]. Він дозволяє кожному токену «бачити» всі інші токени в послідовності та визначати, які з них найважливіші для поточного передбачення.

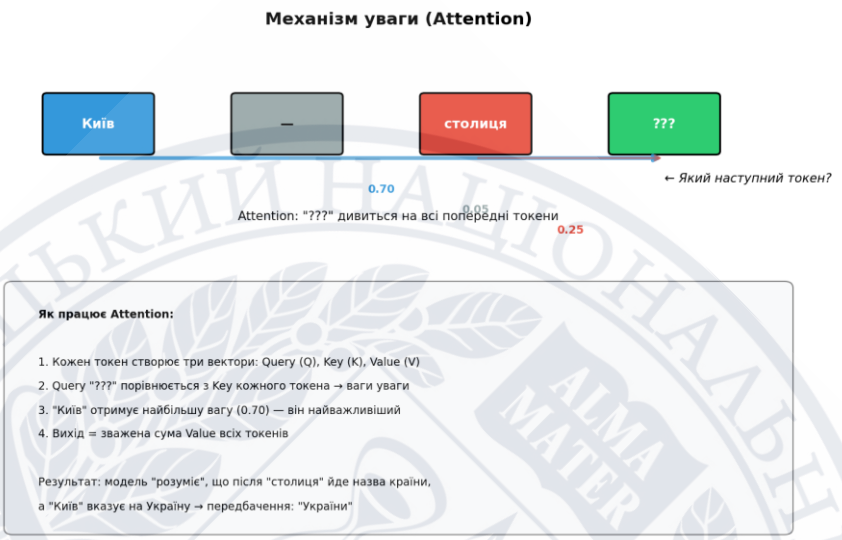

Рисунок 1.2 – Візуалізація механізму уваги

Для обчислення уваги кожен токен генерує три вектори: Query (запит), Key (ключ) та Value (значення). Query поточного токена порівнюється з Key всіх токенів через скалярний добуток, результат нормалізується через softmax — отримуємо ваги уваги. Фінальний вихід — зважена сума Value всіх токенів.

KV-кеш та генерація

При генерації тексту модель передбачає по одному токену за раз. Щоб не перераховувати Key та Value для всіх попередніх токенів на кожному кроці, їх зберігають у KV-кеші.

Рисунок 1.3 – Зростання KV-кеша при генерації

KV-кеш зростає лінійно з довжиною контексту. Для моделей з довгим контекстом (128K+ токенів) це може становити десятки гігабайт пам'яті — порівнянно з розміром самих ваг моделі. Оптимізація KV-кеша є одним із ключових напрямків зменшення споживання пам'яті.

1.5. Обчислювальні характеристики інференсу

Вимоги до пам'яті для інференсу можна обчислити з перших принципів. Кожен параметр у форматі fp16 займає 2 байти. Модель з P параметрами потребує 2P байтів для зберігання ваг. 7B-модель у fp16 займає 14 ГБ, 70B — 140 ГБ. При 4-бітній квантизації кожен параметр займає 0.5 байта: 7B-модель стискається до 3.5 ГБ, 70B — до 35 ГБ.

Але це лише ваги. Під час генерації модель зберігає KV-кеш — проміжні значення attention для кожного токена в контексті. На кожен токен потрібно:

KV-кеш на токен = 2 × 2 × n_layers × d_model байтів

Для LLaMA 3 70B (80 шарів, d_model = 8192) це близько 2.6 МБ на токен. При контексті 8192 токени — 21 ГБ додатково до ваг моделі.

Швидкість генерації обмежена пропускною здатністю пам'яті, а не обчислювальною потужністю. Для генерації одного токена потрібно прочитати всі ваги моделі з пам'яті. Затримка для малих батчів:

Затримка = 2P / пропускна здатність пам'яті

RTX 5090 має bandwidth близько 1.8 ТБ/с. Для 14B-моделі у fp16: 28 ГБ / 1.8 ТБ/с ≈ 15 мс на токен, або ~65 токенів/с. Для 32B у 4-bit: 16 ГБ / 1.8 ТБ/с ≈ 9 мс на токен, або ~110 токенів/с.

1.6. Розрив у доступності обладнання

Найпотужніша споживча відеокарта — NVIDIA RTX 5090 (січень 2025): 32 ГБ GDDR7, $2000. На ній запускаються: 14B dense-модель у fp16 (~28 ГБ), 32B у 4-bit квантизації (~16 ГБ + KV-кеш), або Qwen 3 30B-A3B MoE. 70B у 4-bit потребує 35-40 ГБ — вже не вміщується.

NVIDIA GB200 NVL72 — стійка вагою 1.36 тонни з 72 GPU Blackwell, з'єднаними через NVLink: понад 30 ТБ пам'яті, 120 кВт споживання, близько $3 мільйонів.

Таблиця 1.2 – Порівняння споживчого та серверного обладнання

Характеристика	**RTX 5090**	**GB200 NVL72**
Пам'ять	32 ГБ	30+ ТБ
Ціна	$2,000	~$3,000,000
Енергоспоживання	575 Вт	120,000 Вт

Різниця в ~1000 разів за пам'яттю. Моделі рівня Kimi K2 чи DeepSeek V3 не запускаються на обладнанні, доступному індивідуальним дослідникам.

1.7. Висновки до розділу

Аналіз еволюції мовних моделей виявляє фундаментальне протиріччя. З одного боку, продуктивність моделей зростає з кількістю параметрів та обсягом тренувальних даних. З іншого — існують жорсткі фізичні обмеження: межа щільності знань (2 біти на параметр), вимоги до пам'яті для зберігання ваг та KV-кеша, залежність швидкості генерації від пропускна здатність пам'яті.

Індустрія відповіла на ці обмеження архітектурою MoE, яка дозволяє збільшити ємність знань без пропорційного зростання обчислювальних витрат на токен. Але MoE не вирішує проблему пам'яті: всі експерти потрібно тримати завантаженими.

Розрив між можливостями споживчого та серверного обладнання сягає трьох порядків. Це означає, що найпотужніші відкриті моделі (DeepSeek V3, Kimi K2) недоступні для локального запуску на обладнанні індивідуальних дослідників та розробників.

Звідси мотивація цієї роботи: розробити методи зменшення кількості параметрів, необхідних для інференсу, без суттєвої втрати якості — щоб наблизити frontier-моделі до можливостей споживчого обладнання.

РОЗДІЛ 2

ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ ПАМ'ЯТІ У ВЕЛИКИХ МОВНИХ МОДЕЛЯХ

2.1. Проблема використання пам'яті у великих мовних моделях

Основною мотивацією для дослідження та оптимізації використання пам'яті в LLM є зменшення споживання пам'яті під час інференсу. Зі збільшенням розміру моделей та обсягу даних, що обробляються під час інференсу, вимоги до пам'яті стають все більш критичними. Моделі з трильйонами параметрів, такі як GPT-4, вимагають величезних обсягів пам'яті для зберігання ваг і активацій під час інференсу.

Зменшення споживання пам'яті дозволяє розгортати більші й потужніші моделі на наявному апаратному забезпеченні, що робить їх більш доступними та практичними для реального використання. Крім того, оптимізація використання пам'яті може призвести до більш швидкого інференсу, оскільки зменшується кількість обмінів даними між пам'яттю та процесором.

Розуміння того, як LLM використовують пам'ять під час інференсу, є важливою темою з кількох причин. По-перше, ефективне використання пам'яті є ключовим фактором продуктивності та масштабованості LLM під час розгортання та використання моделей. Оскільки розмір моделей продовжує зростати, оптимізація використання пам'яті стає все більш важливою для забезпечення можливості розгортання цих моделей на реальних системах.

По-друге, розуміння того, як LLM зберігають та використовують інформацію під час інференсу, може дати цінні знання про їх можливості та обмеження. Це може допомогти дослідникам і практикам розробляти більш ефективні та дієві моделі, а також вирішувати потенційні проблеми, такі як затримки чи неточності в генерованому тексті.

По-третє, поглиблене вивчення використання пам'яті в LLM під час інференсу може привести до нових відкриттів і інновацій у галузі штучного

інтелекту та обробки природної мови, таких як розробка більш ефективних архітектур або методів оптимізації моделей для розгортання.

Типи використання пам'яті LLM

Активації (виходи) нейронів зберігаються на кожному шарі мережі під час обробки вхідних даних. Ці активації представляють собою числові значення, які відображають ступінь активності нейронів у відповідь на вхідні дані та містять інформацію про контекст і семантичні зв'язки між токенами в послідовності. Активації передаються між шарами мережі та використовуються для обчислення наступних активацій і, зрештою, для генерації наступних токенів у послідовності.

Активації зберігаються в оперативній пам'яті (RAM) під час прямого проходу (forward pass) через мережу і можуть бути використані під час зворотного поширення (backpropagation) для оновлення ваг моделі. Хоча активації відіграють важливу роль у роботі LLM, вони зазвичай не займають значну частину пам'яті порівняно з вагами моделі.

KV-Cache. Під час генерації тексту LLM враховує контекст попередніх токенів (слів або частин слів), зберігаючи в пам'яті обмежену кількість попередніх токенів і використовуючи їх для прогнозування наступного токена. Ця контекстна пам'ять дозволяє моделі генерувати послідовний і зв'язний текст, враховуючи попередній контекст.

У сучасних LLM розмір контекстної пам'яті може сягати кількох сотень тисяч або навіть мільйонів токенів, що дозволяє моделі враховувати значно більший контекст порівняно з попередніми моделями. Наприклад, Gemini 3 Pro, випущена Google у листопаді 2025 року, надає комерційний доступ до контексту в 1 мільйон токенів, що дозволяє аналізувати приблизно 1500 сторінок тексту, 50 000 рядків коду або понад 200 подкаст-епізодів одночасно. Але варто враховувати, що розмір KV-Cache зростає лінійно з довжиною контексту та може сягати сотень гігабайт для довгих контекстів.

Ваги (Weights). LLM має мільярди або навіть трильйони внутрішніх ваг (параметрів), які налаштовуються під час тренування на великих обсягах текстових даних. Ваги представляють собою числові значення, які визначають силу зв'язків між нейронами в мережі. Наприклад, у моделі GPT-4 з 1.8 трильйонами параметрів, кожен параметр є вагою, яка зберігається у вигляді 16-бітного числа з рухомою комою (FP16), що означає, що модель займає близько 3.6 петабайт пам'яті для зберігання всіх своїх ваг.

Під час тренування ваги оновлюються за допомогою алгоритмів оптимізації, таких як Adam, щоб мінімізувати функцію втрат і покращити здатність моделі генерувати зв'язний текст. Ваги зберігаються в пам'яті моделі (зазвичай у GPU або TPU) і використовуються для обчислення активацій і генерації тексту під час інференсу.

Розподіл пам'яті між цими трьома компонентами суттєво залежить від розміру моделі та довжини контексту. Для великих моделей з сотнями мільярдів параметрів, таких як GPT-4 або DeepSeek V3, ваги займають домінуючу частку пам'яті — понад 90% у більшості сценаріїв використання. KV-кеш стає значним фактором лише при дуже довгих контекстах (десятки та сотні тисяч токенів), тоді як активації зазвичай займають менше 5% загального обсягу пам'яті.

Важливо розуміти відмінності між споживанням пам'яті під час тренування та інференсу. Під час тренування додатково потрібно зберігати градієнти та стани оптимізатора, що може збільшити вимоги до пам'яті в 3-4 рази порівняно з інференсом. Наприклад, тренування моделі з 7B параметрів у FP32 з оптимізатором Adam потребує близько 84 ГБ пам'яті лише для ваг та станів оптимізатора, без урахування активацій та градієнтів.

Сучасні підходи до тренування використовують різноманітні техніки для зменшення споживання пам'яті: змішану точність (mixed precision тренування) з використанням FP16 або BF16, градієнтне накопичення (gradient accumulation), контрольні точки активацій (activation checkpointing) та

розподілене тренування (distributed тренування) з паралелізмом даних, тензорів та конвеєрів.

Математичний апарат обчислення пам'яті

Загальне використання пам'яті моделлю можна описати формулою:

Mtotal = Mweights + Mkvcache + M_activations

де кожен компонент визначається наступним чином. Пам'ять для ваг моделі:

Mweights = P × bw

де P — кількість параметрів моделі, b_w — кількість байтів на параметр (2 для FP16, 1 для INT8, 0.5 для INT4).

Пам'ять для KV-кеша (для архітектури з Grouped-Query Attention):

Mkvcache = 2 × L × nkvheads × dhead × seqlen × b_kv

де L — кількість шарів, nkvheads — кількість KV-голів, dhead — розмірність голови, seqlen — довжина послідовності, b_kv — байтів на значення.

Для конкретного прикладу розрахунку пам'яті розглянемо модель Qwen3-8B — сучасну щільну модель з 8.2 мільярдами параметрів, випущену Alibaba у квітні 2025 року. Архітектура Qwen3-8B використовує Grouped-Query Attention (GQA) з 32 головами запитів та 8 головами ключів/значень, що значно зменшує розмір KV-кеша порівняно з повною Multi-Head Attention.

Таблиця 2.1 – Архітектура Qwen3-8B

Кількість параметрів	**8.2B**	**6.95B non-embedding**
Кількість шарів (L)	36	---
Розмір прихованого стану	4096	---
Query / KV heads	32 / 8	GQA 4:1
Head dimension	128	---
Нативний контекст	32K	до 128K з YaRN
Пам'ять для ваг (FP16)	\~16.4 ГБ	$8.2\times10^9 \times 2$ байти

Критичність проблеми KV-кеша стає очевидною при розгляді сучасних моделей з довгим контекстом. Розрахуємо розмір KV-кеша для Qwen3-8B при різних довжинах контексту:

Таблиця 2.2 – Розмір KV-кеша для Qwen3-8B при різних контекстах

Контекст	**KV-Cache**	**KV-Cache**	**KV-Cache**
(FP16)	(FP8)	(INT4)	
32K (нативний)	4.5 ГБ	2.25 ГБ	1.13 ГБ
128K (YaRN)	18 ГБ	9 ГБ	4.5 ГБ
1M (Gemini 3 Pro)	\~140 ГБ	\~70 ГБ	\~35 ГБ

Для Qwen3-8B з 1M контекстом у FP16:

$M\_kv = 2 \times 36 \times 8 \times 128 \times 1{,}000{,}000 \times 2 \approx 147.5$ ГБ

Це демонструє, чому оптимізація KV-кеша є важливою для моделей з довгим контекстом — навіть компактна 8B модель потребує понад 140 ГБ пам'яті лише для KV-кеша при мільйонному контексті.

2.2. Методи оптимізації активацій

Розрідженість активацій (activation sparsity) — це явище, коли значна частина активацій (виходів) нейронів у нейронній мережі дорівнює нулю. Це особливо характерно для мереж, які використовують функцію активації ReLU (випрямлена лінійна одиниця). ReLU обнуляє всі від'ємні входи, що призводить до значної частини нульових активацій. Інші популярні функції активації, такі як GELU чи SiLU, дають менше нульових значень.

Математично функція ReLU визначається як:

$\mathrm{ReLU}(x) = \max(0, x)$

Розрідженість активацій призводить до того, що значна частина обчислень (множення на ваги у наступному шарі) дає нульовий результат. Це дозволяє оптимізувати обчислення, пропускаючи ці операції. Крім економії обчислень, розріджені активації також дозволяють зменшити кількість операцій читання ваг з пам'яті. Якщо активація дорівнює нулю, немає потреби читати відповідний рядок матриці ваг.

Сучасні моделі трансформерів, як правило, використовують функції активації GELU або SiLU замість ReLU. Однак дослідження показали, що заміна їх на ReLU (так звана «ReLUfication») дозволяє значно збільшити розрідженість активацій при незначному впливі на якість моделі.

Розрідженість активацій має напівструктурований характер — обнуляються цілі рядки матриці ваг. Це дозволяє ефективно реалізувати відповідні оптимізації на апаратному забезпеченні на відміну від неструктурованої розрідженості, як при одиничному прунінгу вагів.

Розрідженість активацій дозволяє суттєво пришвидшити обчислення в моделях з великою кількістю параметрів, таких як сучасні великі мовні моделі. Використання функцій активації, які сприяють розрідженості, як ReLU, та методи на кшталт «ReLUfication» є перспективним напрямком оптимізації цих моделей. Однак важливо зазначити, що розрідження активацій майже не підвищує ефективність використання пам'яті, через те що активації займають дуже невелику частку пам'яті.

Дослідження показують, що в типових трансформерних моделях рівень природної розрідженості активацій (без спеціальних оптимізацій) становить 30-50% для функцій GELU та SiLU. Застосування ReLUfication може підвищити цей показник до 70-90%, що призводить до пропорційного прискорення обчислень у шарах Feed-Forward Network (FFN), які складають близько двох третин обчислювальних витрат типового трансформера.

Практична реалізація використання розрідженості активацій вимагає спеціалізованого апаратного забезпечення або оптимізованих бібліотек. На стандартних GPU ефективне використання розрідженості досягається через структуровані патерни (наприклад, 2:4 розрідженість на NVIDIA Ampere та новіших архітектурах) або спеціалізовані sparse GEMM бібліотеки.

2.3. Методи оптимізації KV-кеша

Multi-Query Attention (MQA)

Multi-Query Attention — радикальний підхід до оптимізації KV-кеша, запропонований Ноамом Шазером у 2019 році. У цьому методі всі запити використовують один спільний набір ключів та значень, що суттєво зменшує розмір кеша. MQA скорочує обсяг пам'яті, необхідний для зберігання KV-кеша, у кількість разів, рівну кількості голів уваги в моделі.

Хоча MQA забезпечує значне зменшення розміру KV-кеша, це може призвести до певного зниження якості моделі порівняно з повноцінною багатоголовою увагою. Проте для багатьох застосувань компроміс між ефективністю та продуктивністю виявляється прийнятним, особливо для задач, що вимагають обробки довгих послідовностей або роботи на пристроях з обмеженою пам'яттю.

Grouped-Query Attention (GQA)

Grouped-Query Attention є методом, що покращує ефективність використання пам'яті шляхом оптимізації зберігання і завантаження ключових і значеннєвих голів (KV-кеш). GQA ділить головки запитів на групи, де кожна група головок запитів ділить між собою одну ключову і одну значеннєву головки. Наприклад, якщо у моделі є 16 головок запитів, вони можуть бути розділені на 4 групи по 4 головки, де кожна група буде мати спільні ключові і значеннєві головки.

Існуючі моделі з багатоголовою увагою (MHA) можуть бути перетворені на моделі з груповаою увагою (GQA) шляхом усереднення (mean pooling) проєкційних матриць ключових і значеннєвих голів для створення єдиної проєкційної матриці для кожної групи. Це зберігає важливу інформацію і забезпечує швидку адаптацію до нової структури.

GQA є компромісом між повною багатоголовою увагою (MHA), де кожна голова запитів має власну пару ключ-значення, та Multi-Query Attention (MQA), де всі голови запитів ділять одну пару ключ-значення. Типові конфігурації GQA використовують співвідношення 4:1 або 8:1 між головами запитів та KV-головами. Наприклад, LLaMA 2 70B використовує 64 голови запитів та 8 KV-голів, досягаючи 8-кратного зменшення розміру KV-кеша порівняно з MHA.

Емпіричні дослідження показують, що GQA з правильно підібраним співвідношенням груп може досягати якості, дуже близької до MHA, при значно меншому споживанні пам'яті. Це особливо важливо для довгих

контекстів, де KV-кеш може стати домінуючим фактором споживання пам'яті. Сучасні моделі, такі як Qwen 3, LLaMA 3 та Gemma 2, використовують GQA як стандартний механізм уваги.

Квантизація KV-кеша

Квантизація є ефективним методом зменшення розміру KV-кеша шляхом зниження точності представлення даних. Цей підхід передбачає зберігання значень ключів та значень з меншою бітовою глибиною, наприклад, використовуючи 8-бітне або навіть 4-бітне представлення замість стандартного 16-бітного формату з плаваючою комою.

Основною перевагою квантизації є значне зменшення обсягу пам'яті, необхідного для зберігання KV-кеша, що може призвести до 2-4-кратного скорочення розміру кеша. Проте квантизація вимагає ретельного балансування між зменшенням розміру та збереженням якості моделі.

Multi-Head Latent Attention (MLA)

Multi-Head Latent Attention (MLA) — механізм уваги, який забезпечує ефективне виведення шляхом значного зменшення розміру кеш-пам'яті ключів-значень (KV), що зберігається під час генерування тексту. Він використовує спільне низькорангове стиснення ключів та значень в один латентний вектор.

Замість зберігання роздільних великих масивів ключів та значень для кожної голови уваги, MLA спочатку проєктує вхідний прихований стан в низькорозмірний латентний вектор за допомогою матриці проєкції понижуючого зразка. Під час прямого проходження цей латентний вектор розгортається назад у ключі та значення уваги за допомогою матриць проєкції підвищуючого зразка.

Математично MLA працює наступним чином: вхідний прихований стан h розмірності dmodel спочатку проєктується в латентний простір розмірності dlatent (де dlatent \<\< dmodel) за допомогою матриці Wdown. Під час

обчислення уваги латентний вектор c = Wdown × h розгортається у ключі та значення: K = Wupk × c та V = Wupv × c. Це дозволяє зберігати лише латентний вектор c замість повних K та V, досягаючи стиснення в dmodel / dlatent разів.

DeepSeek V2 та V3 використовують MLA з латентною розмірністю 512, що забезпечує приблизно 10-20-кратне стиснення KV-кеша порівняно зі стандартною увагою. Це дозволяє обробляти значно довші контексти при тих самих обмеженнях пам'яті. Важливою особливістю MLA є те, що він вимагає навчання моделі з нуля з цією архітектурою — його не можна застосувати до існуючих моделей без повного перенавчання.

Проєкція низького рангу для стиснення KV-кеша (Palu)

Цей метод, вперше використаний у фреймворку Palu, пропонує новий підхід до стиснення KV-кеша за допомогою проєкції низького рангу. Основна ідея полягає в розкладанні лінійних шарів на матриці низького рангу, кешуванні менших проміжних станів і відновленні повних ключів та значень на льоту.

Експерименти показують, що цей підхід може стиснути KV-кеш більш ніж на 91.25%, зберігаючи при цьому значно кращу точність порівняно з сучасними методами квантизації KV-кеша при аналогічному або навіть більшому використанні пам'яті. На відміну від MLA метод не потребує тренування моделі для його використання.

Cross-Layer Attention (CLA)

Cross-Layer Attention — це новий підхід до оптимізації KV-кеша, який передбачає спільне використання ключів та значень між сусідніми шарами моделі. CLA дозволяє зменшити розмір KV-кеша ще вдвічі порівняно з MQA, зберігаючи при цьому майже таку ж точність.

Комбінування локальної та глобальної уваги

Цей підхід до механізму уваги поєднує локальне та глобальне уважне сканування. Метод передбачає чергування шарів з локальним ковзним вікном уваги та шарів з глобальною увагою.

Типово, розмір ковзного вікна для шарів локальної уваги встановлюється на 4096 токенів, тоді як охоплення шарів глобальної уваги може сягати 8192 токенів. Таке чергування дозволяє ефективно обробляти як локальні, так і глобальні залежності в послідовності.

Сучасні дослідження демонструють ефективність гібридних стратегій уваги через вибір позиційного кодування. Робота «RoPE to NoPE and Back Again» пропонує архітектуру, що чергує шари з Rotary Position Embedding (RoPE) та шари без позиційних ембедингів (NoPE). RoPE-шари відповідають за локальну увагу — позиційне кодування оптимально працює для сусідніх токенів. NoPE-шари забезпечують глобальну увагу — відсутність позиційного кодування усуває проблеми екстраполяції на довгі контексти. Для KV-кеша це означає: RoPE-шари потребують кешу лише для локального вікна, тоді як NoPE-шари зберігають повний кеш, але без прив'язки до позицій.

Практична реалізація чергування локальної та глобальної уваги вимагає ретельного проектування. Шари з локальною увагою мають значно менший KV-кеш (пропорційний розміру вікна), тоді як шари з глобальною увагою потребують повного KV-кеша. Оптимальне співвідношення між типами шарів залежить від конкретного застосування та доступних обчислювальних ресурсів.

Моделі, такі як Mistral, використовують ковзне вікно уваги з розміром 4096 токенів на всіх шарах, досягаючи ефективної обробки послідовностей довжиною до 32K токенів. При цьому модель може «бачити» токени за межами вікна через накопичення інформації в прихованих станах на попередніх шарах.

KV Cache Sharing

KV Cache Sharing, використаний у Gemma 3n, є підходом для прискорення обробки довгих послідовностей, особливо важливим для потокових застосувань. Обробка довгих вхідних даних, таких як послідовності, отримані з аудіо та відео потоків, є критичною для багатьох мультимодальних застосувань на пристроях.

KV Cache Sharing оптимізує фазу початкової обробки вхідних даних (так звану фазу prefill). Ключі та значення середнього шару з локальної та глобальної уваги безпосередньо поділяються з усіма верхніми шарами, що забезпечує приблизно 2-кратне покращення продуктивності фази prefill порівняно з Gemma 3 4B.

Технічно KV Cache Sharing працює наступним чином: замість обчислення та зберігання унікальних ключів і значень для кожного шару, середні шари моделі обчислюють «репрезентативні» K та V, які потім перевикористовуються верхніми шарами. Це значно зменшує обсяг обчислень та пам'яті, необхідних для фази prefill, яка обробляє весь вхідний контекст перед початком генерації.

Для потокових застосувань, таких як обробка аудіо та відео в реальному часі, швидкість фази prefill є критичною — вона безпосередньо впливає на час до першого токена відповіді. KV Cache Sharing дозволяє Gemma 3n швидше почати генерацію при обробці довгих мультимодальних вхідних даних, таких як відео-потоки або тривалі аудіо-записи.

Eviction-методи

Eviction-методи, або методи видалення, є важливим підходом до оптимізації KV-кеша, особливо при роботі з дуже довгими послідовностями. Ці методи дозволяють вибірково видаляти менш важливі токени з KV-кеша під час генерації, підтримуючи обмежений розмір кеша навіть при обробці великих обсягів контексту.

Існують різні стратегії для визначення важливості токенів та їх видалення: видалення за часом (найстаріші токени видаляються першими), видалення на основі уваги (токени з найнижчими значеннями уваги видаляються), адаптивне видалення (комбінація різних метрик для визначення токенів для видалення).

StreamingLLM є прикладом eviction-підходу, який дозволяє моделям обробляти нескінченно довгі потоки тексту. Алгоритм зберігає фіксовану кількість початкових токенів (sink tokens) та останніх токенів (recent window), видаляючи проміжні токени. Дослідження показали, що збереження лише 4 початкових токенів разом з ковзним вікном забезпечує стабільну якість генерації при значному зменшенні розміру KV-кеша.

H2O (Heavy-Hitter Oracle) використовує накопичену статистику уваги для визначення найважливіших токенів. Замість простого видалення старих токенів, H2O зберігає токени, які отримували найбільшу увагу протягом генерації. Це дозволяє зберігати ключову інформацію з раннього контексту, яка залишається релевантною для подальшої генерації.

Scissorhands та подібні методи використовують навчені предиктори для визначення, які токени можуть бути безпечно видалені. Предиктор тренується передбачати вплив видалення кожного токена на якість виходу моделі, дозволяючи робити більш інформовані рішення про видалення порівняно з евристичними методами.

Збільшення ширини моделі

Збільшення ширини моделі замість її глибини є ефективним підходом до оптимізації розміру KV-кеша. При однаковій загальній кількості параметрів, ширші моделі з меншою кількістю шарів мають значно менший розмір KV-кеша порівняно з глибокими вузькими моделями.

Розмір KV-кеша лінійно залежить від кількості шарів у моделі. Наприклад, модель з 1 шаром і прихованим розміром 23,157 має KV-кеш розміром близько 0.71 ГБ для послідовності довжиною 8192 токени. Модель

зі 128 шарами і прихованим розміром 2,048 вимагає близько 8.0 ГБ для тієї ж довжини послідовності.

Таблиця 2.3 – Порівняння механізмів уваги за ефективністю KV-кеша

Метод	KV-голови	Скорочення	Приклади
MHA	= Q голів	1x	GPT-2, BERT
MQA	1	Nx	PaLM, Falcon
GQA	G груп	N/Gx	LLaMA 2, Qwen3
MLA	Латентний	10-20x	DeepSeek V3

2.4. Методи оптимізації ваг моделі

Розрідження ваг

Розрідження ваг (розрідженість ваг) — це ще один підхід до оптимізації пам'яті в LLM, який полягає у зменшенні кількості ненульових параметрів у моделі. У розріджених моделях більшість ваг встановлюються в нуль, що дозволяє зберігати та обробляти тільки ненульові значення, тим самим значно зменшуючи споживання пам'яті та обчислювальні витрати.

Історично розрідження ваг широко використовувалося в згорткових нейронних мережах, де воно може досягати 90%+ розрідженості без суттєвої втрати якості. Однак для трансформерних моделей, особливо великих мовних моделей, досягнення високої розрідженості є складнішим завданням через щільну природу механізму уваги та важливість кожного параметра для збереження знань моделі.

Структуроване розрідження, де обнуляються цілі структурні одиниці (рядки, стовпці, канали), є більш практичним для апаратної реалізації порівняно з неструктурованим розрідженням. NVIDIA Ampere та новіші архітектури підтримують 2:4 структуроване розрідження на апаратному рівні, де з кожних 4 елементів 2 можуть бути нулями. Це забезпечує 2-кратне прискорення матричних операцій без необхідності спеціалізованих бібліотек.

Основні методики розрідження моделей включають: регуляризацію (під час тренування моделі до функції втрат додаються регуляризаційні члени, такі як L1-регуляризація, які заохочують ваги наближатися до нуля), підрізання або

pruning (після тренування моделі ваги з малими абсолютними значеннями встановлюються в нуль, створюючи розріджену структуру), розріджену архітектуру (моделі можуть бути спеціально розроблені з розрідженими з'єднаннями між шарами), розріджене тренування (під час тренування моделі можуть використовуватися спеціальні алгоритми оптимізації, які заохочують розрідженість).

Квантизація

Квантизація — це техніка оптимізації пам'яті, яка використовується для зменшення розміру моделей машинного навчання, зокрема LLM, шляхом зменшення точності представлення ваг та активацій. В оригінальних моделях ваги та активації зазвичай зберігаються як 16-бітні числа з рухомою комою (FP16/BF16), що забезпечує високу точність, але потребує значного обсягу пам'яті. Квантизація дозволяє представляти ваги та активації з меншою точністю, наприклад, використовуючи 8-бітні (INT8), 4-бітні (INT4) або навіть бітові (двійкові) значення.

Процес квантизації включає в себе кілька кроків: визначення діапазону значень (аналіз розподілу ваг для визначення мінімального та максимального значень), масштабування значень (перетворення значень з оригінального діапазону в цільовий діапазон квантизації), квантування значень (округлення масштабованих значень до найближчого дискретного рівня), зберігання квантованих значень (збереження результату разом з параметрами масштабування для відновлення).

Сучасні методи квантизації, такі як GPTQ, LLM.int8() та інші, вже дозволяють стиснути моделі до низьких бітових ширин, до 1.58 біта на параметр, зберігаючи при цьому продуктивність моделі.

GPTQ (GPT-Quantized) є одним з найпопулярніших методів post-тренування квантизації для великих мовних моделей. Алгоритм базується на оптимальному квантуванні мозку (Optimal Brain Quantization) та використовує калібраційний датасет для визначення оптимальних параметрів квантизації.

GPTQ може квантувати моделі до INT4 з мінімальною втратою якості за кілька годин на одному GPU.

LLM.int8() використовує змішану точність, де викиди (outliers) в активаціях обробляються в FP16, тоді як основна маса значень квантується до INT8. Це дозволяє уникнути значної деградації якості, яка виникає при наївному квантуванні, де великі значення-викиди можуть суттєво спотворюватися.

Формат MXFP4, що використовується в GPT-OSS та інших сучасних моделях, є блоковим форматом з 4.25 біта на параметр: кожен блок з 32 значень ділить спільний 8-бітний масштаб, а самі значення зберігаються у 4-бітному форматі. Це забезпечує кращу точність порівняно з простим INT4 при незначному збільшенні розміру.

Квантизація до низьких бітових ширин (1-2 біти) є активною областю досліджень. Методи на кшталт BitNet демонструють, що моделі можуть бути натреновані з тернарними вагами (-1, 0, +1), досягаючи ефективної бітової ширини 1.58 біта на параметр. Такі моделі потенційно можуть працювати на процесорах без GPU, використовуючи лише цілочисельну арифметику.

Таблиця 2.4 – Порівняння методів квантизації

Формат	**Біти**	**Стиснення**	**Втрата якості**
FP16	16	1x (базовий)	Немає
INT8	8	2x	\< 1%
INT4	4	4x	1-3%
MXFP4	4.25	\~3.8x	\< 2%
1.58-bit	1.58	\~10x	3-5%

Факторизація ембедингів

Факторизація ембедингів (factorized embeddings) — це метод зменшення кількості параметрів у таблиці ембедингів (embedding table) в нейронних мережах, зокрема в мовних моделях. Проблема полягає в тому, що для великих словників (десятки тисяч слів) і великих розмірностей ембедингів (сотні), розмір таблиці ембедингів може бути дуже великим — десятки мільйонів параметрів.

Для моделі з словником 50,000 токенів та розмірністю ембедингу 4096, таблиця ембедингів містить 204.8 мільйона параметрів (50,000 × 4,096), що займає близько 400 МБ у FP16. Для моделей з більшими словниками (наприклад, 128K токенів у LLaMA 3) це значення зростає до понад 1 ГБ.

ALBERT (A Lite BERT) був одним з перших застосувань факторизації ембедингів у великих мовних моделях. Автори показали, що зменшення розмірності ембедингів з 768 до 128 з наступною проекцією назад до 768 дозволяє зменшити кількість параметрів ембедингів на 80% без суттєвої втрати якості.

Факторизація ембедингів вирішує цю проблему шляхом представлення матриці ембедингів E у вигляді добутку двох менших матриць:

$$E = U \times W$$

де U — матриця розмірності V × K, а W — матриця розмірності K × H. При цьому K значно менше за H, що дозволяє суттєво зменшити загальну кількість параметрів.

Input-output embedding sharing

Input-output embedding sharing — це техніка, яка використовується в архітектурах sequence-to-sequence. Ідея полягає в тому, щоб використовувати один і той самий набір embedding векторів як для вхідних, так і для вихідних токенів. Основною перевагою цього підходу є суттєве зменшення кількості параметрів у моделі, оскільки один набір ембедингів використовується замість двох окремих.

Universal Transformer

Universal Transformer (UT) — це архітектурний підхід до редукції параметрів, який використовує рекурсивне застосування одного і того ж набору ваг на всіх шарах моделі. На відміну від стандартного Transformer, де кожен шар має унікальні параметри, UT застосовує один трансформерний

блок багаторазово, що дозволяє суттєво зменшити загальну кількість параметрів моделі.

У стандартному Transformer з L шарами кожен шар має власні ваги для механізму уваги та FFN, що дає загальну кількість параметрів пропорційну L. Universal Transformer використовує спільні ваги для всіх ітерацій обробки, зменшуючи кількість параметрів у L разів при збереженні глибини обробки.

Ключовим компонентом UT є механізм позиційного кодування, який оновлюється на кожній ітерації рекурсії. Це дозволяє моделі розрізняти різні етапи обробки та враховувати глибину рекурсії. Формально, на кожному кроці t обробка визначається як:

h^(t) = LayerNorm(h^(t-1) + Attention(h^(t-1))) + FFN(...)

де той самий набір параметрів Attention та FFN застосовується на всіх кроках t = 1, 2, ..., T.

Universal Transformer також може використовувати адаптивний час обчислень (Adaptive Computation Time, ACT), що дозволяє моделі динамічно визначати кількість ітерацій для кожного токена. Складніші токени можуть оброблятися більшою кількістю ітерацій, тоді як простіші — меншою, що підвищує ефективність обчислень.

Основним недоліком Universal Transformer є обмежена ємність моделі порівняно зі стандартним Transformer тієї ж глибини. Спільне використання ваг обмежує здатність моделі до спеціалізації різних шарів на різних аспектах обробки. Це особливо критично для великих мовних моделей, де різні шари виконують різні функції: ранні шари обробляють синтаксичну інформацію, середні — семантичну, пізні — генерують вихідні представлення.

Mixture-of-Experts Universal Transformer (MoEUT)

MoEUT — це вдосконалення Universal Transformer, яке вирішує проблему обмеженої ємності через поєднання рекурсивного застосування шарів з архітектурою Mixture-of-Experts. Замість використання фіксованого

FFN блоку на всіх ітераціях, MoEUT використовує набір експертів з динамічною маршрутизацією.

У MoEUT кожна ітерація обробки включає: спільний механізм уваги (як у звичайному UT); MoE шар з набором експертів, де маршрутизатор динамічно вибирає підмножину експертів для кожного токена на кожній ітерації. Це дозволяє моделі адаптивно використовувати різних експертів на різних етапах обробки одного токена.

Дослідження показують, що в MoEUT спостерігається перевикористання експертів: ті самі експерти виявляються корисними на різних ітераціях обробки. Це підтверджує гіпотезу про те, що певні обчислювальні патерни є універсальними для різних глибин обробки.

MoEUT демонструє, що shared-layer архітектура може бути конкурентною зі стандартними Transformer на задачах мовного моделювання. Однак залишаються невирішені проблеми: складність балансування навантаження між експертами при рекурсивному застосуванні, стабільність тренування та ефективність інференсу.

Архітектурні ідеї Universal Transformer та MoEUT є основою для подальших досліджень редукції параметрів у великих мовних моделях через перевикористання компонентів між шарами.

2.5. Ефективні алгоритмічні реалізації

Стандартна реалізація уваги (N^2)

Стандартна реалізація механізму уваги в трансформерах має квадратичну складність за пам'яттю відносно довжини послідовності. Це пов'язано з необхідністю зберігати матрицю уваги розміром NxN, де N — довжина послідовності.

Формула обчислення уваги має вигляд:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d\_k}) \times V$$

де Q, K, V — матриці запитів, ключів та значень відповідно, d_k — розмірність ключів. Проміжна матриця QK^T має розмір N×N, що і зумовлює квадратичну складність за пам'яттю.

FlashAttention

FlashAttention — це алгоритм для обчислення уваги, який значно зменшує використання пам'яті та прискорює обчислення. Ключова ідея полягає у використанні блочного підходу та перерахунку проміжних результатів замість їх зберігання. FlashAttention розбиває вхідні матриці на блоки, які можуть поміститися в швидку пам'ять GPU (SRAM).

Алгоритм базується на розумінні ієрархії пам'яті GPU: швидка SRAM (shared memory) має пропускну здатність близько 19 ТБ/с, але обмежений розмір (близько 20 МБ на SM), тоді як HBM (High Bandwidth Memory) має більший обсяг (40-80 ГБ), але значно меншу пропускну здатність (1.5-3 ТБ/с). Стандартна реалізація уваги вимагає багаторазових читань та записів у HBM для проміжних результатів, що створює вузьке місце.

FlashAttention використовує техніку тайлінгу (tiling) для обчислення уваги блоками, що повністю поміщаються в SRAM. Для правильного обчислення softmax при блочній обробці використовується онлайн softmax алгоритм, який дозволяє накопичувати часткові результати без зберігання повної матриці уваги. Це вимагає зберігання лише row-wise максимумів та сум для кожного рядка, що займає O(N) пам'яті замість $O(N^2)$.

Алгоритм працює наступним чином: замість обчислення повної матриці уваги N×N, FlashAttention обробляє дані блоками, обчислюючи часткові результати softmax та накопичуючи їх поступово. Це вимагає додаткових обчислень для правильного об'єднання часткових softmax, але значно зменшує обсяг необхідної пам'яті.

FlashAttention досягає лінійного використання пам'яті відносно довжини послідовності, на відміну від квадратичного у стандартної реалізації. Це дозволяє обробляти значно довші послідовності при тих самих обмеженнях

пам'яті. Крім того, алгоритм краще використовує ієрархію пам'яті GPU, що призводить до значного прискорення обчислень.

FlashAttention-2 та FlashAttention-3 розширюють оригінальний алгоритм додатковими оптимізаціями: кращим паралелізмом по головах уваги, оптимізованим використанням тензорних ядер та підтримкою різних форматів точності (FP16, BF16, FP8). FlashAttention-3 досягає до 2x прискорення порівняно з FlashAttention-2 на GPU архітектури Hopper (H100) завдяки використанню нових апаратних можливостей, таких як асинхронне виконання та warp-specialization.

vLLM (PagedAttention)

vLLM (virtual Large Language Model) — це система для ефективного обслуговування великих мовних моделей, яка використовує концепцію віртуальної пам'яті для оптимізації використання GPU пам'яті. Основна ідея полягає в розділенні KV-кеша на блоки фіксованого розміру, подібно до сторінок у віртуальній пам'яті операційних систем.

Традиційні системи обслуговування LLM виділяють неперервний блок пам'яті для KV-кеша кожного запиту на основі максимально можливої довжини послідовності. Це призводить до значної фрагментації пам'яті та неефективного використання GPU ресурсів, оскільки більшість запитів не досягають максимальної довжини. PagedAttention вирішує цю проблему через віртуалізацію пам'яті KV-кеша.

PagedAttention, що лежить в основі vLLM, дозволяє динамічно виділяти та звільняти блоки пам'яті для KV-кеша різних запитів. Це особливо корисно при обслуговуванні багатьох запитів одночасно, оскільки дозволяє уникнути фрагментації пам'яті та ефективно використовувати всю доступну пам'ять GPU.

Архітектура vLLM включає три ключові компоненти: логічну таблицю блоків для відстеження віртуальних блоків KV-кеша; фізичний пул блоків для управління фактичною пам'яттю GPU; та модифікований механізм уваги, який

може працювати з непослідовними блоками пам'яті. Розмір блоку типово становить 16 токенів, що забезпечує баланс між гранулярністю виділення та накладними витратами на управління.

PagedAttention дозволяє досягти майже нульової фрагментації пам'яті та ефективно обробляти запити змінної довжини. Це особливо важливо для систем обслуговування, де одночасно обробляються багато запитів з різними довжинами контексту. Експерименти показують, що vLLM може обслуговувати в 2-4 рази більше запитів одночасно порівняно з традиційними системами при тих самих обмеженнях пам'яті.

Додатковою перевагою PagedAttention є можливість спільного використання KV-кеша між запитами з однаковими префіксами. Наприклад, якщо декілька запитів мають однаковий системний промпт, їхні KV-блоки для цього промпту можуть бути спільними, що додатково економить пам'ять. Ця техніка, відома як prefix caching, особливо ефективна для чат-ботів та інших застосувань з повторюваними шаблонами вводу.

2.6. Надлишковість глибших шарів

Дослідження 2024 року показали, що глибші шари LLM вносять менший внесок у якість моделі, ніж припускалось раніше. Робота «The Unreasonable Ineffectiveness of the Deeper Layers» [26] демонструє, що видалення частини глибших шарів моделі призводить до помірної втрати якості, яку можна частково компенсувати дотренуванням.

ShortGPT [50] досліджує надлишковість шарів у LLM. Автори вводять метрику Block Influence (BI), яка оцінює важливість кожного шару через вимірювання зміни прихований стань після проходження через шар. Результати показують, що частина шарів має низький BI — їх видалення слабо впливає на вихід моделі.

Спостереження з досліджень надлишковості шарів:

Cosine similarity між прихований стань сусідніх шарів часто перевищує 0.9. Середні та глибші шари демонструють більшу надлишковість. LLaMA 2

70B зберігає прийнятну якість після видалення ~25% шарів. Модель демонструє більшу надлишковість по глибині, ніж по ширині.

Це має практичне значення: замість pruning окремих ваг, можна видаляти цілі шари. Такий підхід простіший в реалізації та зберігає структуру моделі, сумісну зі стандартними бібліотеками інференсу.

Надлишковість пов'язана з концепцією залишковий потік: якщо прихований стан містить достатньо інформації, додаткові шари лише незначно його коригують. Це узгоджується з гіпотезою про можливість перевикористання експертів між шарами.

2.7. Архітектури Mixture-of-Experts та інноваційні підходи до ефективності

Концепція Mixture-of-Experts

Mixture-of-Experts (MoE) — це архітектурний підхід, який розділяє обробку між множиною спеціалізованих підмереж (експертів). Замість обробки кожного токена всіма параметрами моделі, MoE використовує механізм маршрутизації для направлення кожного токена до підмножини найбільш відповідних експертів.

Математично вихід MoE шару визначається як:

$$y = \Sigma(i=1 \text{ to } K)\ g_i(x) \times E_i(x)$$

де $g_i(x)$ — ваги маршрутизатора для i-го експерта, $E_i(x)$ — вихід i-го експерта, K — кількість активованих експертів на токен (зазвичай значно менше загальної кількості експертів).

Ключовою особливістю MoE є розділення між загальною кількістю параметрів моделі та кількістю активних параметрів на токен. Це дозволяє зменшити обчислювальні витрати під час інференсу, оскільки активується лише невелика частка загальних параметрів. Проте важливо зауважити, що MoE моделі вимагають зберігання всіх параметрів у пам'яті — модель з 671B параметрів все одно потребує відповідного обсягу пам'яті для ваг, навіть якщо активуються лише 37B на токен.

Основні переваги MoE включають: масштабування до трильйонів параметрів при помірних обчислювальних витратах на токен; зменшення вартості інференсу в 3-10 разів порівняно з еквівалентними щільними моделями; спеціалізація експертів на різних доменах та типах завдань; досягнення цільового loss приблизно в 3 рази швидше при фіксованому обчислювальному бюджеті.

Механізм маршрутизації є критичним компонентом MoE архітектури. Найпоширеніший підхід — топ-k маршрутизація, де для кожного токена вибираються k експертів з найвищими оцінками від маршрутизатора. Типові значення k становлять 1-8, причому більші значення забезпечують кращу якість за рахунок більших обчислювальних витрат. DeepSeek V3 використовує top-6 маршрутизацію з 256 експертів, тоді як Qwen 3 MoE використовує top-8 з 128 експертів.

Проблема балансування навантаження між експертами є однією з ключових при тренуванні MoE моделей. Без спеціальних механізмів маршрутизатор може схилитися до використання лише декількох експертів, що знижує ефективність моделі. Традиційний підхід використовує допоміжні функції втрат (auxiliary losses) для заохочення рівномірного використання експертів, але це може негативно впливати на якість моделі.

Інфраструктурні вимоги для MoE моделей є значними через необхідність зберігання всіх параметрів у пам'яті. Для моделі з 256 експертами та 671B параметрів, як DeepSeek V3, потрібно розподілити експертів між багатьма GPU з використанням expert parallelism. Це вимагає швидкого міжвузлового зв'язку (NVLink, InfiniBand) для ефективного перенаправлення токенів до відповідних експертів. Типова конфігурація для обслуговування DeepSeek V3 включає 8 вузлів по 8 GPU H100/H200.

DeepSeek V3: MoE з MLA

DeepSeek V3, випущений наприкінці 2024 року, демонструє поєднання MoE з Multi-Head Latent Attention (MLA). Модель має 671 мільярд загальних

параметрів, з яких 37 мільярдів активуються на кожен токен. Архітектура включає 256 експертів на шар з одним завжди активним спільним експертом.

Для зберігання ваг DeepSeek V3 у FP16 потрібно близько 1.34 ТБ пам'яті GPU, що вимагає багатовузлової інфраструктури з мінімум 17 GPU H100 (80 ГБ кожен). MLA значно зменшує розмір KV-кеша порівняно зі стандартною увагою, але модель залишається непрактичною для розгортання на обмеженому обладнанні саме через розмір ваг.

Особливості DeepSeek V3 включають: Multi-head Latent Attention (MLA) для ефективного стиснення KV-кеша; DeepSeekMoE з дрібнозернистими експертами та спільними експертами; FP8 змішану точність тренування. Вартість тренування DeepSeek V3 склала лише близько 5.6 мільйонів доларів (2.788 мільйона GPU-годин на H800), що є значно меншим за попередні моделі такого масштабу.

DeepSeekMoE використовує концепцію дрібнозернистих експертів (fine-grained experts) та спільних експертів (shared experts). Замість великих експертів, кожен з яких обробляє значну частину обчислень, DeepSeek використовує 256 менших експертів. Спільний експерт активується для кожного токена разом з вибраними маршрутизованими експертами, забезпечуючи стабільну базову обробку для всіх вхідних даних.

Для балансування навантаження DeepSeek V3 використовує інноваційний підхід без допоміжних функцій втрат (auxiliary-loss-free). Замість традиційних допоміжних функцій втрат, які можуть негативно впливати на якість моделі, використовується динамічне зміщення (bias) для кожного експерта. Це зміщення автоматично коригується під час тренування для забезпечення рівномірного розподілу навантаження.

FP8 тренування є ще однією ключовою інновацією DeepSeek V3. Використання 8-бітного формату з плаваючою комою замість стандартного FP16/BF16 дозволяє зменшити використання пам'яті та збільшити пропускну здатність обчислень. DeepSeek розробив спеціальні техніки для стабілізації тренування в FP8, включаючи детальне масштабування та обробку градієнтів.

Qwen 3: масштабування MoE

Alibaba випустила серію Qwen 3 у квітні 2025 року, яка включає як щільні, так і MoE моделі. Qwen3-30B-A3B має 30 мільярдів загальних параметрів з 3 мільярдами активних на токен, використовуючи 128 експертів на шар з 8 активними. Архітектура включає 48 шарів трансформера та Grouped-Query Attention з 32 головками запитів та 4 KV-головками.

При розгортанні модель вимагає зберігання всіх 30B параметрів (\~60 ГБ у FP16), що робить її придатною для серверів з сучасними GPU, але не для споживчого обладнання. Примітно, що Qwen3-30B-A3B перевершує попередню модель QwQ-32B, яка мала в 10 разів більше активних параметрів, демонструючи ефективність правильно спроектованої MoE архітектури.

Qwen3-235B-A22B є флагманською MoE моделлю серії з 235 мільярдами загальних параметрів та 22 мільярдами активних на токен. Модель підтримує контекст до 128K токенів та використовує 128 експертів з 8 активними на токен.

Gemma 3n: MatFormer та Per-Layer Embeddings

Gemma 3n, випущена Google на I/O 2025, представляє принципово інший підхід до ефективності — архітектуру MatFormer (Matryoshka Transformer), оптимізовану для мобільних пристроїв. На відміну від MoE, MatFormer містить вкладені, повністю функціональні підмоделі в рамках однієї більшої моделі.

Архітектура MatFormer реалізує вкладену структуру Feed-Forward Network (FFN) в стандартних шарах трансформера. Замість фіксованого проміжного розміру FFN, MatFormer використовує ієрархічну організацію, де кожен підблок меншого розміру є повністю навченою життєздатною підмережею. Для Gemma 3n E4B, прихований розмір FFN може варіюватися від 8192 до 16384, причому модель E2B використовує менший розмір, вкладений у більшу модель.

Ключовою інновацією є Per-Layer Embeddings (PLE), які дозволяють значну частину параметрів ембедингів кешувати окремо та обчислювати на CPU, залишаючи в пам'яті прискорювача (GPU/TPU) лише основні ваги трансформера. Традиційні трансформери вбудовують всі параметри в швидку пам'ять прискорювача (GPU/TPU VRAM). Для мобільних пристроїв з обмеженим VRAM (типово 4-8 ГБ) це створює серйозні обмеження. PLE розділяє параметри моделі на дві категорії: основні ваги трансформера, що залишаються в VRAM, та ембединги для кожного шару, що можуть бути завантажені з CPU під час виконання.

Модель Gemma 3n E2B має 5 мільярдів загальних параметрів, але потребує пам'яті як типова 2B модель (\~2 ГБ). E4B має 8 мільярдів параметрів з ефективним споживанням пам'яті як 4B модель (\~3 ГБ). Це досягається через потокове завантаження PLE з кеша під час виведення: для кожного шару і завантажується PLEi з кеша в пам'ять CPU, обчислюється покращення ембедингу, передається покращений вхід на шар і в прискорювачі, видаляється PLEi з пам'яті, переходить до шару i+1.

MatFormer дозволяє вибирати підмоделі різних розмірів (E2B, E3B, E4B) з одного набору ваг без перенавчання. Функція Mix-n-Match дозволяє створювати кастомні проміжні моделі шляхом вибору розміру FFN для кожного шару окремо. На відміну від MoE, де всі експерти потрібно зберігати в пам'яті, MatFormer дозволяє завантажувати лише необхідну частину моделі.

Таблиця 2.5 – Порівняння використання пам'яті сучасних моделей (* – з PLE caching)

Модель	**Ваги**	**KV-кеш 128K**	**Загалом**	**GPU**
DeepSeek V3 (MoE, FP8)	671 ГБ	\~20 ГБ	\~691 ГБ	9x
Qwen3-30B-A3B	60 ГБ	\~18 ГБ	\~78 ГБ	1x
Qwen3-8B	16.4 ГБ	\~18 ГБ	\~34 ГБ	\<1x
Gemma 3n E4B	\~3 ГБ*	N/A (32K)	\~3 ГБ	\<1x
Gemma 3n E2B	\~2 ГБ*	N/A (32K)	\~2 ГБ	\<1x

Таблиця демонструє ключову проблему сучасних LLM: для великих моделей домінуючим фактором споживання пам'яті є ваги, а не KV-кеш.

Навіть значне скорочення KV-кеша не змінює вимог до кількості GPU для DeepSeek V3. Натомість підходи як MatFormer з PLE, що безпосередньо адресують проблему ваг, демонструють реальне зменшення вимог до пам'яті.

У першому розділі проведено комплексний аналіз методів оптимізації використання пам'яті у великих мовних моделях. Виявлено, що основними напрямками оптимізації є: робота з активаціями, оптимізація KV-кеша та зменшення розміру ваг моделі.

Використання пам'яті LLM поділяється на три основні категорії: ваги моделі (домінуюча частка для великих моделей), KV-кеш (зростає з довжиною контексту) та активації (відносно невелика частка). Математичний аналіз демонструє, що для моделей масштабу DeepSeek V3 з 671B параметрів, ваги займають понад 670 ГБ пам'яті у FP8, тоді як KV-кеш навіть при 128K контексті — лише близько 20 ГБ. Це співвідношення є критичним для розуміння пріоритетів оптимізації.

Встановлено, що розрідженість активацій дозволяє пришвидшити обчислення, але має обмежений вплив на зменшення споживання пам'яті через незначну частку активацій у загальному бюджеті пам'яті. Методи ReLUfication дозволяють збільшити розрідженість активацій при незначному впливі на якість моделі.

Проаналізовано широкий спектр методів оптимізації KV-кеша: Multi-Query Attention, Grouped-Query Attention, квантизація KV-кеша, Multi-Head Latent Attention, проєкція низького рангу (Palu), Cross-Layer Attention, KV Cache Sharing, комбінування локальної та глобальної уваги, eviction-методи та збільшення ширини моделі. Проте навіть значні оптимізації KV-кеша не вирішують проблему зберігання ваг для моделей з сотнями мільярдів параметрів.

Розглянуто методи оптимізації ваг моделі: розрідження ваг, квантизацію, факторизацію ембедингів та спільне використання ембедингів. Встановлено, що квантизація є найбільш зрілим та ефективним методом для

зменшення споживання пам'яті, дозволяючи стискати моделі до 1.58 біта на параметр без суттєвої втрати якості.

Проаналізовано ефективні алгоритмічні реалізації, такі як FlashAttention та vLLM (PagedAttention), які дозволяють значно зменшити використання пам'яті та прискорити обчислення за рахунок оптимізації роботи з апаратним забезпеченням та уникнення квадратичної складності за пам'яттю.

Mixture-of-Experts архітектури, такі як DeepSeek V3 та Qwen 3, зменшують обчислювальні витрати через активацію лише частини параметрів на токен, але не вирішують проблему пам'яті для ваг — всі параметри повинні зберігатися в пам'яті. Модель з 671B параметрів вимагає багатовузлової інфраструктури незалежно від кількості активних параметрів на токен.

Підхід MatFormer у Gemma 3n з Per-Layer Embeddings демонструє альтернативну парадигму — пряме зменшення вимог до пам'яті через кешування ембедингів на CPU та вкладену структуру моделі. Це дозволяє моделі з 8B параметрів працювати з ефективним споживанням пам'яті як 4B модель (\~3 ГБ).

Результати аналізу показують, що оптимізація ваг моделі є найбільш перспективним напрямком для реального зменшення вимог до пам'яті. Хоча існує багато методів оптимізації KV-кеша, це не є пріоритетом для більшості розробників моделей, оскільки основні моделі оптимізуються для хмарних середовищ, де обмеження пам'яті для ваг можна вирішити додаванням GPU. Для демократизації доступу до потужних LLM та їх розгортання на споживчому обладнанні необхідні архітектурні інновації, що безпосередньо адресують проблему зберігання ваг, як це демонструє підхід MatFormer з Per-Layer Embeddings.

РОЗДІЛ 3

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1. Технологічний стек

JAX як основа для експериментів

JAX — бібліотека для обчислень від Google, яка поєднує синтаксис NumPy з можливостями трансформації функцій. На відміну від PyTorch, де обчислювальний граф будується динамічно, JAX використовує функціональний підхід: програма описується як композиція чистих функцій, які потім компілюються.

Можливості JAX для тренування нейронних мереж:

Автоматичне диференціювання. Функція grad обчислює градієнти довільного порядку. JAX диференціює через контрольний потік (if/for/while), що спрощує реалізацію складних архітектур.

JIT-компіляція. Декоратор @jit компілює Python-функцію в оптимізований XLA код. Компілятор об'єднує операції, оптимізує пам'ять та розміщує обчислення на пристрої. Для transformer-моделей це дає 2-5× прискорення.

Векторизація через vmap. Перетворює функцію для одного прикладу на батчеву версію автоматично.

Паралелізація через pmap. Розподіляє обчислення на кілька пристроїв (TPU, GPU) з автоматичною синхронізацією.

Підтримка TPU. JAX розроблявся разом з TPU і працює з ним без додаткових бібліотек.

TPU v4

TPU (Tensor Processing Unit) — прискорювач від Google для матричних операцій. TPU v4 — четверте покоління.

Архітектура. Кожен чіп містить два TensorCores з матричними блоками 128×128. Продуктивність ~275 TFLOPS на чіп у bfloat16. Пропускна здатність пам'яті 1.2 TB/s (HBM2e).

TPU v4-32. Конфігурація з 32 ядер, з'єднаних мережею ICI. Топологія 4×4×2 забезпечує швидку синхронізацію градієнтів.

Для MoE. Динамічний роутинг токенів створює нерегулярні патерни доступу до пам'яті. TPU обробляє їх завдяки високій пропускній здатності HBM.

Вибір стеку

Вибір JAX + TPU:

Доступ. TPU Research Cloud надає безкоштовний доступ до TPU v4 для дослідницьких проєктів.

Швидкість. XLA компіляція та архітектура TPU дають високу утилізацію для attention та FFN.

Розпаралелювання. pmap дозволяє реалізувати паралелізм з мінімальним кодом.

Відтворюваність. Функціональний підхід JAX спрощує відтворення експериментів.

3.2. Експеримент 1: валідація методів редукції

Мета

Перший експеримент перевіряє концепцію shared MoE layer на малому масштабі перед застосуванням до Qwen 3.

Гіпотеза: якщо репрезентації на різних шарах схожі, один набір MoE експертів може обслуговувати всі шари. Замість 12 окремих FFN блоків використовуємо один shared MoE, який викликається на кожному шарі.

Якщо shared MoE не працює на 50M моделі, підхід не масштабується на 30B.

Baseline: Dense LLaMA 125M

Для порівняння тренуємо щільна модель:

Таблиця 3.1 – Архітектура Dense LLaMA 125M

Параметр	Значення
Шари	12
Hidden size	768
Attention heads	12
FFN intermediate	3072 (4× hidden)
Vocabulary	32,000
Загальні параметри	~125M

Архітектура LLaMA: RMSNorm, SwiGLU, RoPE, pre-norm.

Архітектура експериментальної моделі

Таблиця 3.2 – Архітектура експериментальної Shared MoE моделі

Параметр	Значення
Шари	12
Hidden size	768
Attention heads (Q)	12
KV heads (GQA)	4
Vocabulary	32,000
Embedding factor dim	128
Shared MoE	
Експертів	16
Expert intermediate size	720
Активних на токен	2

Компоненти: RMSNorm, SwiGLU в експертах, RoPE.

Архітектурні рішення

Shared MoE Layer. Один MoE блок з 16 експертами на всіх 12 шарах — аналог Universal Transformer для FFN частини.

На кожному шарі прихований стан проходить через той самий MoE блок. Роутер обирає 2 з 16 експертів. Це ті самі 16 експертів на всіх шарах, не копії.

Економія: замість 12 × FFN (~56M) маємо 1 × MoE (~26.5M).

Grouped-Query Attention. 12 голів Q поділено на 4 групи. Кожна група має спільні K, V проєкції розміром 192. Зменшує параметри K, V втричі.

Факторизовані спільні ембединги. Матриця 32000×768 факторизується: E = U × W, де U розміром 32000×128, W розміром 128×768. Зменшення з 24.6M до 4.2M.

Вихідна проєкція використовує ту саму матрицю (зв'язування ваг).

Розподіл параметрів

Таблиця 3.3 – Розподіл параметрів експериментальної моделі

Компонент	Параметри	Частка
Факторизовані ембединги	4.2M	8.4%
Attention (12 шарів)	18.9M	37.8%
Shared MoE (16 експертів)	26.5M	53.0%
Маршрутизатор + RMSNorm	~0.03M	0.1%
Загалом	~50M	100%

Більше половини параметрів у shared MoE — він має замінити 12 окремих FFN.

Тренування

Датасет: FineWeb-Edu — освітній контент з Common Crawl.

Конфігурація:

Розмір батчу: 512 послідовностей Довжина: 1024 токени Темп навчання: 3e-4, косинусне затухання до 3e-5 Розігрів: 1000 кроків Оптимізатор: AdamW (β_1=0.9, β_2=0.95, затухання ваг=0.1) Обмеження градієнту: 1.0 Обладнання: TPU v4-32

Балансування навантаження. Auxiliary loss з коефіцієнтом 0.01 заохочує рівномірне використання експертів.

Результати

Таблиця 3.4 – Порівняння результатів тренування

Модель	Параметри	Final loss
Dense LLaMA (baseline)	125M	3.12
Shared MoE + GQA + факт. емб.	50M	3.15

Криві loss обох моделей практично співпадають. Різниця ~0.03 (менше 1%) — в межах варіації.

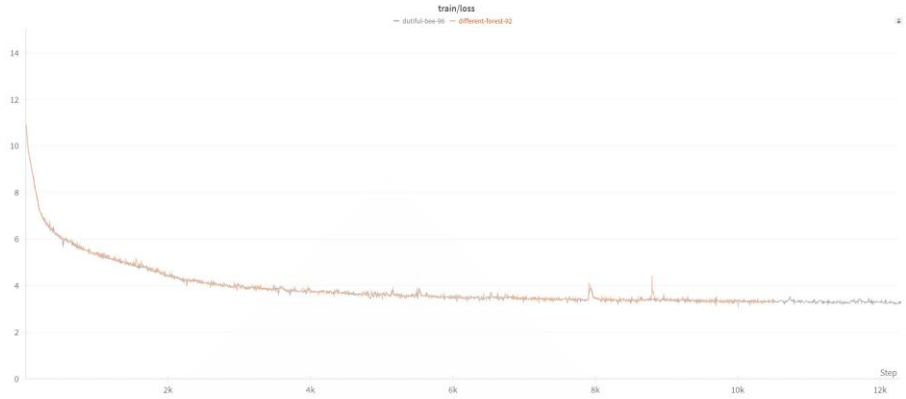

Рисунок 3.1 – Криві тренувального loss для Dense LLaMA та Shared MoE моделей

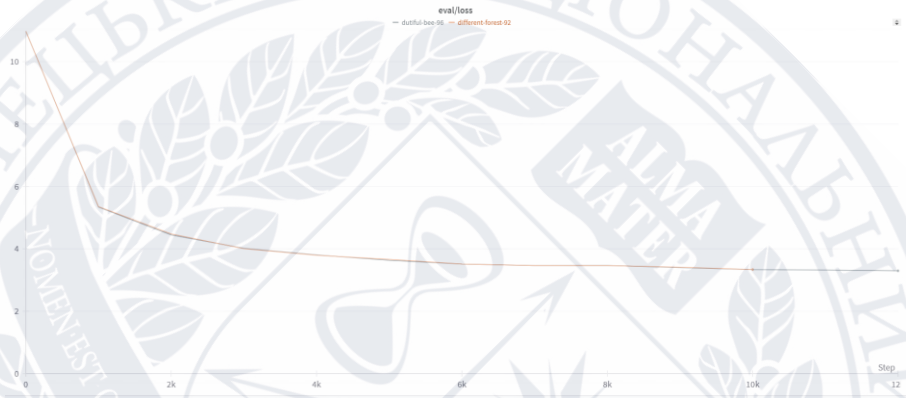

Рисунок 3.2 – Криві валідаційного loss для Dense LLaMA та Shared MoE моделей

Рисунки 3.1 та 3.2 демонструють динаміку тренування обох моделей протягом 12000 кроків. Обидві криві показують типову поведінку оптимізації: швидке зниження loss на початку тренування з поступовим виходом на плато. Важливо відзначити, що модель Shared MoE з 50M параметрами досягає практично ідентичних показників loss порівняно з Dense моделлю з 125M параметрами.

Аналіз

Результат підтверджує гіпотезу: один пул експертів може замінити 12 окремих FFN. Модель з 50M параметрів (40% від baseline) досягає порівнянної якості.

Експерти вивчили функції, корисні на різних глибинах. Роутер на шарі 3 і шарі 10 може обирати тих самих експертів — репрезентації достатньо схожі.

Для Qwen 3: якщо 16 shared експертів замінюють 12 dense FFN, то 6144 експертів у глобальному пулі мають надлишковість.

3.3. Експеримент 2: конвертація Qwen 3 MoE

Архітектура Qwen 3 30B-A3B

Qwen 3 30B-A3B — Mixture-of-Experts модель від Alibaba з наступними характеристиками:

Таблиця 3.5 – Характеристики Qwen 3 30B-A3B

Параметр	Значення
Загальні параметри	30.5B
Активні параметри	3.3B
Кількість шарів	48
Голови Q / KV	32 / 4 (GQA)
Кількість експертів	128
Активних експертів на токен	8
Розмір контексту	32K (131K з YaRN)
Тренувальні токени	36T

Архітектура використовує: GQA для ефективного KV-кеша, SwiGLU активацію, RoPE позиційне кодування, RMSNorm з qk-нормалізацією, global-батч load balancing loss для стабільного тренування MoE.

На відміну від попередніх Qwen MoE моделей, Qwen 3 не використовує shared experts — всі 128 експертів спеціалізовані. Маршрутизатор вибирає top-8 експертів для кожного токена на основі softmax scores.

Ідея конвертації

Стандартна MoE архітектура має окремий пул експертів на кожному шарі: 48 шарів × 128 експертів = 6144 FFN-блоки. Роутер кожного шару вибирає лише зі «своїх» 128 експертів. Це обмеження: експерт з шару 15 недоступний для токена, який потребує саме його на шарі 30.

Ідея конвертації: об'єднати всі FFN-експерти в єдиний пул і тренувати новий глобальний роутер, який вибирає з усіх 6144 експертів незалежно від їх оригінального шару. Attention-шари залишаються без змін — 48 окремих шарів зі своїми вагами.

Оригінальна архітектура:

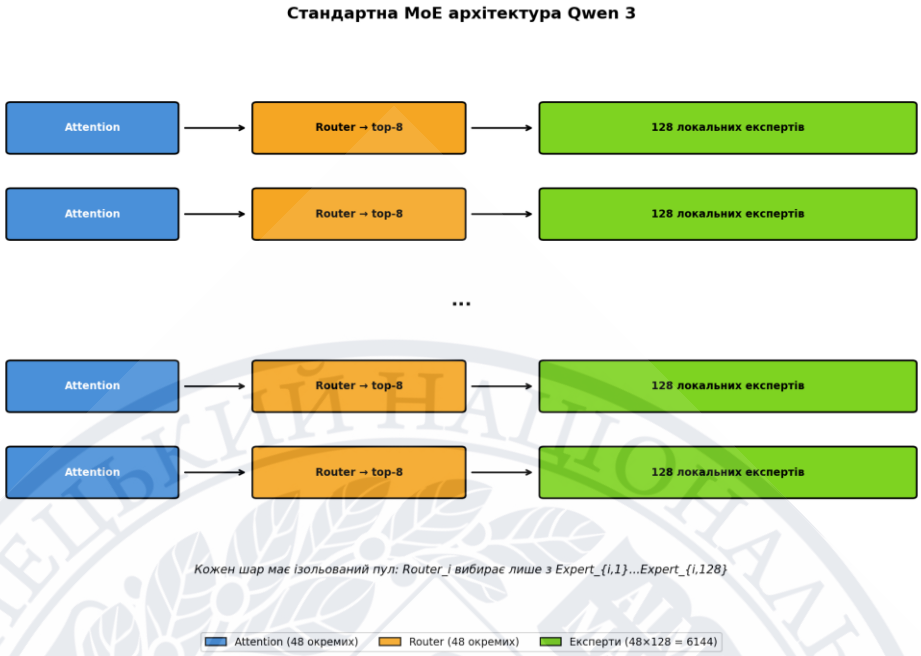

Рисунок 3.3 – Стандартна MoE архітектура Qwen 3

Конвертована архітектура:

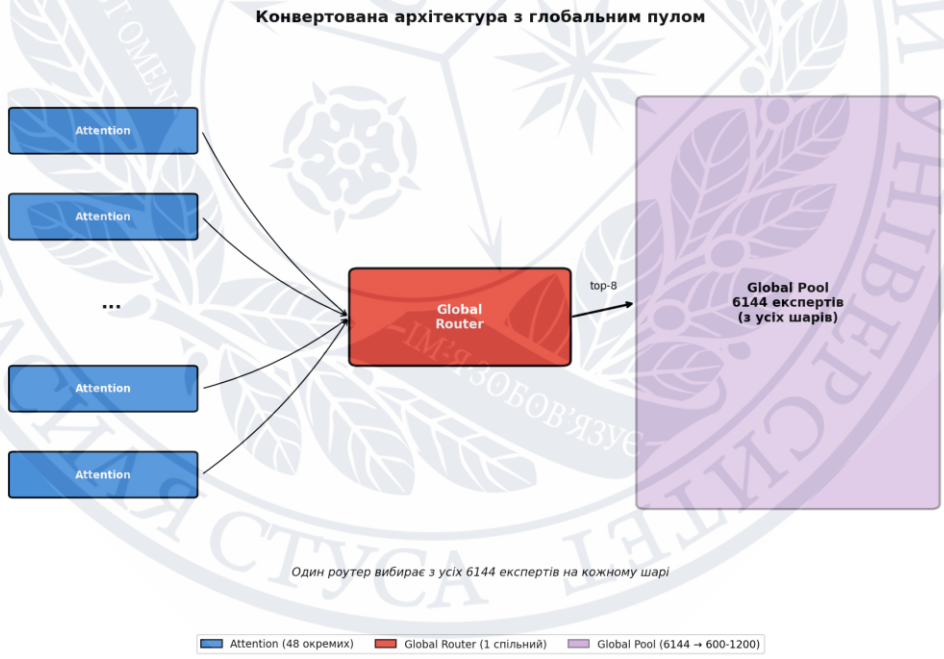

Рисунок 3.4 – Архітектура з глобальним пулом експертів

Переваги підходу

Більший вибір. Роутер вибирає з 6144 експертів замість 128. Вища ймовірність знайти оптимального експерта для конкретного токена.

Перевикористання експертів. Якщо експерт корисний на різних «глибинах» обробки, він може бути обраний на кількох шарах. Це наближає архітектуру до Universal Transformer.

Виявлення надлишковості. Після тренування роутера стає очевидним, які експерти ніколи не обираються — їх можна видалити через REAP.

Зменшення загальних параметрів. Якщо 50% експертів виявляться непотрібними, модель зменшується з 30B до ~15B при збереженні 3.3B активних.

Механізм роутингу

Замість softmax, який «розмазує» ймовірності на 6144 експертів, використовуємо sigmoid — як у DeepSeek V3:

Sigmoid на кожен експерт незалежно: $a_e = \sigma(W_e \cdot x)$ — score від 0 до 1. Top-k вибір: 8 експертів з найвищими scores. Нормалізація серед обраних: $g_e = a_e / \Sigma a_{e'}$ для e' з top-k.

Sigmoid вирішує проблему градієнтів: кожен експерт оцінюється незалежно, немає «розмазування» як у softmax на 6144 класів.

Теоретичне обґрунтування

Стандартна інтерпретація transformer як послідовності незалежних шарів є спрощенням. Точніша модель — залишковий потік: безперервний потік інформації, до якого кожен компонент (attention, FFN) додає свій внесок. Математично: $x_n = x_0 + \Sigma\ \Delta_i$ для i від 1 до n-1, де Δ_i — внесок i-го шару. Прихований стан на будь-якій глибині містить оригінальний сигнал плюс накопичені корекції.

З цього випливає: репрезентації на різних шарах не є принципово різними об'єктами. Вони живуть у спільному векторному просторі, кожен шар лише уточнює позицію. Cosine similarity між сусідніми шарами висока (0.8-0.95), а logit lens показує, що проміжні репрезентації можна декодувати в токени на будь-якій глибині.

Для FFN-експертів це означає: експерт — функція $f: \mathbb{R}^d \to \mathbb{R}^d$, яка приймає нормалізований прихований стан і повертає корекцію. Якщо прихований стань на різних шарах схожі за структурою, експерт з шару 40 може видавати корисні корекції і на шарі 10.

Universal Transformer демонструє це: один блок застосовується рекурсивно до репрезентації різної глибини. MoEUT показує, що MoE-експерти у shared-layer архітектурі перевикористовуються — ті самі експерти корисні на різних ітераціях.

Чому очікуємо 80-90% надлишковості

Стандартний REAP видаляє 20-50% експертів без суттєвої втрати якості. Але там експерти конкурують лише всередині свого шару: 8 активних з 128 доступних. Кожен шар — ізольований пул.

У нашій архітектурі 6144 експерти конкурують глобально за кожну активацію. Експерт, який виконував функцію X на шарі 15, тепер конкурує з експертами, що виконували функцію X на шарах 1-48. Якщо функція X універсальна, один найкращий експерт може замінити 48 схожих.

Дослідження MoE моделей показують значну кореляцію між експертами різних шарів: вони активуються на схожих патернах, видають схожі виходи. Під час попереднього тренування градієнти формують схожі функції на різних шарах, бо задача однакова.

Крім того, 128 експертів на шар — штучне обмеження для зручності паралелізації, а не оптимальна архітектура. DeepSeek та інші роботи показують, що модель може досягти тієї ж ємності меншою кількістю спеціалізованих експертів.

Гіпотеза: при глобальній конкуренції 80-90% експертів або дублюють функції інших, або не критичні для якості. Роутер сконцентрується на 600-1200 найкорисніших з 6144.

Що очікуємо спостерігати

Після тренування роутера очікуємо побачити характерні патерни:

Універсальні експерти. Невелика група експертів, які активуються на багатьох шарах для різних входів.

Спеціалізовані експерти. Експерти, які активуються рідко, але критичні для специфічних задач (код, математика, певні мови).

Мертві експерти. Значна частина експертів з майже нульовою частотою активації — кандидати на видалення.

Кластеризація по функціях. Експерти, що виконують схожі функції (незалежно від оригінального шару), матимуть схожі патерни активації.

Розподіл важливості очікуємо асиметричним: невелика кількість експертів з високою важливістю, довгий хвіст з низькою. Це типово для надлишкових систем.

Тренування глобального роутера

Етап 1: Дистиляція. Глобальний роутер ініціалізується випадково. Тренуємо його імітувати оригінальні 48 роутерів через KL-divergence на відповідних зрізах:

$L\_distill = \Sigma\ KL(\sigma(R_i(x)) \,||\, \sigma(G(x)[128i:128(i+1)]))$ для i від 1 до 48

де R_i — оригінальний роутер шару i, G — глобальний роутер. Кілька тисяч кроків достатньо — це лише початкова ініціалізація, щоб модель не розвалилась від випадкового вибору.

Етап 2: Тренування на LM loss. Ваги експертів та attention заморожені, тренується лише роутер на передбачення наступного токена. Роутер отримує градієнти від реальної задачі і може виявити, що експерти з інших шарів корисні.

Конфігурація:

Датасет: FineWeb-Edu Розмір батчу: 64, довжина послідовності: 2048 Темп навчання: 1e-4 з розігрів Обладнання: TPU v4-32

Метод REAP

REAP (Маршрутизатор-weighted Expert Activation Pruning) — метод видалення експертів для MoE моделей (Lasby et al., 2025). Ідея: видаляти експертів з найменшим внеском, а не об'єднувати їх.

Чому видалення краще за об'єднання? При об'єднанні експертів роутер втрачає незалежний контроль над ними. Якщо роутер видає різні ваги для двох токенів, об'єднаний експерт не може відтворити цю різницю. Математично: якщо політика роутера залежить від входу і експерти не ідентичні, об'єднання завжди вносить помилку.

При видаленні роутер зберігає контроль над експертами, що залишились. Помилка масштабується з вагою видалених експертів — чим рідше їх використовували, тим менша втрата.

Застосування REAP

Після тренування роутера багато експертів матимуть низьку важливість — їх рідко вибирають, або вони мають малий вплив на вихід. REAP видаляє таких експертів.

Оцінка важливості:

$S_e = E[g_e(x) \cdot ||f_e(x)||_2]$ для x з розподілу D

де $g_e(x)$ — нормалізована вага роутера, $f_e(x)$ — вихід експерта.

Очікування: при глобальній конкуренції 6144 експертів виявиться значна надлишковість.

Гіпотеза: 80-90% експертів можна видалити (6144 → 600-1200), зберігаючи якість.

Конфігурація експерименту

Вхідна модель: Qwen 3 30B-A3B

48 attention шарів (зберігаються) 6144 FFN експерти (об'єднуються в пул) Глобальний роутер (тренується)

Цільові метрики:

Видалення 80-90% експертів (6144 → 600-1200) Збереження ≥90% якості на бенчмарках (код, математика) Зменшення пам'яті: 30B → 5-8B параметрів

Етапи:

Конвертація архітектури (об'єднання пулу експертів) Дистиляція роутера Тренування роутера з LM loss Збір статистики важливості REAP pruning (80-90% експертів) Файнтюнінг роутера Оцінка на бенчмарках

Результати

На момент написання роботи експеримент з конвертації Qwen 3 30B-A3B знаходиться на етапі виконання. Процес об'єднання експертів у глобальний пул та подальший REAP pruning потребує значних обчислювальних ресурсів. Остаточні результати будуть представлені під час захисту роботи.

3.4. Висновки до розділу

Експериментальне дослідження включає два підходи до редукції параметрів, побудовані на спільній ідеї — перевикористання MoE експертів між шарами.

Результати першого експерименту

Експеримент на малому масштабі (50M параметрів) підтвердив гіпотезу: один shared MoE layer замінює 12 окремих FFN блоків. Модель досягла якості щільної LLaMA 125M при 2.5× меншій кількості параметрів.

Висновок: репрезентації на різних глибинах достатньо схожі, щоб один набір експертів обслуговував усі шари.

Дизайн другого експерименту

Другий експеримент масштабує підхід на Qwen 3 30B-A3B:

Об'єднання 6144 експертів (48 шарів × 128) у глобальний пул Тренування глобального роутера на sigmoid (як у DeepSeek V3) REAP pruning для видалення надлишкових експертів

При глобальній конкуренції очікуємо 80-90% надлишковості — більше ніж 20-50% у стандартному REAP.

Порівняння експериментів

Таблиця 3.6 – Порівняння експериментів

Аспект	Експеримент 1	Експеримент 2
Масштаб	50M параметрів	30B → 5-8B
Підхід	Тренування з нуля	Конвертація моделі
Експерти	16 shared	6144 → 600-1200
Роутер	Тренується з моделлю	Тренується окремо
Мета	Валідація концепції	Компресія моделі

Перший експеримент — перевірка, що shared experts працюють. Другий — застосування для компресії великої моделі.

Очікувані результати та ризики

Оптимістичний сценарій. 80-90% експертів видаляються, модель 30B → 5-8B при збереженні >90% якості.

Консервативний сценарій. 50-70% експертів видаляються, модель ~12-15B.

Ризики:

Колапс роутингу — концентрація на малій кількості експертів Втрата рідкісних функцій (математика, рідкісні мови) при агресивному pruning Повільніший роутер на 6144 експертів порівняно з 48 локальних на 128

Протидія:

Балансування навантаження під час тренування Поступовий pruning (спочатку 50%, потім до 80-90%) Оцінка на різних бенчмарках перед кожним етапом

Методологія другого експерименту

Етапи:

Конвертація — об'єднання пулу експертів, ініціалізація роутера. Дистиляція — роутер імітує 48 оригінальних через KL-divergence. Тренування — роутер оптимізується на передбачення токенів. Збір статистики — оцінка важливості експертів. REAP pruning — видалення 80-90% експертів. Файнтюнінг — адаптація після pruning. Оцінка — тестування на бенчмарках.

ВИСНОВКИ

У кваліфікаційній роботі досліджено проблему редукції параметрів великих мовних моделей для наближення передових моделей до можливостей споживчого обладнання. Основні результати роботи:

1. Проаналізовано сучасний стан методів оптимізації пам'яті у великих мовних моделях. Виявлено, що основними напрямками є: оптимізація KV-кеша (MQA, GQA, MLA), квантизація ваг, факторизація ембедингів та архітектури Mixture-of-Experts. Встановлено, що для моделей масштабу DeepSeek V3 з 671B параметрів ваги займають понад 670 ГБ пам'яті у FP8, що є домінуючим фактором.

2. Обґрунтовано теоретичну можливість перевикористання MoE експертів між шарами на основі концепції залишковий потік. Показано, що репрезентації на різних шарах живуть у спільному векторному просторі з високою косинусна подібність (0.8-0.95).

3. Розроблено та експериментально валідовано метод shared MoE layer. Модель з 50M параметрів досягла якості щільної LLaMA 125M (фінальний loss 3.15 vs 3.12), підтвердивши можливість заміни 12 окремих FFN блоків одним набором з 16 експертів.

4. Запропоновано методологію конвертації існуючих MoE моделей (Qwen 3 30B-A3B) шляхом об'єднання 6144 експертів у глобальний пул з наступним застосуванням REAP pruning. При глобальній конкуренції очікується редукція 80-90% експертів.

5. Очікуване практичне значення: зменшення вимог до пам'яті з 30B до 5-8B параметрів при збереженні ≥90% якості, що робить модель доступною для розгортання на споживчому обладнанні (32 ГБ VRAM).

Подальші напрямки дослідження включають: експериментальну верифікацію другого експерименту; дослідження впливу на специфічні домени (математика, код); адаптацію методу для інших MoE архітектур (DeepSeek V3, Kimi K2).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani A. et al. Attention is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008. URL: https://arxiv.org/abs/1706.03762
2. Kaplan J. et al. Scaling Laws for Neural Language Models. arXiv. 2020. arXiv:2001.08361. URL: https://arxiv.org/abs/2001.08361
3. Hoffmann J. et al. Тренування Compute-Optimal Large Language Models. arXiv. 2022. arXiv:2203.15556. URL: https://arxiv.org/abs/2203.15556
4. Allen-Zhu Z., Li Y. Physics of Language Models. arXiv. 2024. arXiv:2407.20311. URL: https://arxiv.org/abs/2407.20311
5. Shazeer N. et al. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. ICLR. 2017. URL: https://arxiv.org/abs/1701.06538
6. DeepSeek-AI. DeepSeek-V3 Technical Report. arXiv. 2024. arXiv:2412.19437. URL: https://arxiv.org/abs/2412.19437
7. Moonshot AI. Kimi K2 Technical Report. 2025. URL: https://github.com/MoonshotAI/Kimi-K2
8. Qwen Team. Qwen3 Technical Report. arXiv. 2025. arXiv:2505.09388. URL: https://arxiv.org/abs/2505.09388
9. Dehghani M. et al. Universal Transformers. ICLR. 2019. URL: https://arxiv.org/abs/1807.03819
10. Csordás R., Irie K., Schmidhuber J. MoEUT: Mixture-of-Experts Universal Transformers. arXiv. 2024. arXiv:2405.16039. URL: https://arxiv.org/abs/2405.16039
11. Lasby S. et al. REAP the Experts: Why Pruning Prevails for One-Shot MoE Compression. arXiv. 2025. arXiv:2510.13999. URL: https://arxiv.org/abs/2510.13999

12. Elhage N. et al. A Mathematical Framework for Transformer Circuits. Anthropic. 2021. URL: https://transformer-circuits.pub/2021/framework/index.html
13. Geva M. et al. Transformer Feed-Forward Layers Build Predictions by Promoting Concepts in the Vocabulary Space. EMNLP. 2022. URL: https://arxiv.org/abs/2203.14680
14. Shazeer N. Fast Transformer Decoding: One Write-Head is All You Need. arXiv. 2019. arXiv:1911.02150. URL: https://arxiv.org/abs/1911.02150
15. Ainslie J. et al. GQA: Тренування Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. EMNLP. 2023. URL: https://arxiv.org/abs/2305.13245
16. Dao T. et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. NeurIPS. 2022. URL: https://arxiv.org/abs/2205.14135
17. Kwon W. et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. SOSP. 2023. URL: https://arxiv.org/abs/2309.06180
18. Frantar E. et al. GPTQ: Accurate Post-Тренування Quantization for Generative Pre-trained Transformers. ICLR. 2023. URL: https://arxiv.org/abs/2210.17323
19. Dettmers T. et al. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. NeurIPS. 2022. URL: https://arxiv.org/abs/2208.07339
20. Lan Z. et al. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. ICLR. 2020. URL: https://arxiv.org/abs/1909.11942
21. Frostig R. et al. Compiling machine learning programs via high-level tracing. MLSys. 2018. URL: https://mlsys.org/Conferences/2019/doc/2018/146.pdf
22. Bondarenko A. et al. One Wide Feedforward is All You Need. arXiv. 2023. arXiv:2309.01826. URL: https://arxiv.org/abs/2309.01826
23. Ma S. et al. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. arXiv. 2024. arXiv:2402.17764. URL: https://arxiv.org/abs/2402.17764

24. Mirzadeh I. et al. ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models. arXiv. 2023. arXiv:2310.04564. URL: https://arxiv.org/abs/2310.04564

25. Frantar E., Alistarh D. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. arXiv. 2023. arXiv:2301.00774. URL: https://arxiv.org/abs/2301.00774

26. Gromov A. et al. The Unreasonable Ineffectiveness of the Deeper Layers. arXiv. 2024. arXiv:2403.17887. URL: https://arxiv.org/abs/2403.17887

27. Su J. et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. arXiv. 2021. arXiv:2104.09864. URL: https://arxiv.org/abs/2104.09864

28. Radford A. et al. Language Models are Unsupervised Multitask Learners. OpenAI. 2019. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

29. OpenAI. GPT-4 Technical Report. arXiv. 2023. arXiv:2303.08774. URL: https://arxiv.org/abs/2303.08774

30. Gemma Team. Gemma 2: Improving Open Language Models at a Practical Size. arXiv. 2024. arXiv:2408.00118. URL: https://arxiv.org/abs/2408.00118

31. Wang H. et al. DeepNet: Scaling Transformers to 1,000 Layers. arXiv. 2022. arXiv:2203.00555. URL: https://arxiv.org/abs/2203.00555

32. Character.AI. Optimizing AI Інференс at Character.AI. 2024. URL: https://research.character.ai/optimizing-інференс/

33. Pope R. et al. Efficiently Scaling Transformer Інференс. arXiv. 2022. arXiv:2211.05102. URL: https://arxiv.org/abs/2211.05102

34. Rabe M. N., Staats C. Self-attention Does Not Need $O(n^2)$ Memory. arXiv. 2021. arXiv:2112.05682. URL: https://arxiv.org/abs/2112.05682

35. Liu Z. et al. MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. arXiv. 2024. arXiv:2402.14905. URL: https://arxiv.org/abs/2402.14905

36. Muralidharan S. et al. LLM Pruning and Distillation in Practice: The Minitron Approach. arXiv. 2024. arXiv:2408.11796. URL: https://arxiv.org/abs/2408.11796

37. Chang J. et al. Palu: Compressing KV-Cache with Low-Rank Projection. arXiv. 2024. arXiv:2407.21118. URL: https://arxiv.org/abs/2407.21118

38. Brandon W. et al. Reducing Transformer Key-Value Cache Size with Cross-Layer Attention. arXiv. 2024. arXiv:2405.12981. URL: https://arxiv.org/abs/2405.12981

39. Dettmers T. et al. Intriguing Properties of Quantization at Scale. arXiv. 2023. arXiv:2305.19268. URL: https://arxiv.org/abs/2305.19268

40. Hooper C. et al. KVQuant: Towards 10 Million Context Length LLM Інференс with KV Cache Quantization. arXiv. 2024. arXiv:2401.18079. URL: https://arxiv.org/abs/2401.18079

41. Llama Team. The Llama 3 Herd of Models. arXiv. 2024. arXiv:2407.21783. URL: https://arxiv.org/abs/2407.21783

42. Penedo G. et al. FineWeb: decanting the web for the finest text data at scale. 2024. URL: https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

43. Jouppi N. et al. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv. 2023. arXiv:2304.01433. URL: https://arxiv.org/abs/2304.01433

44. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. arXiv. 2014. arXiv:1412.6980. URL: https://arxiv.org/abs/1412.6980

45. DeepSeek-AI. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv. 2024. arXiv:2405.04434. URL: https://arxiv.org/abs/2405.04434

46. Wang L. et al. Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts. arXiv. 2024. arXiv:2408.15664. URL: https://arxiv.org/abs/2408.15664

47. Devvrit F. et al. MatFormer: Nested Transformer for Elastic Інференс. arXiv. 2023. arXiv:2310.07707. URL: https://arxiv.org/abs/2310.07707

48. Google DeepMind. Gemma 3n Technical Report. 2025. URL: https://developers.googleblog.com/en/introducing-gemma-3n/

49. Yang B. et al. RoPE to NoPE and Back Again: A New Hybrid Attention Strategy. arXiv. 2025. arXiv:2501.18795. URL: https://arxiv.org/abs/2501.18795

50. Men X. et al. ShortGPT: Layers in Large Language Models are More Redundant Than You Expect. arXiv. 2024. arXiv:2403.03853. URL: https://arxiv.org/abs/2403.03853