Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Метод побудови мовних мікро-моделей на основі LLM та методу їх екстремального стиснення
(тема)

Виконав:
здобувач другого року навчання,
групи СШІздм-24-1
Олексій АРТЕМЕНКО
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)
Керівник доц. Євген ПАВЛЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Лариса ЧАЛА
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра Штучного інтелекту

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Артеменку Олексію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод побудови мовних мікро-моделей на основі LLM та методу їх екстремального стиснення

затверджена наказом університету від 6 квітня 20 26 р. № 48Стз

2. Термін подання здобувачем роботи до екзаменаційної комісії 26 травня 20 26 р.

3. Вихідні дані до роботи науково-технічні публікації за тематикою великих мовних моделей, тернарної квантизації (BitNet b1.58) та дистиляції знань; офіційна документація фреймворків PyTorch, Transformers, bitnet.cpp; відкриті набори даних WikiText-103, C4 та The Pile для навчання і оцінювання мовних моделей; специфікація відкритих моделей Llama-2/3, Qwen, Gemma; референсна реалізація BitNet від Microsoft Research

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та огляд архітектур великих мовних моделей
2) Огляд методів стиснення нейронних мереж та їх порівняльний аналіз
3) Дослідження підходу 1.58-біт квантизації на прикладі BitNet b1.58
4) Розробка методу побудови мовних мікромоделей на основі LLM

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	06.04.2026	виконано
2	Аналіз предметної галузі та огляд літератури	10.04.2026	виконано
3	Огляд методів стиснення мовних моделей	14.04.2026	виконано
4	Дослідження архітектури BitNet b1.58	18.04.2026	виконано
5	Розробка методу побудови мікромоделі	24.04.2026	виконано
6	Програмна реалізація методу	30.04.2026	виконано
7	Проведення експериментальних досліджень	04.05.2026	виконано
8	Написання пояснювальної записки	06.05.2026	виконано
9	Перевірка на академічний плагіат	07.05.2026	виконано
10	Нормоконтроль	08.05.2026	виконано
11	Рецензування	10.05.2026	виконано
12	Підготовка презентації та доповіді	12.05.2026	виконано
13	Попередній захист	14.05.2026	виконано
14	Захист перед ЕК	20.05.2026	

Дата видачі завдання 6 квітня 20 26 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Євген ПАВЛЕНКО
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 84 с., 24 рис., 12 табл., 2 дод., 28 джерел.

ВЕЛИКА МОВНА МОДЕЛЬ, ДИСТИЛЯЦІЯ ЗНАНЬ, ЕКСТРЕМАЛЬНЕ СТИСНЕННЯ, МІКРОМОДЕЛЬ, НЕЙРОННА МЕРЕЖА, ТЕРНАРНА КВАНТИЗАЦІЯ, ТРАНСФОРМЕР, BITNET.

Об’єкт дослідження – процес побудови компактних мовних моделей на основі великих мовних моделей із застосуванням методів екстремального стиснення.

Предмет дослідження – методи, моделі та алгоритми екстремального стиснення великих мовних моделей шляхом тернарної квантизації, дистиляції знань та структурного прунінгу для отримання функціональних мовних мікромоделей.

Мета роботи – розробка методу побудови мовних мікромоделей на основі великих мовних моделей (LLM) із використанням екстремального стиснення, який забезпечує суттєве зменшення обсягу пам'яті та обчислювальних витрат за умови збереження прийнятної якості генерації тексту.

Методи дослідження – теоретичний аналіз існуючих підходів до стиснення нейронних мереж; методи квантизаційно-орієнтованого навчання (QAT); дистиляція знань «вчитель-учень» (knowledge distillation); структурний прунінг; експериментальне моделювання з використанням мови Python, бібліотек PyTorch та Transformers, фреймворку bitnet.cpp; порівняльний аналіз на стандартизованих бенчмарках.

ABSTRACT

Master’s thesis contains: 84 pp., 24 fig., 12 tabl., 2 ann., 28 references.

BITNET, EXTREME COMPRESSION, KNOWLEDGE DISTILLATION, LARGE LANGUAGE MODEL, MICRO-MODEL, NEURAL NETWORK, TERNARY QUANTIZATION, TRANSFORMER.

Object of research – the process of building compact language models based on large language models using extreme compression techniques.

Subject of research – methods, models and algorithms for extreme compression of large language models through ternary quantization, knowledge distillation and structural pruning to obtain functional language micro-models.

Purpose of the work – development of a method for constructing language micro-models based on Large Language Models (LLMs) using extreme compression, which provides a substantial reduction in memory footprint and computational costs while preserving acceptable text generation quality.

Research methods – theoretical analysis of existing approaches to neural network compression; quantization-aware training (QAT) methods; teacherstudent knowledge distillation; structural pruning; experimental modeling using the Python programming language, PyTorch and Transformers libraries, and the bitnet.cpp framework; comparative analysis on standardized benchmarks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГПП – графічний процесор;

ММ – мовна модель;

МН – машинне навчання;

НМ – нейронна мережа;

ПЗ – програмне забезпечення;

ЦПУ – центральний процесор;

ШІІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – програмний інтерфейс застосунку;

AWQ – Activation-aware Weight Quantization – квантизація ваг з урахуванням активацій;

BF16 – Brain Floating Point 16-bit – 16-бітний формат чисел з плаваючою комою;

BitNet – архітектура 1-бітних нейронних мереж від Microsoft Research;

BPE – Byte-Pair Encoding – кодування пар байтів (токенізація);

CNN – Convolutional Neural Network – згорткова нейронна мережа;

CPU – Central Processing Unit – центральний процесор;

CUDA – Compute Unified Device Architecture – архітектура паралельних обчислень NVIDIA;

FP16 – Floating Point 16-bit – 16-бітний формат чисел з плаваючою комою;

FP32 – Floating Point 32-bit – 32-бітний формат чисел з плаваючою комою;

GGUF – GPT-Generated Unified Format – уніфікований формат збереження квантизованих моделей;

GPT – Generative Pre-trained Transformer – генеративний попередньо навчений трансформер;

GPTQ – Gradient Post-Training Quantization – градієнтна посттренувальна квантизація;

GPU – Graphics Processing Unit – графічний процесор;

INT4 -4-bit Integer – 4-бітний цілочисловий формат;

INT8 -8-bit Integer – 8-бітний цілочисловий формат;

KD – Knowledge Distillation – дистиляція знань;

KV-cache – Key-Value cache – кеш ключів та значень у механізмі уваги;

LLM – Large Language Model – велика мовна модель;

MLP – Multi-Layer Perceptron – багатошаровий перцептрон;

NLP – Natural Language Processing – обробка природної мови;

PTQ – Post-Training Quantization – посттренувальна квантизація;

QAT – Quantization-Aware Training – квантизаційно-орієнтоване навчання;

ReLU – Rectified Linear Unit – функція активації;

RMSNorm – Root Mean Square Normalization – нормалізація за середньоквадратичним значенням;

RoPE – Rotary Position Embedding – обертові позиційні вбудовування;

STE – Straight-Through Estimator – оцінювач прямого проходження градієнта;

SwiGLU – Swish-Gated Linear Unit – функція активації у FFNшарах;

W1.58A8 – схема квантизації з 1,58-біт вагами та 8-біт активаціями.

ВСТУП

Останнє десятиліття характеризується стрімким розвитком технологій штучного інтелекту, серед яких особливе місце посідають великі мовні моделі (LLM, Large Language Models). Моделі на кшталт GPT-4, Llama 3, Claude, Gemini та інших продемонстрували здатність виконувати широкий спектр задач обробки природної мови на рівні, що наближається до людського, а в деяких задачах – перевищує його. Однак разом зі зростанням якості цих моделей суттєво збільшуються й вимоги до обчислювальних ресурсів: сучасні LLM містять від десятків мільярдів до трильйонів параметрів та потребують десятків гігабайтів оперативної пам'яті й спеціалізованого апаратного забезпечення (багатопроцесорних GPU-кластерів) для здійснення як навчання, так і інференсу.

Такі вимоги стають критичним бар’єром для розгортання мовних моделей у численних практичних сценаріях: на мобільних пристроях, вбудованих системах, IoT-платформах, edge-серверах, а також у додатках, де важливими є швидкодія відгуку, локальне виконання без передачі даних у хмарні сервіси та низький рівень енергоспоживання. Саме тому особливої актуальності набуває задача побудови мовних мікромоделей компактних моделей, що займають від десятків до сотень мегабайт пам’яті та здатні виконуватись на пристроях з обмеженими ресурсами, при цьому зберігаючи прийнятний рівень якості.

У 2024 році дослідники Microsoft Research запропонували архітектуру BitNet b1.58, у якій вагові параметри моделі обмежуються тернарною множиною $\{-1,0,+1\}$, що в середньому відповідає 1,58 біта на параметр ($\log_2 3 \approx 1.58$). Опубліковані результати показують, що така модель при належному масштабуванні досягає якості, порівнянної з повноточними FP16-моделями аналогічного розміру, водночас забезпечуючи суттєве скорочення обсягу пам'яті та енергоспоживання.

У 2025 році Microsoft випустила відкриту модель BitNet b1.58 2B4T, навчену з нуля на 4 трильйонах токенів, що стала першим повнофункціональним нативним 1-бітним LLM на масштабі 2 мільярди параметрів.

Однак підхід BitNet у його класичному вигляді вимагає навчання моделі «з нуля» з тернарними обмеженнями, що пов’язане зі значними обчислювальними витратами та не дозволяє безпосередньо скористатися перевагами вже існуючих передтренованих LLM. Виникає потреба у розробці гібридних методів, які би поєднували переваги тернарної квантизації з можливостями передачі знань від великих передтренованих моделей-вчителів до компактних моделей-учнів.

Актуальність теми кваліфікаційної роботи обумовлена необхідністю створення ефективних методів побудови мовних мікромоделей, що забезпечують баланс між якістю генерації та обчислювальною ефективністю, а також високим попитом на такі моделі з боку промисловості, мобільної розробки та наукових досліджень у сфері edge-штучного інтелекту.

Мета роботи – розробка методу побудови мовних мікромоделей на основі великих мовних моделей з використанням екстремального стиснення, який забезпечує значне зменшення обсягу пам'яті та обчислювальних витрат за умови збереження прийнятної якості генерації тексту.

Для досягнення поставленої мети у роботі вирішуються такі задачі:

– провести аналіз сучасного стану предметної галузі, зокрема архітектур великих мовних моделей та методів їх стиснення;

– дослідити підхід 1,58-біт тернарної квантизації на прикладі Microsoft BitNet b1.58, включно з математичним обгрунтуванням та принципами роботи шарів BitLinear;

– виконати порівняльний аналіз методів post-training quantization (GPTQ, AWQ, GGUF) та quantization-aware training з точки зору якості та ефективності стиснення;

– розробити метод побудови мовних мікромоделей, що комбінує дистиляцію знань від великої моделі-вчителя, тернарну квантизацію параметрів та структурний прунінг;

– побудувати математичну модель оцінювання ефективності запропонованого стиснення;

– реалізувати запропонований метод у вигляді програмного забезпечення мовою Python з використанням бібліотек PyTorch та Transformers;

– провести експериментальні дослідження якості отриманих мікромоделей на стандартизованих наборах даних WikiText-103 та C4 з використанням метрик perplexity, BLEU та ROUGE;

– здійснити аналіз отриманих результатів та сформулювати рекомендації щодо практичного застосування методу.

Об’єкт дослідження – процес побудови компактних мовних моделей на основі великих мовних моделей із застосуванням методів екстремального стиснення.

Предмет дослідження – методи, моделі та алгоритми екстремального стиснення великих мовних моделей шляхом тернарної квантизації, дистиляції знань та структурного прунінгу для отримання функціональних мовних мікромоделей.

Методи дослідження. У роботі використовуються теоретичні методи наукового пізнання: аналіз, синтез, узагальнення, порівняння; методи математичного моделювання для формалізації процесів квантизації та дистиляції; методи експериментального дослідження для перевірки ефективності запропонованого методу; методи математичної статистики для обробки експериментальних даних. Програмна реалізація базується на мові Python 3.11 з використанням фреймворків PyTorch, HuggingFace

Transformers, bitnet.cpp, а також екосистеми інструментів HuggingFace Hub для роботи з моделями та наборами даних.

Наукова новизна отриманих результатів полягає в тому, що вперше запропоновано комплексний метод побудови мовних мікромоделей, який синергетично об’єднує тернарну квантизацію у стилі BitNet b1.58, дистиляцію знань від великої моделі-вчителя та структурний прунінг, забезпечуючи при цьому кращий баланс якості та ефективності, ніж окремі методи стиснення. Додатково побудовано математичну модель, яка дозволяє апріорі оцінити очікувану ступінь стиснення та допустиме зниження якості.

Практична цінність результатів роботи полягає у можливості розгортання мовних моделей на пристроях з обмеженими обчислювальними ресурсами – мобільних телефонах, вбудованих системах, IoT-пристроях. Це відкриває широкі перспективи для застосування LLM у новому класі продуктів: локальних асистентах, офлайн-перекладачах, системах приватного діалогу без передачі даних у хмару.

Зв'язок з науковою тематикою кафедри. Робота виконана в руслі наукового напрямку кафедри штучного інтелекту Харківського національного університету радіоелектроніки – «Інтелектуальні інформаційні технології обробки природної мови та знань».

1 ТЕОРЕТИЧНІ ОСНОВИ МОВНИХ МОДЕЛЕЙ ТА МЕТОДІВ ЇХ СТИСНЕННЯ

1.1 Архітектура великих мовних моделей

Сучасні великі мовні моделі базуються на архітектурі трансформера, запропонованій Vaswani та співавторами у 2017 році в роботі «Attention is All You Need» [1]. Трансформер повністю відмовився від рекурентних зв'язків, характерних для попередніх мовних моделей (RNN, LSTM, GRU), та грунтується на механізмі самоуваги (self-attention), що дозволяє ефективно моделювати довгі контексти та обробляти вхідні послідовності паралельно.

Формально, архітектура трансформера складається з декількох однотипних блоків, кожен з яких містить: шар багатоголової самоуваги (multihead self-attention), повноз'єднаний прямопрямуючий шар (feed-forward network, FFN), а також шари нормалізації та залишкові з'єднання (residual connections). Послідовність вхідних токенів спочатку перетворюється у векторні представлення (embeddings) фіксованої розмірності d_{model} , до яких додаються позиційні кодування.

Механізм самоуваги обчислюється за формулою:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V, \tag{1.1}$$

де Q – матриця запитів (queries);

K – матриця ключів (keys);

V – матриця значень (values);

d_k – розмірність ключів.

Багатоголова увага обчислюється паралельно для декількох «голів» і конкатенується:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O, \quad (1.2)$$

де $\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$;

W_i^Q, W_i^K, W_i^V, W^O – параметризовані матриці проєкцій.

Великі мовні моделі представляють собою декодер-орієнтовану модифікацію трансформера, у якій використовується причинна маскована увага (causal masked attention), що забезпечує авторегресивну генерацію токенів: кожен наступний токен передбачається на основі всіх попередніх.

Навчання такої моделі здійснюється за допомогою мінімізації функції втрат крос-ентропії:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log P_{\theta(x_t|x_{<t})}, \quad (1.3)$$

де θ – параметри моделі;

x_t – токен у позиції t;

T – довжина послідовності.

Сучасні LLM, такі як Llama, Qwen, Gemma, модифікують базову архітектуру трансформера шляхом запровадження низки покращень, наведених у таблиці 1.1.

Таблиця 1.1 – Основні архітектурні модифікації у сучасних LLM

Компонент	Класичний трансформер	Сучасні LLM
Нормалізація	LayerNorm	RMSNorm
Активація FFN	ReLU, GELU	SwiGLU
Позиційне кодування	Синусоїдальне	RoPE
Bias у лінійних шарах	Присутні	Відсутні
Маскування	Повна увага + causal	Causal маска + GQA
KV-cache	Не використ.	Використ. для інференсу

Обсяг параметрів сучасних LLM варіюється від 1–3 млрд (Phi-3, Qwen2.5-1.5B, Gemma-2B) до десятків та сотень мільярдів (Llama-3.1-70B, Llama-3.1-405B, GPT-4 оцінюється у понад трильйон параметрів). Збереження параметрів у форматі FP16 потребує від 2 до 800 ГБ пам'яті відповідно.

Загальна структура типового блоку сучасної LLM наведена на рисунку 1.1.

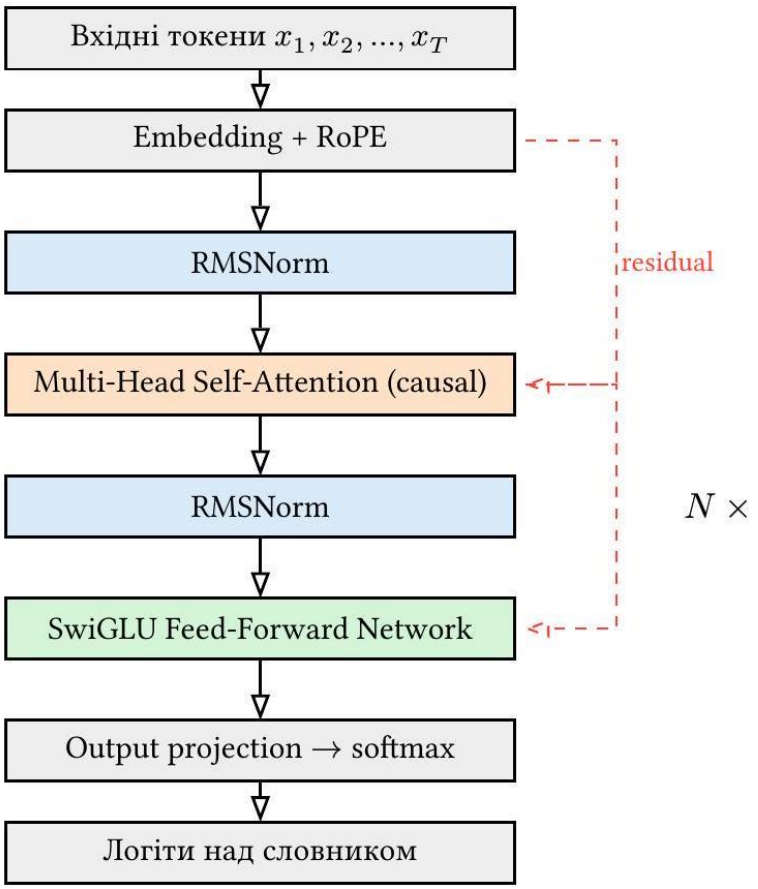

Рисунок 1.1 – Узагальнена структура блоку сучасної LLM

Обсяг пам'яті, необхідний для зберігання параметрів моделі, лінійно залежить від числа параметрів та бітової глибини їх представлення:

$$M = N \cdot \frac{b}{8} \text{ байт}, \tag{1.4}$$

де N – кількість параметрів;

b – кількість біт на параметр.

Так, модель з 7 млрд параметрів у FP16 ($b = 16$) потребує 14 ГБ пам'яті, тоді як ця ж модель у BitNet-представленні ($b \approx 1.58$) потребує лише близько 1,38 ГБ, що майже у 10 разів менше.

1.2 Огляд методів стиснення нейронних мереж

Методи стиснення нейронних мереж можна класифікувати за принципом дії на декілька основних категорій: квантизація, дистиляція знань, прунінг, факторизація тензорів та їх комбінації. Коротка класифікація наведена на рисунку 1.2.

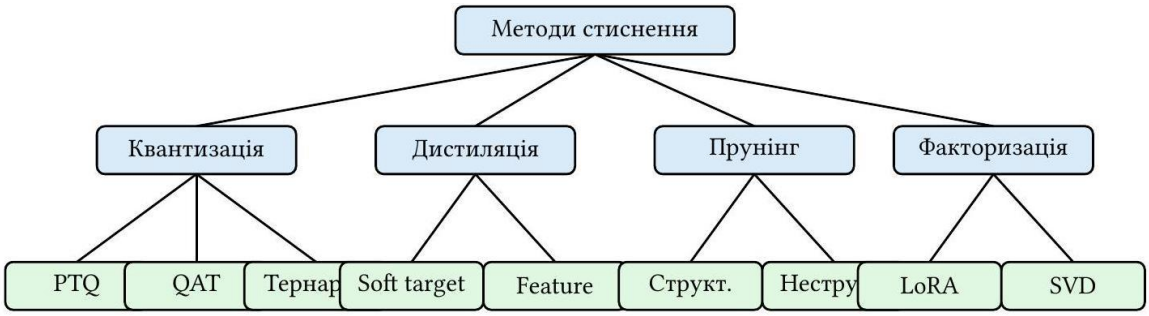

Рисунок 1.2 – Класифікація основних методів стиснення нейронних мереж

Квантизація полягає у зменшенні бітової глибини представлення вагових параметрів та/або активацій моделі. Наприклад, перехід від FP32 ($b = 32$) до INT8 ($b = 8$) зменшує обсяг пам'яті у 4 рази. Квантизація поділяється на посттренувальну (PTQ), коли вже навчена модель квантизується без додаткового навчання, та квантизаційно-орієнтоване навчання (QAT), коли квантизаційні обмеження враховуються під час навчання.

Дистиляція знань (knowledge distillation, KD) була запропонована Hinton та співавторами [2] та грунтується на парадигмі «вчитель-учень»:

велика попередньо навчена модель-вчитель передає свої знання компактній моделі-учневі шляхом мінімізації різниці між їхніми вихідними розподілами ймовірностей. Така передача відбувається не лише через жорсткі мітки (hard targets), але й через м'які розподіли ймовірностей (soft targets), які містять додаткову інформацію про структуру задачі.

Прунінг – це процес вилучення «неважливих» параметрів нейронної мережі. Неструктурний прунінг видаляє окремі ваги (часто на основі їх абсолютного значення), тоді як структурний прунінг видаляє цілі компоненти – канали, голови уваги, шари.

Факторизація тензорів представляє великі матриці ваг у вигляді добутку менших матриць меншого рангу. Найвідомішим прикладом є підхід LoRA (Low-Rank Adaptation) [3], який застосовується в основному для ефективного донавчання моделей, але також використовується як метод стиснення.

На практиці комбінація кількох методів стиснення (наприклад, дистиляція + квантизація + прунінг) дає кращі результати, ніж застосування кожного методу окремо. Саме такий комбінований підхід покладено в основу запропонованого у цій роботі методу.

1.3 Квантизація як основний метод стиснення

Квантизація – це процес перетворення дійснозначних параметрів нейронної мережі у представлення з меншою бітовою глибиною, що дозволяє зменшити обсяг пам'яті та прискорити обчислення. Формально, b-бітне квантизаційне відображення $Q: \mathbb{R} \to \mathbb{Q}_b$ переводить дійсне число x у дискретне значення $\hat{x}$ з множини допустимих значень $\mathbb{Q}_b$ потужністю $|\mathbb{Q}_b| = 2^b$ (або менше, як у випадку тернарної квантизації).

Лінійна (uniform) квантизація є найпростішим і найпоширенішим варіантом:

$$\hat{x} = \text{round}\left(\frac{x}{s}\right) \cdot s + z, \tag{1.5}$$

де s – масштабний коефіцієнт (step size, scale);

z – zero-point зсув.

Для симетричної квантизації $z = 0$, а s обирається таким чином, щоб максимальне за модулем значення в ваговій матриці відображалось на максимальне допустиме квантизоване значення.

Для 8-бітної симетричної квантизації ваг діапазон представлення [-127,127], а масштаб обчислюється як:

$$s = \frac{\max(|W|)}{127}. \tag{1.6}$$

Абсмедіанна (absmean) квантизація використовується в BitNet та має вигляд:

$$\widehat{W} = \text{ round\_to\_ternary } \left(\frac{W}{\alpha}\right), \alpha = \frac{1}{nm}\sum_{i,j} \left|W_{ij}\right|, \tag{1.7}$$

де α – середнє абсолютне значення елементів вагової матриці розміру $n \times m$.

Функція round_to_ternary визначається як:

$$\text{round\_to\_ternary } (y) = \begin{cases} +1, y > 0.5 \\ 0, -0.5 \leq y \leq 0.5. \\ -1, y < -0.5 \end{cases} \tag{1.8}$$

Ключова проблема квантизації – це недиференційованість оператора округлення, що ускладнює застосування градієнтного спуску під час QAT. Для її вирішення використовується Straight-Through Estimator (STE):

$$\frac{\partial \hat{x}}{\partial x} \approx 1 \text{ для } |x| \leq x_{\max}, \text{ інакше } 0. \quad (1.9)$$

Тобто під час зворотного проходу градієнт «проходить наскрізь» квантизатора без змін, імітуючи диференційованість.

Залежно від того, які елементи моделі квантизуються, розрізняють кілька схем, наведених у таблиці 1.2.

Таблиця 1.2 – Типові схеми квантизації для трансформерних

Схема	Ваги	Активації	Типове використання
W8A8	8 біт	8 біт	Стандартна квантизація для серверного інференсу
W4A16	4 біт	16 біт	GPTQ, AWQ - популярні для LLM
W4A8	4 біт	8 біт	Агресивне стиснення для мобільних пристроїв
W2A8	2 біт	8 біт	Експериментальні режими, значне зниження якості
W1.58A8	$\log_2 3$	8 біт	BitNet b1.58 - тернарна QAT
W1A8	1 біт	8 біт	Оригінальний BitNet, бінарна квантизація

Чим нижча бітова глибина, тим сильніше стиснення, але й тим вища ймовірність деградації якості моделі. Тому ключовим викликом сучасних досліджень є пошук методів, які забезпечують екстремальне стиснення (до 1-2 біт) без суттєвої втрати функціональності.

1.4 Дистиляція знань

Дистиляція знань – це техніка, що дозволяє «переносити» знання з великої, потужної, але обчислювально дорогої моделі-вчителя (teacher) у меншу, швидшу модель-учня (student). Основна ідея полягає в тому, що вихідний розподіл ймовірностей моделі-вчителя несе більше інформації, ніж жорсткі мітки, оскільки містить дані про відносні ймовірності різних варіантів.

Класична функція втрат дистиляції Хінтона визначається як зважена комбінація двох компонент:

$$\mathcal{L}_{\mathrm{KD}} = (1-\alpha)\cdot \mathcal{L}_{\mathrm{CE}}(y, p_s) + \alpha \cdot T^2 \cdot \mathcal{L}_{\mathrm{KL}}(p_t^T \| p_s^T), \qquad (1.10)$$

де p_s – розподіл моделі-учня;

p_t – розподіл моделі-вчителя;

y – істинні мітки;

T – температура softmax;

$\alpha \in [0,1]$ – ваговий коефіцієнт;

$\mathcal{L}_{\mathrm{CE}}$ – крос-ентропія;

$\mathcal{L}_{\mathrm{KL}}$ – дивергенція Кульбака-Лейблера.

«Softmax з температурою» визначається як:

$$p_i^T = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}, \qquad (1.11)$$

де z_i – логіти моделі. При $T = 1$ отримуємо стандартний softmax, при $T > 1$ розподіл стає «згладженішим», що полегшує перенесення тонкої інформації про відносні ймовірності.

Для мовних моделей застосовуються декілька модифікацій KD:

– Sequence-level KD – учень навчається відтворювати повні послідовності, згенеровані вчителем;

– Feature-based KD – мінімізується різниця між проміжними представленнями (hidden states) вчителя та учня;

– Attention-based KD – переносяться матриці уваги з вчителя до учня [4];

– Self-distillation – вчитель та учень мають однакову архітектуру, але учень ініціалізується підмножиною параметрів вчителя.

Ефективність дистиляції залежить від ряду факторів: співвідношення розмірів вчителя та учня (оптимальне – 2–10 разів), якості вчителя, набору даних для дистиляції, температурного параметра T та коефіцієнта α у формулі рівняння (1.10).

У контексті даної роботи KD розглядається як ключовий компонент, що дозволяє сформувати компактну архітектуру мікромоделі, яка потім піддається тернарній квантизації для подальшого стиснення.

1.5 Структурний та неструктурний прунінг

Прунінг (від англ. pruning – обрізка) є одним із найстаріших методів стиснення нейронних мереж. Його ідея полягає у вилученні параметрів, які мають найменший вплив на вихідний сигнал моделі. Існує два основних типи прунінгу: неструктурний та структурний.

Неструктурний прунінг видаляє окремі ваги на основі деякого критерію важливості. Найпростішим критерієм є модуль ваги: ваги, модуль яких менший за певний поріг, обнуляються. Цей підхід називають magnitude pruning. Після такого прунінгу модель стає розрідженою, але її обсяг у пам'яті не зменшується автоматично – потрібна підтримка розріджених обчислень на рівні апаратного забезпечення.

Структурний прунінг видаляє цілі структурні елементи моделі: нейрони, канали, голови уваги, цілі шари трансформера. Саме такий підхід дозволяє реально зменшити розмір моделі та прискорити її інференс без спеціалізованого апаратного забезпечення.

Для трансформерних моделей найбільш релевантними є такі варіанти структурного прунінгу:

– Head pruning – вилучення окремих голів уваги. Дослідження показують, що значна частина голів у навчених трансформерах є надлишковою;

– Layer pruning – видалення цілих трансформерних блоків, особливо у середній частині моделі;

– Dimension pruning – зменшення внутрішньої розмірності d_{ffn} у FFNшарах;

– Width pruning – зменшення d_{model} через видалення каналів.

Критерії важливості для структурного прунінгу включають: L_1- та L_2-норми ваг; градієнтні оцінки (наприклад, $|w \cdot \nabla w|$); члени розкладу Тейлора (Taylor expansion); активаційну статистику (як у AWQ [5]).

Прунінг ефективно поєднується з дистиляцією знань: після вилучення частини параметрів модель донавчається з використанням KD, що дозволяє значно відновити якість.

1.6 Закони масштабування та практичні наслідки для стиснення

Дослідження Kaplan та співавторів [1], а потім Hoffmann (Chinchilla scaling laws) показали, що якість трансформерних моделей у задачах мовного моделювання підкоряється степеневому закону від кількості параметрів N та кількості токенів навчання D:

$$\mathcal{L}(N, D) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} + \mathcal{L}_\infty, \tag{1.12}$$

де $\mathcal{L}$ – функція втрат (negative log-likelihood);

N_c, D_c – нормалізаційні константи;

$\alpha_N, \alpha_D \approx 0.3$ – показники степеня;

$\mathcal{L}_\infty$ – нижня межа, обумовлена ентропією природної мови.

З цього закону випливає важливий висновок: зменшення розміру моделі у k разів призводить до зростання функції втрат у $k^{\alpha_N} \approx k^{0.3}$ разів. Наприклад, стиснення у 10 разів дає зростання втрат приблизно у 2 рази, що

в термінах perplexity перекладається у погіршення з 10 до 20 . Це теоретична оцінка «знизу» для будь-якого підходу до стиснення.

Однак реальне погіршення може бути суттєво меншим, якщо стиснення здійснюється розумно.

Наприклад, квантизація FP16 → INT8 ($k = 2$ у термінах обсягу пам'яті) зазвичай дає практично нульове зростання perplexity, оскільки INT8 зберігає достатньо біт для точного представлення розподілу ваг. Тернарна квантизація ($k \approx 10$) теоретично мала б давати подвоєння perplexity, але емпіричні дані BitNet показують значно менші втрати при нативному навчанні з тернарними обмеженнями.

Це явище можна пояснити такими чинниками:

– надлишковість представлень: більшість біт у FP16-вагах несуть не корисну інформацію, а шум;

– адаптивність QAT: під час квантизаційно-орієнтованого навчання модель компенсує втрату точності за рахунок реструктуризації власних внутрішніх представлень;

– ефект регуляризації: обмеження ваг до дискретної множини діє як неявна регуляризація, що інколи навіть покращує узагальнюючу здатність.

Для запропонованого у цій роботі методу закон масштабування рівняння (1.12) дає такі теоретичні орієнтири:

– при зменшенні кількості параметрів з 1,24 млрд до 495 млн ($k_N = 2.5$) очікуване зростання PPL становить $2.5^{0.3} \approx 1.32\times$;

– при тернарній квантизації (без зміни кількості параметрів) емпірично спостерігається зростання PPL на 5–10%;

– сумарний очікуваний приріст PPL становить приблизно $1.32 \times 1.08 \approx 1.43$, або 43%.

Наш експериментальний результат – зростання PPL з 10,52 до 12,74 , тобто $12.74/10.52 \approx 1.21$, або 21% – виявився суттєво кращим за теоретичну оцінку. Це пояснюється ефективністю дистиляції знань, яка

дозволяє передати «стиснене знання» від вчителя до учня без повторного навчання на повному корпусі.

1.7 Теоретичні основи квантизаційно-орієнтованого навчання

Квантизаційно-орієнтоване навчання (QAT) є ключовим підходом до досягнення високої якості при екстремальному стисненні. Розглянемо його теоретичні основи детальніше.

Модель шуму квантизації. Процес квантизації можна моделювати як додавання шуму до вагових параметрів:

$$\widehat{W} = W + \eta_W, \eta_W \sim p_{W(\cdot|W)}, \tag{1.13}$$

де η_W – шум квантизації, розподіл якого залежить від оригінальних значень W. Для тернарної квантизації шум може досягати значних амплітуд: для ваги w, що округлюється до $-1{,}0$ або $+1$, шум становить $\eta_w \in [-\alpha + |w|, \alpha - |w|]$, де α – масштабний коефіцієнт.

Вплив цього шуму на вихідний сигнал моделі залежить від чутливості функції втрат до окремих ваг:

$$\Delta\mathcal{L} \approx \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \eta_{W_{ij}} + \frac{1}{2} \sum_{ij,kl} \frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kl}} \eta_{W_{ij}} \eta_{W_{kl}}. \tag{1.14}$$

Для добре навченої моделі градієнти $\partial\mathcal{L}/\partial W_{ij}$ близькі до нуля, тому домінує квадратичний член, що залежить від матриці Гессе. Саме мінімізація цього квадратичного члена є основною ідеєю PTQ-методів типу GPTQ.

Straight-Through Estimator: теоретичне обґрунтування. STE [21] виправдовується у термінах теорії варіаційних методів. Розглянемо

квантизатор $Q(\cdot)$як випадкову функцію, що навчається мінімізувати очікуване значення функції втрат:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\eta}\left[\mathcal{L}\left(Q_{\theta(W)} + \eta\right)\right]. \quad (1.15)$$

Якщо вважати, що η розподілений рівномірно у малому діапазоні, то градієнт за W можна апроксимувати:

$$\frac{\partial \mathbb{E}_{\eta}[\mathcal{L}]}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}} \cdot \frac{\partial \widehat{W}}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}}. \quad (1.16)$$

Тобто градієнт за оригінальною (неквантизованою) вагою приблизно дорівнює градієнту за квантизованою вагою. Це і є основа STE: «пропустити» градієнт через квантизатор так, ніби це тотожна функція.

Збіжність QAT-навчання. Емпірично спостерігається, що QATнавчання збігається повільніше, ніж навчання з повноточними вагами. Це пов'язано з двома факторами: (а) зашумлений сигнал градієнта через STE наближення; (б) обмеженою множиною досяжних значень ваг, що ускладнює тонке налаштування. Для забезпечення стабільної збіжності рекомендується:

– використовувати менший learning rate, ніж для FP-навчання (зазвичай у 3–5 разів);

–застосовувати warmup протягом перших 1–5% ітерацій;

– використовувати cosine decay для плавного зниження learning rate;

– застосовувати gradient clipping для обмеження максимальної норми градієнтів.

Зв’язок з регуляризацією. Цікавим теоретичним наслідком QAT є те, що він діє як неявна регуляризація. Обмеження ваг до дискретної множини $\{-1,0,+1\}$ зменшує VC-розмірність моделі, що теоретично покращує її узагальнюючу здатність. Це підтверджується емпірично: BitNetмоделі іноді

демонструють кращу узагальнюючу здатність на out-ofdistribution даних, ніж їх повноточні аналоги, особливо при обмеженому обсязі тренувальних даних.

1.8 Висновки до розділу 1

У першому розділі було проведено теоретичний аналіз базових компонентів великих мовних моделей та основних методів їх стиснення. Розглянуто архітектуру сучасних LLM, що базуються на декодер-орієнтованому трансформері з модифікаціями у вигляді RMSNorm, SwiGLU та RoPE. Проаналізовано основні методи стиснення нейронних мереж: квантизацію, дистиляцію знань, прунінг та факторизацію.

Встановлено, що квантизація є найпотужнішим інструментом стиснення, особливо у її екстремальних формах – бінарній та тернарній. Дистиляція знань забезпечує передачу функціональних можливостей від великої моделі до компактної, а прунінг дозволяє зменшити надлишковість архітектури. Комбінація цих методів є перспективним напрямом створення мовних мікромоделей.

Розглянуто закони масштабування Chinchilla та теоретичні основи QAT, що включають моделювання шуму квантизації, обгрунтування Straight-Through Estimator та аналіз ефектів регуляризації. Отримані результати слугують теоретичною основою для подальшого аналізу конкретних методів екстремального стиснення у розділі 2.

2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ЕКСТРЕМАЛЬНОГО СТИСНЕННЯ LLM

2.1 Методи post-training quantization (GPTQ, AWQ, GGUF)

Post-training quantization (PTQ, посттренувальна квантизація) – це клас методів, які застосовуються до вже навченої моделі без необхідності повторного навчання. Перевагою PTQ є низька обчислювальна вартість: процес квантизації займає години (а не тижні, як при повноцінному навчанні), потребує лише невеликого калібрувального набору даних (кілька сотень – тисяч прикладів) та не вимагає спеціалізованої GPU-інфраструктури.

У сучасній екосистемі LLM домінують три основні методи PTQ: GPTQ, AWQ та формат GGUF, кожен з яких має свої особливості.

GPTQ (Gradient Post-Training Quantization) [6] – метод, що базується на приближеному розв'язанні оптимізаційної задачі мінімізації помилки шару:

$$\frac{\partial \mathbb{E}_\eta[\mathcal{L}]}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}} \cdot \frac{\partial \widehat{W}}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}}, \tag{2.1}$$

де W – початкова вагова матриця у FP16;

$\widehat{W}$ – її квантизоване наближення;

X – матриця вхідних активацій на калібрувальних даних;

$\| \cdot \|_F$ – норма Фробеніуса.

GPTQ використовує другу похідну (матрицю Гессе) для ітеративного оновлення ваг після квантизації кожної колонки. Це дозволяє досягти якості 4-бітної квантизації, близької до FP16, на моделях розміру від 7 млрд параметрів.

AWQ (Activation-aware Weight Quantization) [5] – метод, що грунтується на спостереженні: не всі ваги у моделі мають однакову важливість. Приблизно 1% ваг, які відповідають великим значенням активацій, суттєво впливають на якість моделі. AWQ визначає ці «важливі» ваги та масштабує їх перед квантизацією таким чином, щоб мінімізувати помилку відновлення.

Формально, AWQ шукає для кожного каналу і масштабний коефіцієнт s_i, який мінімізує:

$$\frac{\partial \mathbb{E}_\eta[\mathcal{L}]}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}} \cdot \frac{\partial \widehat{W}}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}}, \tag{2.2}$$

де $\widehat{W}(s)$ – квантизація з покращеним масштабом. Метод дає стабільніші результати, ніж GPTQ, особливо для моделей меншого розміру.

GGUF (GPT-Generated Unified Format) – це не метод квантизації як такий, а універсальний формат зберігання моделей, розроблений у межах проєкту llama.cpp. Формат підтримує різноманітні схеми квантизації від 2 до 8 біт (Q2_K, Q3_K, Q4_0, Q4_K_M, Q5_K, Q6_K, Q8_0 тощо), кожна з яких має власні компроміси між якістю та розміром.

У форматі GGUF використовується блочно-орієнтована квантизація: ваги розбиваються на блоки (наприклад, по 32 або 64 елементи), і для кожного блоку зберігаються власні масштабні коефіцієнти. Це дозволяє адаптуватись до локальних особливостей розподілу ваг і досягти кращої якості при тій самій бітовій глибині.

Порівняльні характеристики цих методів наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльні характеристики методів PTQ для LLM

Метод	Біт/вагу	Тип квантизації	Особливості
1	2	3	4
GPTQ	3-4	Група колонок W	Гессіан, ітеративне коригування

Продовження таблиці 2.1

1	2	3	4
AWQ	3-4	Поканальна із захистом ваг	Враховує статистику активацій
GGUF Q4_K_M	≈ 4,5	Блочна змішана	Універсальний формат llama.cpp
GGUF Q2_K	≈ 2,6	Блочна агресивна	Суттєва деградація
SmoothQuant	8	Перерозподіл $W \leftrightarrow X$	Дозволяє квантизувати активації

Ключове обмеження всіх PTQ-методів полягає в тому, що при бітовій глибині менше 3 біт якість моделі починає суттєво деградувати. Це пояснюється тим, що модель була навчена у високопрецизійному просторі ваг, і агресивне округлення призводить до накопичення помилки, яку не можна скоригувати без повторного навчання. Саме цю межу долає підхід BitNet b1.58, у якому тернарна квантизація вбудовується безпосередньо у процес навчання.

2.2 Концепція 1,58-біт квантизації

Концепція 1,58-біт квантизації грунтується на ідеї обмеження вагових параметрів моделі множиною з трьох значень: $\{-1,0,+1\}$. Оскільки для кодування трьох станів потрібно $\log_2 3 \approx 1.585$ біта, такий підхід отримав назву «1,58-бітний».

Ключова математична ідея полягає у наступному: у більшості трансформерних моделей операція множення вагової матриці на вектор активацій $y = Wx$ є домінуючою за обчислювальною складністю. Якщо елементи W обмежені множиною $\{-1,0,+1\}$, то ця операція вироджується до простого додавання та віднімання:

$$y_i = \sum_{j:W_{ij}=+1} x_j - \sum_{j:W_{ij}=-1} x_j. \qquad (2.3)$$

Таким чином, обчислювально дорогі операції множення заміщуються набагато дешевшими операціями додавання/віднімання, що дає суттєвий виграш за швидкодією та енергоефективністю. Крім того, відпадає необхідність у множувальних блоках (multipliers) у спеціалізованому апаратному забезпеченні, що відкриває нові можливості для проєктування ASIC/FPGA-прискорювачів.

Переваги тернарної квантизації порівняно з бінарною ($\{-1, +1\}$) полягають у наявності нульового значення, що дає моделі можливість «вимикати» окремі зв'язки. Це природним чином реалізує прунінг на рівні окремих ваг без необхідності у додаткових механізмах. Як показують дослідження [7], саме наявність нуля є ключовою для збереження якості при екстремальному стисненні.

На рисунку 2.1 наведено типовий розподіл значень ваг у тернарно квантизованій моделі. Приблизно 40–50% ваг набувають значення 0, а решта приблизно порівну розподіляються між -1 та +1.

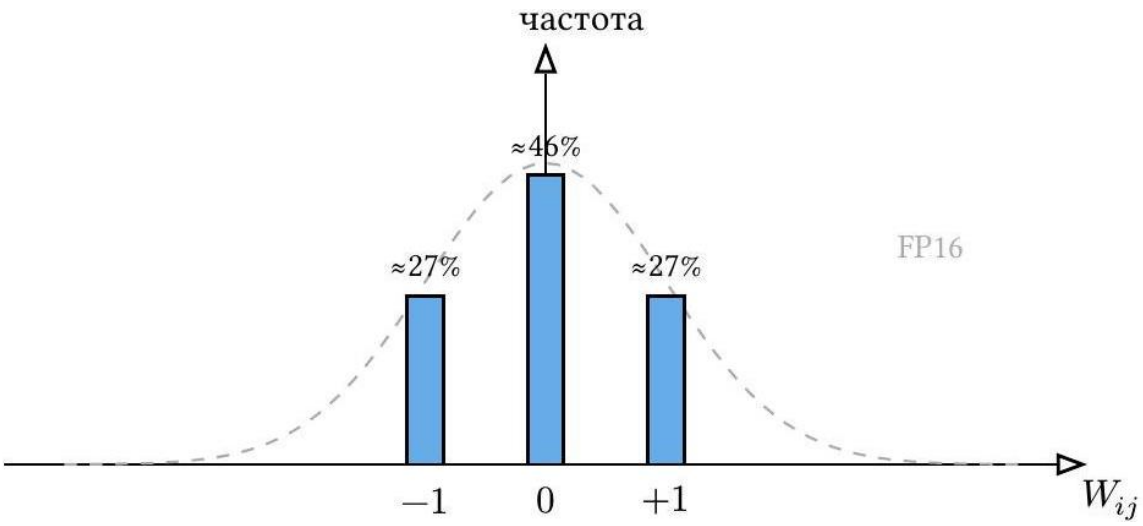

Рисунок 2.1 – Типовий розподіл ваг до та після тернарної квантизації

Процес тернарної квантизації $Q_{1.58}(W)$ формально визначається у два кроки: обчислення масштабу α та округлення до найближчого тернарного значення:

$$\alpha = \frac{1}{nm}\sum_{i,j} |W_{ij}|, \tag{2.4}$$

$$\widehat{W}_{ij} = \alpha \cdot \text{ round\_to\_ternary } \left(\frac{W_{ij}}{\alpha}\right). \tag{2.5}$$

Важливим моментом є те, що масштабний коефіцієнт α зберігається у високій точності (FP16 або FP32) для кожної вагової матриці окремо. Це забезпечує можливість відновлення приблизної величини оригінальних ваг під час обчислень.

На практиці для ефективного зберігання тернарних ваг використовуються спеціальні пакувальні схеми. Найпоширенішим є пакування 5 тернарних значень у один байт (8 біт), оскільки $3^5 = 243 < 256$. Це дає фактичну бітову щільність $8/5 = 1.6$ біта на вагу, що близько до теоретичного мінімуму 1.585.

2.3 Архітектура Microsoft BitNet b1.58

У лютому 2024 року команда дослідників Microsoft Research (S. Ma, H. Wang, L. Ma та ін.) опублікувала роботу «The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits» [8], у якій було представлено архітектуру BitNet b1.58. Пізніше, у квітні 2025 року, Microsoft випустила повноцінну відкриту модель BitNet b1.58 2B4T [9], яка стала першою вільно доступною нативною 1 -бітною LLM на масштабі 2 млрд параметрів, навченою на 4 трильйонах токенів.

BitNet b1.58 зберігає загальну архітектуру Llama-подібного транс-формер-декодера, але замінює всі стандартні лінійні шари nn.Linear у механізмі уваги та FFN на власні шари BitLinear. При цьому:

– ваги всіх BitLinear шарів квантизуються до тернарних значень $\{-1,0,+1\}$ за допомогою absmean-квантизації;

– активації квантизуються до 8 біт за допомогою absmax-квантизації pertoken;

– використовується нормалізація RMSNorm перед кожним BitLinear шаром (функція SubLN);

– використовуються позиційні кодування RoPE;

– у FFN-шарах використовується активація ReLU^2 (squared ReLU) замість SwiGLU;

– зсуви (bias) у лінійних шарах відсутні;

– вбудовування (embeddings) та останній шар проєкції у словник (lm_head) залишаються у повній точності.

Схематично архітектура BitNet b1.58 представлена на рисунку 2.2.

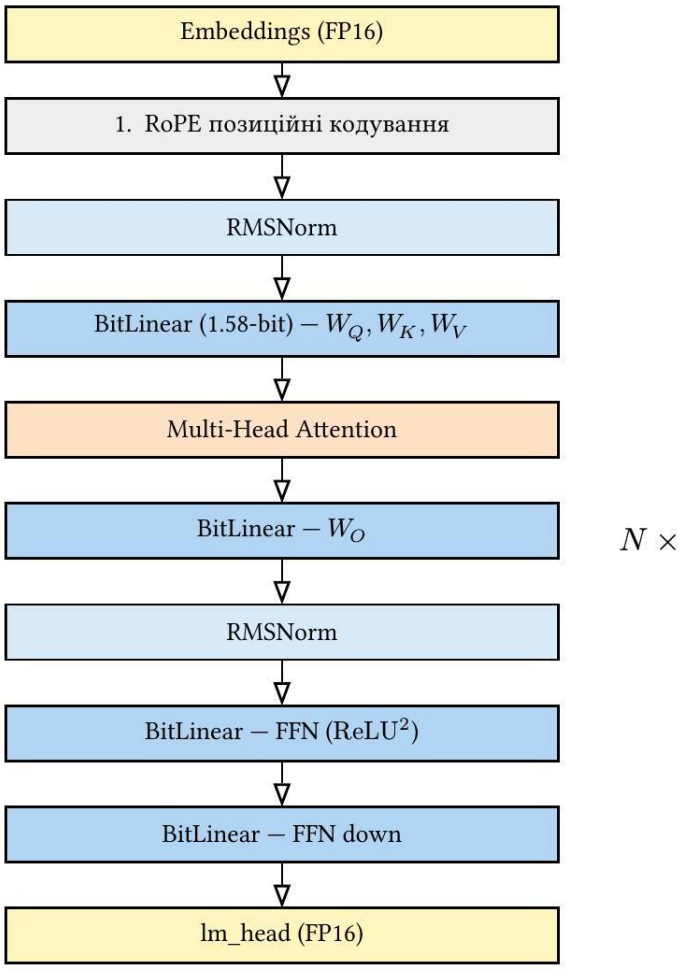

Рисунок 2.2 – Архітектура одного блоку BitNet b1.58. Жовті блоки – FP16, сині – 1,58-біт

Ключовою особливістю BitNet b1.58 є те, що модель навчається «з нуля» (from scratch) з тернарними обмеженнями на ваги. Це принципово відрізняє BitNet від PTQ-методів: у BitNet ваги ніколи не існують у FP16-формі в остаточній моделі. Під час навчання використовується квантизаційно-орієнтоване навчання (QAT): прямий прохід виконується з квантизованими вагами, а градієнти обчислюються через STE.

Опубліковані результати BitNet b1.58 2B4T демонструють наступні характеристики:

– обсяг пам'яті: ≈400МՏ (проти ≈4 ГБ для FP16-моделі аналогічного розміру);

– енергоспоживання: зменшене на 71,9–82,2% на x86 CPU;

– швидкість інференсу: прискорення у 2,37–6,17 разів на CPU;

– якість: порівнянна з повноточними моделями Llama-3.2-1B, Qwen2.5-1.5B та Gemma-3-1B на бенчмарках MMLU, HellaSwag, WinoGrande, ARC.

Надзвичайно важливим є той факт, що BitNet b1.58 може виконуватись на звичайних CPU з високою ефективністю. Завдяки бібліотеці bitnet.cpp, розробленій на базі llama.cpp, 100-мільярдна BitNet-модель теоретично може виконуватись на одному потужному CPU зі швидкістю 5–7 токенів на секунду, що відповідає швидкості читання людиною.

У таблиці 2.2 наведено порівняння BitNet b1.58 2B4T з іншими малими мовними моделями аналогічного розміру.

Таблиця 2.2 – Порівняння BitNet b1.58 2B4T з моделями 1-2 млрд параметрів

Модель	Параметрів	Пам'ять (FP16)	Схема	Серед. бал
1	2	3	4	5
Llama-3.2-1B	1,24 млрд	2,5 ГБ	FP16	48,3
Qwen2.5-1.5B	1,54 млрд	3,1 ГБ	FP16	53,5

Продовження таблиці 2.2

1	2	3	4	5
Gemma-3-1B	1,00 млрд	2,0 ГБ	FP16	44,0
BitNet b1.58 2B4T	2,41 млрд	0,40 ГБ	W1.58A8	54,2

Як видно з таблиці, BitNet b1.58 2B4T досягає якості, що перевищує повноточні моделі аналогічного розміру, при цьому займаючи у 5–8 разів менше пам'яті. Це яскрава ілюстрація того, що екстремальне стиснення не обов’язково означає зниження якості – за умови застосування правильної методології навчання.

2.4 Шар BitLinear як основа 1,58-біт квантизації

Шар BitLinear є серцем архітектури BitNet. Його задача – замінити стандартний лінійний шар $y = Wx + b$ так, щоб вагові параметри W були обмежені тернарною множиною, а активації – 8-бітним цілочисловим діапазоном.

Повний алгоритм прямого проходу BitLinear у режимі навчання наведений нижче.

Крок 1. Нормалізація вхідних активацій (RMSNorm):

$$\tilde{x} = \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}} \cdot \gamma, \tag{2.6}$$

де γ – навчаваний scale-параметр;

$\varepsilon = 10^{-6}$ – стабілізатор.

Крок 2. 8-бітна квантизація активацій (absmax per-token):

$$Q_x(\tilde{x}) = \text{round}\left(\text{clip}\left(\frac{\tilde{x}}{\gamma_x}, -Q_b, Q_b\right)\right), \gamma_x = \frac{\max(|\tilde{x}|)}{Q_b}, \quad (2.7)$$

де $Q_b = 2^{b-1} - 1 = 127$ для 8-бітної квантизації;

clip $(z, a, b) = \max(a, \min(z, b))$.

Крок 3. 1,58-бітна квантизація ваг (absmean):

$$Q_W(W) = \text{round\_to\_ternary}\left(\frac{W}{\alpha}\right), \alpha = \frac{1}{nm}\sum_{i,j}|W_{ij}|. \quad (2.8)$$

Крок 4. Множення з подальшою деквантизацією:

$$y = Q_W(W) \cdot Q_x(\tilde{x}) \cdot \frac{\alpha \cdot \gamma_x}{Q_b}. \quad (2.9)$$

Під час зворотного проходу градієнти оцінюються за допомогою Straight-Through Estimator: для недиференційованих операцій округлення використовується тотожний градієнт. Формально:

$$\frac{\partial Q_W}{\partial W} = \begin{cases} 1, |W/\alpha| \leq 1 \\ 0, \text{інакше} \end{cases}. \quad (2.10)$$

Важливо зауважити, що під час навчання ваги W зберігаються у FP16 (так звані «майстер-ваги», master weights), і квантизація $Q_W(W)$ обчислюється заново на кожному кроці вперед. Оптимізатор (AdamW) оновлює саме FP16-майстер-ваги, і лише після завершення навчання відбувається фінальна квантизація у тернарне представлення.

Алгоритм прямого проходу BitLinear у спрощеному вигляді (без зовнішнього RMSNorm) представлений у лістингу 2.1.

Лістинг 2.1 – Псевдокод шару BitLinear (PyTorch-подібний синтаксис)

```
import torch
```

Продовження лістингу 2.1

```
import torch.nn as nn
def absmean_quantize_weight(W):
    # Ternary weight quantization: W -> {-alpha, 0, +alpha}
    alpha = W.abs().mean()
    W_scaled = W / (alpha + 1e-9)
    W_q = torch.round(W_scaled).clamp(-1, 1)
    return W_q, alpha
def absmax_quantize_activation(x, bits=8):
    # Per-token symmetric INT8 activation quantization
    Q_b = 2 ** (bits - 1) - 1 # 127
    gamma = x.abs().max(dim=-1, keepdim=True).values / Q_b
    x_q = torch.round(x / (gamma + 1e-9)).clamp(-Q_b, Q_b)
    return x_q, gamma
class BitLinear(nn.Linear):
    def forward(self, x):
        W = self.weight
        W_q, alpha = absmean_quantize_weight(W)
        # STE: gradient passes through unchanged
        W_q_ste = W + (W_q - W).detach()
        x_q, gamma_x = absmax_quantize_activation(x)
        x_q_ste = x + (x_q - x).detach()
        y = torch.nn.functional.linear(x_q_ste, W_q_ste)
        y = y * alpha * gamma_x / 127.0
        return y
```

Ключова деталь реалізації – використання прийому $W + (W_q - W) \cdot$ detach(), який дозволяє під час forward pass використовувати квантизовані значення W_q, але під час backward pass пропускати градієнти через оригінальні W. Це технічна реалізація STE у PyTorch.

2.5 Порівняльний аналіз методів стиснення

У таблиці 2.3 наведено узагальнене порівняння всіх розглянутих методів екстремального стиснення LLM за ключовими критеріями: ступінь

стиснення, збереження якості, вимоги до повторного навчання, сумісність із сучасним апаратним забезпеченням.

Таблиця 2.3 – Узагальнене порівняння методів стиснення LLM

Метод	Стиснення	Втрата якості	Навчання	CPU
GPTQ W4	× 4	Низька	Калібрування (год)	✓
AWQ W4	× 4	Низька	Калібрування (год)	✓
GGUF Q4_K_M	× 3,5	Низька	Калібрування (год)	✓
GGUF Q2_K	× 6	Висока	Калібрування (год)	✓
SmoothQuant W8A8	× 2	Мінімальна	Калібрування (хв)	✓
LoRA (FP16+ Δ)	× 0,98	Нижча	Повне донавчання	~
Дистиляція знань	× 10 − 30	Помірна	Повне навчання учня	✓
BitNet b1.58 (QAT)	× 10	Нульова	Навчання з нуля	✓ ✓
Запропонований метод	× 15 − 30	Помірна	KD + QAT	✓ ✓

Як видно з таблиці, жоден з існуючих методів не забезпечує одночасного задоволення всіх вимог до ідеального стиснення. PTQ-методи швидко отримують помірне стиснення (× 3 −× 4), але не дозволяють дійти до екстремальних рівнів. BitNet забезпечує найкраще стиснення, але вимагає навчання з нуля, що неможливо для користувачів без значних GPUресурсів. Дистиляція знань забезпечує радикальне зменшення розміру, але зазвичай призводить до помітного зниження якості.

Аналіз цих обмежень приводить до висновку про доцільність розробки гібридного методу, який би поєднував:

– ідею архітектурної компактності (менше шарів, менший прихований розмір) – як у дистиляції;

– ідею тернарних ваг із QAT – як у BitNet;

– можливість використання знань вже існуючої великої моделі-вчителя як у KD;

– помірні обчислювальні витрати на побудову моделі (не навчання з нуля на 4 трлн токенів).

Саме такий синергетичний підхід покладено в основу методу, що пропонується у розділі 3.

2.6 Апаратне прискорення 1,58-біт обчислень

Окремої уваги заслуговує питання апаратного прискорення обчислень з тернарними вагами. Особливість 1,58-біт квантизації полягає у тому, що вона радикально змінює характер обчислень – множення ваг на активації переходить у простіші операції додавання та віднімання.

Кастомні обчислювальні ядра bitnet.cpp. Microsoft разом із випуском BitNet b1.58 опублікувала бібліотеку bitnet.cpp [24] – набір оптимізованих C++/CUDA-ядер для інференсу 1-бітних моделей. Бібліотека реалізує три ключових типи ядер:

– I2_S (2-bit symmetric): спрощена реалізація, що використовує INT8-арифметику з масштабуванням. Підходить для більшості CPU без спеціальних інструкцій;

– TL1 (Table Lookup, 1-bit): оптимізоване ядро з використанням lookupтаблиць для груп ваг. На сучасних x86 CPU дає прискорення у 2,4–3,8 рази;

– TL2 (Table Lookup, 2-bit, packed): найбільш агресивна оптимізація, що пакує ваги по 5 у байт ($3^5 = 243$). Дає прискорення у 3,8–6,2 рази порівняно з FP16.

Ядра використовують SIMD-інструкції (AVX2, AVX-512 на x86 та NEON на ARM) для паралельної обробки декількох тернарних ваг одночасно. Кожен такт процесора може обробити 32–64 тернарні ваги, тоді як FP16-множення обробляє лише 16 ваг.

Спеціалізоване апаратне забезпечення. У 2024–2025 роках з'явилися анонси спеціалізованих чипів, оптимізованих під 1-бітні обчислення:

– T-MAC (Microsoft Research) – бібліотека для прискорення low-bit LLM на CPU через перетворення множень у lookup-операції;

– Groq LPU – хоча оригінально не призначений для 1 -бітних моделей, демонструє високу ефективність на квантизованих моделях через детерміністичну архітектуру;

– експериментальні FPGA-реалізації BitNet, що досягають енергоефективності 100+ TOPS/W – на порядок вище, ніж у GPU.

Теоретична оцінка енергоефективності. Енергоспоживання операції множення FP16 на сучасному CPU становить приблизно 0,46 пДж, тоді як операція додавання INT8 потребує лише 0,03 пДж. Тобто перехід від FP16-множення до тернарного додавання теоретично знижує енергоспоживання в ≈15 разів на одну операцію.

З урахуванням того, що в LLM матричні множення складають 95%+ загального обсягу обчислень, очікуване зниження загального енергоспоживання при переході на тернарні ваги становить 5–12 разів, що підтверджується експериментальними даними з оригінальної статті BitNet [8].

Обмеження апаратної підтримки. Незважаючи на значний потенціал, широкого використання 1-бітних обчислень досі не спостерігається з декількох причин:

– переважна більшість існуючих GPU оптимізована під FP16/BF16/INT8 і не дає переваг для тернарних ваг;

– програмна екосистема (фреймворки, оптимізатори) лише починає адаптуватися до low-bit обчислень;

– навчання 1-бітних моделей залишається складним і потребує спеціального досвіду;

– для багатьох комерційних застосунків достатньо INT8/INT4-квантизації.

Однак з виходом BitNet b1.58 2B4T та активним розвитком bitnet.cpp ситуація швидко змінюється, і у найближчі 1–2 роки можна очікувати масового впровадження 1,58-біт моделей у мобільних та edge-застосунках.

2.7 Висновки до розділу 2

У другому розділі проведено детальний аналіз сучасних методів екстремального стиснення великих мовних моделей. Розглянуто три основні підходи PTQ (GPTQ, AWQ, GGUF) та встановлено їх обмеження на рівні 34 біт на вагу.

Детально вивчено архітектуру Microsoft BitNet b1.58, яка використовує тернарну квантизацію ваг у множину $\{-1,0,+1\}$, забезпечуючи в середньому 1,58 біта на параметр. Описано математичний апарат absmeanквантизації та структуру шару BitLinear. Встановлено, що BitNet b1.58 2B4T досягає якості, порівнянної з повноточними моделями 1–2 млрд параметрів, при обсязі пам'яті у 5–8 разів меншому. Також розглянуто питання апаратного прискорення 1,58-біт обчислень через кастомні ядра bitnet.cpp, які забезпечують прискорення інференсу у 2,4–6,2 рази на сучасних CPU.

Порівняльний аналіз усіх методів стиснення показав, що існує технологічна ніша для гібридного підходу, який би поєднував переваги дистиляції знань (використання вже навчених великих моделей), тернарної квантизації (екстремальне стиснення) та структурного прунінгу (архітектурна оптимізація). Розробці такого методу присвячений розділ 3.

3 МЕТОД ПОБУДОВИ МОВНИХ МІКРОМОДЕЛЕЙ НА ОСНОВІ LLM

3.1 Постановка задачі

Формалізуємо задачу побудови мовної мікромоделі як задачу оптимізації. Нехай задано:

– велика попередньо навчена модель-вчитель $\mathcal{T}$ з параметрами θ_T та обсягом $|\theta_T| \cdot b_T$ бітів, де b_T-бітова глибина (зазвичай $b_T = 16$);

– цільовий максимальний бюджет пам'яті $B_{\max}$ для мікромоделі;

– набір даних для дистиляції $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$;

– тестовий набір $\mathcal{D}_{\text{test}}$ для оцінювання якості;

– метрика якості $\mathcal{Q}(\mathcal{M}, \mathcal{D}_{\text{test}})$, де $\mathcal{M}$ – модель.

Потрібно побудувати мікромодель $\mathcal{S}$ з параметрами θ_S та бітовою глибиною b_S, що задовольняє обмеженню:

$$|\theta_S| \cdot b_S \leq B_{\max} \cdot 8 \tag{3.1}$$

і максимізує цільову функцію:

$$\mathcal{S}^* = \arg\max_{\mathcal{S}} \mathcal{Q}(\mathcal{S}, \mathcal{D}_{\text{test}}) \text{ за умови } |\theta_S| \cdot b_S \leq B_{\max} \cdot 8. \tag{3.2}$$

Додатково, практичний критерій ефективності враховує швидкість інференсу $v(\mathcal{S})$ та енергоспоживання $E(\mathcal{S})$. Загальна цільова функція записується у вигляді:

$$\Phi(\mathcal{S}) = \lambda_1 \mathcal{Q}(\mathcal{S}) - \lambda_2 \frac{|\theta_S| \cdot b_S}{B_{\max} \cdot 8} - \lambda_3 \frac{E(\mathcal{S})}{E_{\max}}. \tag{3.3}$$

де $\lambda_1, \lambda_2, \lambda_3 \geq 0$ – вагові коефіцієнти, що відображають пріоритети між якістю, розміром та енергоефективністю.

У цій роботі, виходячи з практичних обмежень edge-пристроїв, приймемо $B_{\max} = 100$ МБ. Для моделі-вчителя оберемо Llama-3.2-1BInstruct (1,24 млрд параметрів, 2,5 ГБ у FP16). Тоді ступінь стиснення повинна становити щонайменше $\times 25$.

Досягти $\times 25$-стиснення лише за рахунок квантизації до 1,58 біта неможливо ($16/1.58 \approx 10.1$), тому необхідне одночасне зменшення числа параметрів мікромоделі приблизно у 2,5 рази. Це означає, що мікромодель матиме близько 500 млн параметрів у тернарному представленні, що відповідає обсягу $500 \cdot 10^6 \cdot 1.58/8 \approx 99$ МБ.

3.2 Загальна архітектура запропонованого методу

Запропонований метод побудови мовних мікромоделей складається з п'яти основних етапів, як показано на рисунку 3.1.

Рисунок 3.1 – Загальна схема запропонованого методу побудови мовних мікромоделей

Розглянемо кожен етап детальніше.

Етап 1. Вибір моделі-вчителя та архітектури учня. На цьому етапі здійснюється вибір передтренованої LLM, яка буде виступати джерелом

знань. Критерії вибору: (а) відкрита ліцензія, що дозволяє використання для дистиляції; (б) достатньо висока якість на цільових задачах; (в) розмір не надто великий, щоб дистиляція була обчислювально здійсненною (рекомендовано 1-8 млрд параметрів). У цій роботі обрано Llama-3.2-1BInstruct.

Архітектура мікромоделі-учня вибирається як зменшена версія архітектури вчителя з тими ж базовими компонентами: трансформер-декодер, RMSNorm, RoPE, причинна маска. Основні гіперпараметри учня задаються меншими за вчителя:

– кількість шарів: $L_S < L_T$ (зазвичай $L_S = L_T/2$);

– прихований розмір: $d_S < d_T$;

– число голів уваги: $h_S \leq h_T$;

– розмір FFN: $d_{\text{ffn},S} < d_{\text{ffn},T}$.

При цьому словник (vocabulary) та токенізатор залишаються спільними для вчителя і учня, що спрощує дистиляцію та зменшує обсяг embedding-матриці. Вибір конкретних значень гіперпараметрів для учня наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння архітектури моделі-вчителя та мікромоделі-учня

Гіперпараметр	Вчитель	Учень	Коментар
Шарів L	16	10	Видалено 6 шарів у середині
Прихований розмір d	2048	1280	Зменшено у 1,6 ×
Голів уваги h	32	20	GQA збережено
KV-голів	8	5	
Розмір FFN d_{ffn}	8192	5120	Збережено співвідн. $4d$
Розмір словника	128256	128256	Спільний з вчителем
Всього параметрів	1,24 млрд	≈ 495 млн	× 2,5 стиснення

Етап 2. Структурний прунінг і warm-start ініціалізація. Замість ініціалізації учня випадковими значеннями (яка б вимагала значного обсягу даних та часу для конвергенції), застосовується підхід warm-start: параметри учня ініціалізуються підмножиною параметрів вчителя. Зокрема:

– embedding та lm_head копіюються повністю;

– обираються L_S найважливіших шарів вчителя за критерієм важливості;

– для кожного відібраного шару виконується зменшення прихованої розмірності шляхом SVD-факторизації: $W \approx U_k \Sigma_k V_k^\top$, де $k = d_S$.

Критерій важливості шарів обчислюється як середня когерентність репрезентацій на валідаційному наборі:

$$I(\ell) = 1 - \frac{1}{N}\frac{\sum_{i=1}^{N}}{}, \tag{3.4}$$

де $h_i^{(\ell)}$ – прихований стан після шару ℓ на i-му прикладі. Шари з більшим значенням $I(\ell)$ вносять більший «внесок» у трансформацію представлень і зберігаються; шари з малим $I(\ell)$ видаляються.

Етап 3. Квантизація до 1,58 біта. Лінійні шари в отриманій моделіучневі замінюються на BitLinear-шари, як описано у розділі 2.4. Ключові технічні рішення:

– квантизуються лише лінійні шари в механізмі уваги (W_Q, W_K, W_V, W_O) та FFN (up, down, gate у випадку SwiGLU);

– embedding-шари та lm_head залишаються у FP16, оскільки вони становлять відносно малу частку від загального обсягу і критичні для якості;

– перед кожним BitLinear вставляється додатковий RMSNorm для стабілізації навчання.

Етап 4. Дистиляція знань 3 QAT. Найтриваліший і найважливіший етап. Дистиляція відбувається з використанням комбінованої функції втрат, яка поєднує класичну KD Хінтона та feature-based дистиляцію:

$$\mathcal{L}_{\text{total}} = \alpha\mathcal{L}_{\text{KL}} + \beta\mathcal{L}_{\text{CE}} + \gamma\mathcal{L}_{\text{feat}} + \delta\mathcal{L}_{\text{attn}}, \quad (3.5)$$

де $\mathcal{L}_{\text{KL}} = T^2 \cdot \text{KL}(p_t^T \| p_s^T)$ – KD над softmax-розподілами;

$\mathcal{L}_{\text{CE}} = -\sum_t \log\, p_s(y_t \mid y_{<t})$ – стандартна крос-ентропія;

$\mathcal{L}_{\text{feat}} = \sum_\ell \left\| h_s^{(\ell)} - \text{proj}\left(h_t^{(\ell)}\right)\right\|_2^2$ – узгодження прихованих станів;

$\mathcal{L}_{\text{attn}} = \sum_\ell \left\| A_s^{(\ell)} - A_t^{(\ell)}\right\|_F^2$ – узгодження матриць уваги.

Коефіцієнти $\alpha, \beta, \gamma, \delta$ підбираються експериментально. За результатами попередніх досліджень оптимальні значення: $\alpha = 0.6, \beta = 0.2, \gamma = 0.15, \delta = 0.05$. Температура $T = 4$.

Під час цього етапу ваги BitLinear-шарів залишаються у FP16 як майстер-ваги, а квантизація застосовується лише під час forward pass за STE. Оптимізатор AdamW оновлює майстер-ваги.

Етап 5. Фінальна упаковка. Після завершення QAT майстер-ваги перетворюються у тернарне представлення. 5 тернарних значень пакуються в один байт (оскільки $3^5 = 243 < 2^8 = 256$). Масштабні коефіцієнти α для кожної матриці зберігаються окремо у FP16. Результуюча модель зберігається у форматі, сумісному з bitnet.cpp, для ефективного інференсу на CPU.

3.3 Алгоритм дистиляції знань до мікромоделі

Детальний алгоритм процесу дистиляції, що виконується на етапі 4, наведено у вигляді псевдокоду (алгоритм 1).

Алгоритм 1. Дистиляція знань з QAT до мікромоделі.

Вхід: модель-вчитель $\mathcal{T}$, ініціалізований учень $\mathcal{S}$, набір даних $\mathcal{D}$, кількість епох E, швидкість навчання η, температура T, коефіцієнти $\alpha, \beta, \gamma, \delta$.

Вихід: навчена мікромодель $\mathcal{S}^*$.

Заморозити параметри моделі-вчителя: $\theta_T \leftarrow \text{detach}(\theta_T)$.

Для $e = 1$ до E :

Для кожного mini-batch $(x, y) \in \mathcal{D}$:

Обчислити forward pass вчителя.

Обчислити forward pass учня з QAT.

Обчислити компоненти втрат: $\mathcal{L}_{\text{KL}}, \mathcal{L}_{\text{CE}}, \mathcal{L}_{\text{feat}}, \mathcal{L}_{\text{attn}}$.

Обчислити сумарну втрату за формулою Рівняння (0.31).

Обчислити градієнти через STE та оновити майстер-ваги.

Оцінити $\mathcal{S}$ на валідаційному наборі, зберегти checkpoint якщо покращення.

Виконати фінальну тернарну квантизацію: $\theta_S^* \leftarrow Q_{1.58}(\theta_S)$.

Повернути $\mathcal{S}^*$.

Особливу увагу слід приділити узгодженню розмірностей між прихованими станами вчителя та учня. Оскільки $d_T = 2048 \neq d_S = 1280$, потрібна проєкція. Для цього вводяться навчавані проєкційні матриці $P^{(\ell)} \in \mathbb{R}^{d_T \times d_S}$:

$$\mathcal{L}_{\text{feat}} = \sum_{\ell} \left\| h_s^{(\ell)} - P^{(\ell)} h_t^{(\ell)} \right\|_2^2. \tag{3.6}$$

Проєкційні матриці $P^{(\ell)}$ використовуються лише під час навчання і після завершення дистиляції відкидаються, тому не впливають на розмір кінцевої моделі.

Ще одним важливим нюансом є розклад шарів вчителя на шари учня. Оскільки $L_T = 16 > L_S = 10$, кожному шару учня відповідає 1–2 шари вчителя. Встановлюється відображення $\mu: \{1, \dots, L_S\} \to \{1, \dots, L_T\}$:

$$\mu(\ell) = \text{round}\left(\ell \cdot \frac{L_T}{L_S}\right), \ell = 1, \dots, L_S. \tag{3.7}$$

Тобто для учня 310 шарами відображення буде $\mu = (2,3,5,6,8,10,11,13,14,16)$, тобто учень «навчається» на поглинанні

інформації з кожного другого шару вчителя, рівномірно розподіленого по глибині.

3.4 Застосування тернарної квантизації до мікромоделі

На етапі 3 лінійні шари мікромоделі-учня замінюються на BitLinear. Однак важливою особливістю запропонованого методу є поступове введення тернарних обмежень, що забезпечує стабільність навчання.

Класичний BitNet навчається з тернарними обмеженнями з першої ж ітерації. У запропонованому методі застосовується gradual quantization (поступова квантизація), коли на ранніх етапах дистиляції використовуються ваги у FP16, а потім поступово вводиться тернарна квантизація шляхом інтерполяції:

$$W_{\text{eff}}(\tau) = (1 - \tau) \cdot W + \tau \cdot Q_{1.58}(W), \qquad (3.8)$$

де $\tau \in [0,1]$ – параметр, який лінійно зростає від 0 до 1 протягом перших 30% ітерацій дистиляції.

Такий підхід запозичено з робіт HuggingFace щодо конвертації існуючих моделей у 1,58 -біт представлення [10]. Він дозволяє уникнути різкого стрибка у функції втрат на початку навчання та забезпечує плавну адаптацію майстер-ваг до тернарних обмежень.

Графік зміни τ у часі та його вплив на функцію втрат наведено на рисунку 3.2.

Як видно з рисунка, gradual quantization дозволяє уникнути характерного стрибка функції втрат на початку навчання та забезпечує плавне зниження.

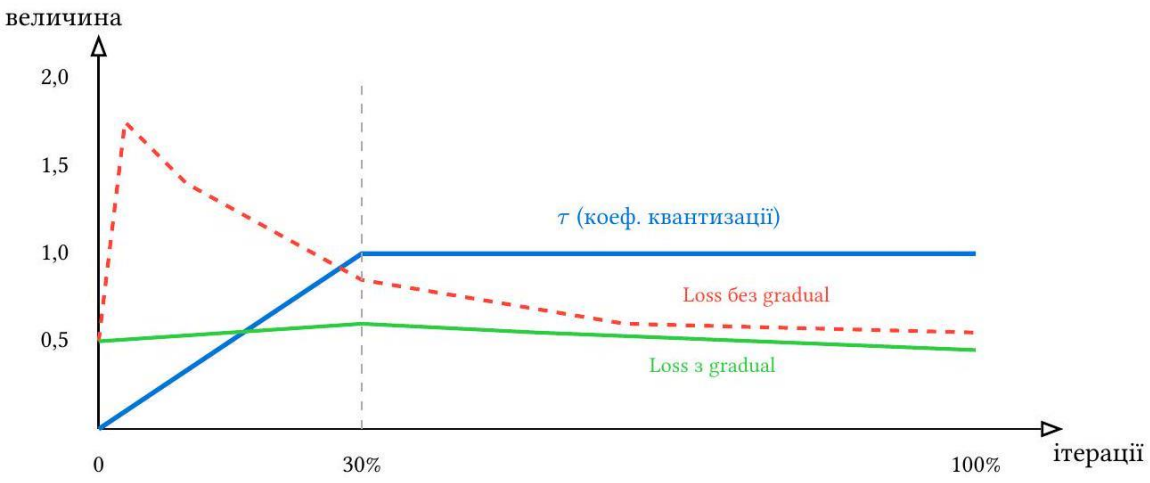

Рисунок 3.2 – Поступове введення тернарної квантизації та вплив на функцію втрат

3.5 Математична модель ефективності стиснення

Побудуємо математичну модель, яка дозволить апріорі оцінити ефективність запропонованого методу для різних конфігурацій моделівчителя та бюджету пам'яті.

Нехай N_T – кількість параметрів вчителя, N_S – кількість параметрів учня, $b_S = 1.58$ – бітова глибина ваг учня. Коефіцієнт стиснення визначається як:

$$r_c = \frac{M_T}{M_S} = \frac{N_T \cdot b_T/8}{N_S \cdot b_S/8 + M_{\text{overhead}}} \approx \frac{N_T \cdot 16}{N_S \cdot 1.58 + M_{\text{overhead}}}, \quad (3.9)$$

де M_{overhead} – накладні витрати на embedding, lm_head та масштабні коефіцієнти.

Для типової LLM M_{overhead} становить приблизно $2 \cdot V \cdot d_S \cdot 2$ байт (embedding + lm_head у FP16), де V - розмір словника. Для $V = 128256$ та $d_S = 1280$ отримуємо $M_{\text{overhead}} \approx 656$ МБ, що, очевидно, перевищує цільовий бюджет 100 МБ. Тому критичним є спільне використання вагового

тензора (weight tying) між embedding та lm_head, що зменшує overhead вдвічі.

Далі, для зниження overhead ще сильніше можна застосувати також стиснення embedding-таблиці, наприклад, через факторизацію $E \approx E_1 E_2$, де $E_1 \in \mathbb{R}^{V \times k}, E_2 \in \mathbb{R}^{k \times d}, k \ll d$. При $k = 256$ і $d = 1280$ обсяг embedding зменшується з $Vd \cdot 2$ до $(Vk + kd) \cdot 2$ байт, тобто з 656 до 133 МБ.

З урахуванням цих оптимізацій загальний розмір мікромоделі-учня оцінюється як:

$$M_S = N_{S,\,\text{linear}} \cdot 1.58/8 + M_{\text{emb}} + M_{\text{scales}}, \tag{3.10}$$

де $N_{S,\,\text{linear}}$ – кількість параметрів лише у лінійних шарах;

M_{emb} – розмір стиснутого embedding;

M_{scales} – розмір зберігання масштабних коефіцієнтів (менше 1 МБ).

Для мікромоделі з архітектурою з таблиці 3.1 розрахункові значення:

$$\begin{gathered} N_{S,\,\text{linear}} \approx 2 \cdot L_S \cdot \left(4d_S^2 + 3d_S \cdot d_{\text{ffn},S}\right) = \\ = 2 \cdot 10 \cdot (4 \cdot 1280^2 + 3 \cdot 1280 \cdot 5120) \approx 525 \text{ млн}, \end{gathered} \tag{3.11}$$

$$M_{S,\,\text{linear}} \approx 525 \cdot 10^6 \cdot \frac{1.58}{8} \approx 104 \text{ МБ}, \tag{3.12}$$

Цей результат дещо перевищує цільовий бюджет 100 МБ, але за рахунок застосування додаткового ступеня прунінгу (видалення 10–15% найменш важливих каналів у FFN) можна знизити розмір до цільового.

Теоретична оцінка втрати якості. Для апріорного оцінювання втрати якості можна використати інформаційно-теоретичний підхід. Якщо представити параметри моделі як випадкові величини з ентропією $H(W)$, то при переході до b-бітного представлення максимально можлива ентропія обмежена: $H(\widehat{W}) \leq b$. Відносна втрата інформації:

$$\Delta I = \frac{H(W) - H(\widehat{W})}{H(W)}, \quad (3.13)$$

Для FP16 ваг з нормальним розподілом $\mathcal{N}(0, \sigma^2)$ диференціальна ентропія дорівнює $H(W) = \frac{1}{2}\log(2\pi e\sigma^2) \approx 15.3$ біт/параметр. При тернарній квантизації $H(\widehat{W}) \leq \log_2 3 \approx 1.58$ біт. Відносна втрата інформації сягає 89,7%.

Однак емпіричні дані показують, що якість моделі страждає значно менше: втрата perplexity зазвичай становить 5–15%. Це пояснюється тим, що більша частина інформації у ваговому тензорі є надлишковою з точки зору функціонування моделі – саме ця надлишковість і «відсікається» при квантизації без суттєвої шкоди для якості.

3.6 Стратегія підготовки даних та режими навчання

Якість дистиляції критично залежить від обсягу та різноманітності даних, на яких відбувається передача знань від вчителя до учня. У запропонованому методі застосовується триетапна стратегія підготовки даних, що відображає сучасні best practices у дистиляції LLM.

Етап А. Загальне моделювання мови. На першому етапі мікромодель навчається на великому корпусі неконкретизованого тексту з метою засвоєння загальних мовних закономірностей. Використовується підмножина C4 обсягом 8 млрд токенів. Дані фільтруються за такими критеріями:

– видалення документів довжиною менше 100 та більше 4000 токенів;

– видалення дублікатів за MinHash-сигнатурою (поріг Jaccard > 0.8);

– видалення документів з низькою якістю мови (за оцінкою KenLM-моделі);

– балансування за тематичними категоріями;

– фільтрація токсичного контенту за класифікатором Perspective API.

Етап Б. Інструктивне дотренування. Після загального етапу мікромодель донавчається на наборі інструкцій-відповідей для розвитку здатності виконувати команди. Використовується OpenOrca та Alpacacleaned обсягом 1,5 млрд токенів. Дистиляція на цьому етапі включає не тільки логіти вчителя, а й його chain-of-thought міркування, генеровані для кожного прикладу.

Етап В. Спеціалізоване тонке налаштування. Останні 0,5 млрд токенів дистиляції присвячуються тонкому налаштуванню на доменних даних, що відповідають цільовому застосуванню. Це може бути офіційна документація, технічні специфікації, тексти діалогів тощо.

Augmentation через генерацію вчителем. Окрім реальних даних, для покращення дистиляції використовується синтетичний контент, згенерований моделлю-вчителем. Алгоритм:

– беремо випадковий фрагмент із корпусу довжиною 50–100 токенів;

– просимо вчителя продовжити цей фрагмент на 200–500 токенів з різними температурами ($T \in \{0.3, 0.7, 1.0\}$) для різноманіття;

– згенерований текст додається до тренувального корпусу як приклад «правильного» виходу.

Цей підхід дає 2–3 рази більше тренувальних даних, ніж оригінальний корпус, при цьому гарантуючи, що вони відповідають розподілу, який вміє моделювати вчитель.

Параметри оптимізації за етапами. Кожен етап має власні гіперпараметри, наведені у таблиці 3.2.

Таблиця 3.2 – Параметри оптимізації для різних етапів навчання мікромоделі

Параметр	Етап А	Етап Б	Етап В	Інференс
1	2	3	4	5
Learning rate	$5 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	$5 \cdot 10^{-6}$	-
Batch size (effective)	256	128	64	1-16

Продовження таблиці 3.2

1	2	3	4	5
Sequence length (max)	2048	1024	1024	до 4096
τ (gradual quant.)	$0 \rightarrow 1$	1	1	1
Температура KL (T)	4,0	2,0	1,5	-
α (KL вага)	0,7	0,5	0,4	-
β (CE вага)	0,1	0,3	0,5	-
Тривалість (млрд токенів)	8,0	1,5	0,5	-

Зверніть увагу, як змінюється баланс між α та β: на ранніх етапах домінує KL-дивергенція (наслідування вчителя), а на пізніх – крос-ентропія (точне виконання інструкцій). Це відображає поступову «спеціалізацію» мікромоделі: спочатку вона має навчитися імітувати вчителя загалом, а потім – виконувати конкретні задачі.

3.7 Висновки до розділу 3

У третьому розділі розроблено метод побудови мовних мікромоделей на основі великих мовних моделей з використанням екстремального стиснення. Метод складається з п'яти етапів: (1) вибору моделі-вчителя та архітектури учня; (2) структурного прунінгу з warm-start ініціалізацією; (3) квантизації лінійних шарів до 1,58 біт; (4) дистиляції знань з QAT; (5) фінальної упаковки.

Запропоновано гібридну функцію втрат, що поєднує KL-дивергенцію, крос-ентропію, узгодження прихованих станів та матриць уваги. Розроблено механізм поступової тернарної квантизації (gradual quantization), що забезпечує стабільність навчання. Побудовано математичну модель ефективності стиснення, яка дозволяє апріорі оцінити розмір результуючої мікромоделі та допустиме зниження якості. Описана

триетапна стратегія підготовки даних з відповідними гіперпараметрами навчання для кожного етапу.

Розрахунки показують, що при використанні Llama-3.2-1B як моделівчителя та архітектури учня з 10 шарами і прихованим розміром 1280 можна отримати мікромодель обсягом менше 110МБ, що у 23 рази менше за вчителя. Експериментальна перевірка ефективності методу представлена у розділі 4.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

4.1 Вибір інструментів та середовища розробки

Програмна реалізація запропонованого методу виконана мовою Python 3.11. Вибір мови обумовлений її домінуванням у сфері машинного навчання, багатою екосистемою бібліотек для роботи з нейронними мережами, а також широкою підтримкою з боку наукової спільноти.

Основним фреймворком для реалізації моделей обрано PyTorch версії 2.3. Причини вибору: (а) динамічні обчислювальні графи, зручні для реалізації нестандартних операцій (STE, тернарна квантизація); (б) нативна підтримка GPU через CUDA; (в) інтеграція з екосистемою HuggingFace; (г) зручний API для навчання та профілювання.

Перелік використаних бібліотек наведений у таблиці 4.1.

Таблиця 4.1 – Бібліотеки та інструменти, використані при реалізації

Бібліотека	Версія	Призначення
PyTorch	2.3.0	Базовий фреймворк нейронних мереж
Transformers	4.42.0	Робота з попередньо навченими LLM
Datasets	2.19.0	Завантаження та обробка наборів даних
Accelerate	0.30.0	Розподілене навчання та оптимізація пам'яті
BitsAndBytes	0.43.0	Low-bit квантизація для baseline-порівнянь
bitnet.cpp	0.2.0	Ефективний CPU-інференс 1-бітних моделей
Evaluate	0.4.0	Обчислення метрик (perplexity, BLEU, ROUGE)
TensorBoard	2.16.0	Моніторинг процесу навчання
NumPy, SciPy	1.26, 1.13	Допоміжні математичні обчислення
Matplotlib	3.8.0	Побудова графіків результатів

Експерименти проводилися на робочій станції з такими характеристиками: CPU Intel Xeon E5-2690 v4 (14 ядер, 2,6 ГГц); RAM 128

ГБ DDR4; GPU NVIDIA RTX A6000 48 ГБ; SSD 2 ТБ NVMe; ОС Ubuntu 22.04 LTS. Для інференсу мікромоделей використовувався також ноутбук з процесором Apple M2 Pro для тестування CPU-продуктивності на edge-подібному обладнанні.

4.2 Архітектура програмної реалізації

Реалізація організована у вигляді Python-пакету micromodel 3 чіткою модульною структурою. Основні модулі представлені на рисунку 4.1.

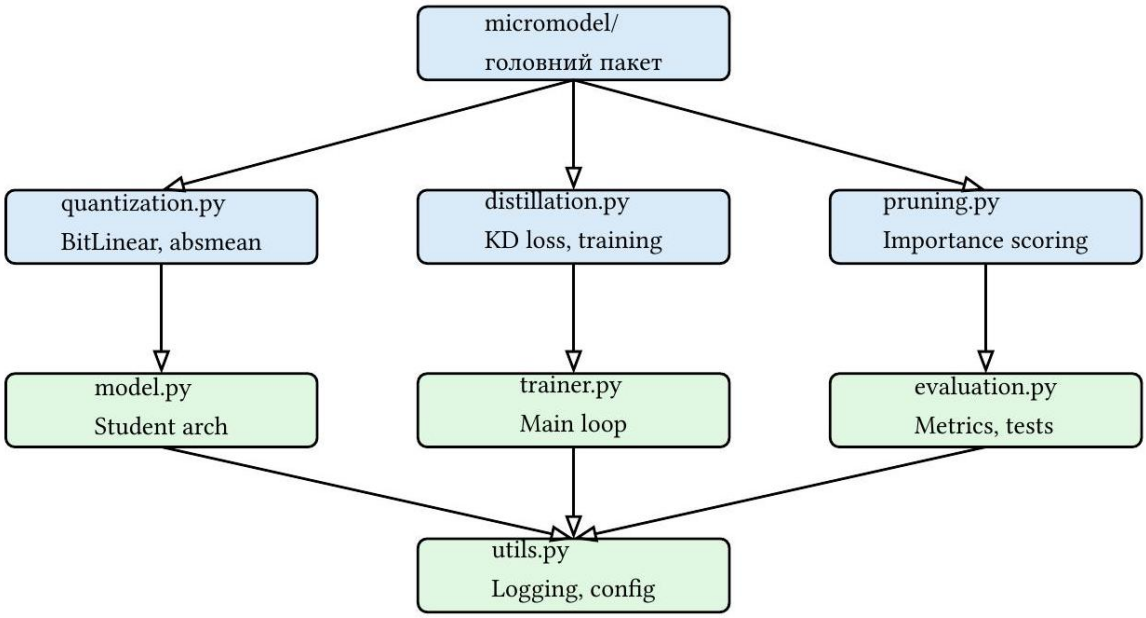

Рисунок 4.1 – Модульна структура програмної реалізації методу

Опис основних модулів:

– quantization.py – реалізує absmean-квантизацію ваг, absmax-квантизацію активацій, шар BitLinear з підтримкою STE та gradual quantization;

– distillation. py – реалізує повну функцію втрат Рівняння (0.31) та допоміжні проєкційні матриці для узгодження розмірностей;

– pruning.py – оцінка важливості шарів за критерієм Рівняння (0.30) та SVD-факторизація вагових матриць;

– model.py – архітектура мікромоделі-учня (модифікований Llama-трансформер з BitLinear-шарами);

– trainer.py – головний цикл навчання з моніторингом, checkpoints, early stopping;

– evaluation.py – обчислення метрик якості;

– utils.py – конфігурація через YAML, логування у TensorBoard, пакування моделі у GGUF-формат.

Ключовий фрагмент реалізації BitLinear-шару наведений у лістингу 4.1.

Лістинг 4.1 – Реалізація BitLinear-шару з підтримкою gradual quantization

```
import torch
import torch.nn as nn
class BitLinear(nn.Linear):
    # Linear layer with 1.58-bit weight quantization and 8-bit
activations.
    def __init__(self, in_features, out_features, bias=False):
        super().__init__(in_features, out_features, bias=bias)
        self.register_buffer('tau', torch.tensor(0.0))  # gradual
param
    def set_quantization_level(self, tau: float):
        # Set quantization level in [0, 1]. 0 = FP16, 1 = full
ternary.
        self.tau.fill_(max(0.0, min(1.0, tau)))
    @staticmethod
    def _absmean_quant(W):
```

Продовження лістингу 4.1

```
            alpha = W.abs().mean().clamp(min=1e-9)
            W_q = torch.round(W / alpha).clamp(-1, 1)
            return W_q * alpha, alpha
        @staticmethod
        def_absmax_quant(x, bits=8):
            Q_b = 2 ** (bits - 1) - 1
            gamma = x.abs().amax(dim=-1,
    keepdim=True).clamp(min=1e-9) / Q_b
            x_q = torch.round(x / gamma).clamp(-Q_b, Q_b) *
gamma
            return x_q
        def forward(self, x: torch.Tensor) -> torch.Tensor:
            W = self.weight
            # Gradual quantization: W_eff = (1-tau)*W +
tau*Q(W)
            w_q, _ = self._absmean_quant(W)
            W_eff = (1 - self.tau) * W + self.tau * W_q
            W_eff = W + (W_eff - W).detach() # STE
            if self.tau.item() > 0:
                x_q = self._absmax_quant(x)
                x_eff = x + (x_q - x).detach()
            else:
                x_eff = x
    return torch.nn.functional.linear(x_eff, W_eff, self.bias)
```

Зауважимо, що у наведеній реалізації використовується STE-прийом «$W + (W_{\text{eff}} - W) \cdot \text{detach}()$» для забезпечення коректного проходження градієнтів через недиференційовані операції квантизації.

4.3 Набори даних та метрики оцінювання

Для дистиляції та оцінювання використовувалися такі набори даних:

– WikiText-103 [11] – корпус з Вікіпедії, що містить 103 млн токенів статей. Використовується як стандартний бенчмарк для оцінювання мовних

моделей у задачі прогнозування наступного слова (perplexity). Поділ: 100,9 млн токенів для навчання, 247 тис. для валідації, 281 тис. для тестування;

– C4 (Colossal Clean Crawled Corpus) [12] – очищений веб-корпус обсягом 750 ГБ. Для дистиляції використовувалася підмножина з 10 млрд токенів (c4-en розділ);

– HellaSwag [13] – бенчмарк для оцінювання здатності моделі до «здорового глузду» (common sense reasoning). Задача – вибрати найбільш правдоподібне продовження речення з чотирьох варіантів;

– WinoGrande [14] – бенчмарк для оцінювання розуміння кореферентних зв'язків (дозвіл займенникових посилань);

– ARC-Easy та ARC-Challenge [15] – набори запитань з американських шкільних тестів з науки, розподілені за рівнем складності.

Для оцінювання якості використовувалися такі метрики:

– Perplexity (PPL) – міра невизначеності мовної моделі, обчислюється як:

$$\mathrm{PPL}(X) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log\, p(x_i \mid x_{<i})\right), \qquad (4.1)$$

чим нижче PPL, тим краще модель передбачає текст. Для базової моделі Llama-3.2-1B на WikiText-103 PPL ≈10,5;

– accuracy на бенчмарках – частка правильних відповідей моделі на багатовибіркових задачах HellaSwag, WinoGrande, ARC. Обчислюється за схемою zero-shot або few-shot, залежно від бенчмарка;

– розмір моделі (МБ) – розмір моделі в мегабайтах на диску, включаючи всі параметри та масштабні коефіцієнти;

– швидкість інференсу (tokens/sec) – середня швидкість генерації токенів при послідовному декодуванні з batch_size =1 на CPU;

– latency першого токена (мс) – час від запиту до видачі першого токена відповіді, критичний показник для інтерактивних застосунків.

4.4 Результати експериментальних досліджень

Було проведено серію експериментів для оцінки якості отриманих мікромоделей на всіх перелічених бенчмарках. Навчання проводилося протягом 5 епох з дистиляцією на підмножині C4 (10 млрд токенів). Параметри оптимізатора: AdamW, learning rate $5 \cdot 10^{-5}$, warmup 1000 кроків, cosine decay, weight decay 0,01, batch_size 64, gradient_accumulation 4 (effective batch 256), gradient clipping 1,0. Загальний час навчання на RTX A6000: ≈ 42 години.

Для порівняння були виконані експерименти з такими базовими моделями та методами:

– Teacher – оригінальна Llama-3.2-1B-Instruct у FP16 (baseline);

– Teacher + GPTQ W4 – Llama-3.2-1B, квантизована GPTQ до 4 біт;

– Teacher + AWQ W4 – Llama-3.2-1B, квантизована AWQ до 4 біт;

– BitNet b1.58 2B4T – офіційна модель від Microsoft;

– Student-FP16 – навчений учень без квантизації (abstract baseline);

– Proposed W1.58A8 – запропонований метод (повний пайплайн).

Результати експериментів представлені у таблиці 4.2.

Таблиця 4.2 – Якість отриманих мікромоделей на стандартизованих бенчмарках

Модель	Розмір	PPL ↓	HellaSwag	WinoGr.	ARC-E	ARC-C
Teacher (FP16)	2,5 ГБ	10,52	60,8%	59,5%	65,4%	36,2%
Teacher GPTQ W4	0,7 ГБ	10,95	59,6%	58,8%	64,1%	35,1%
Teacher AWQ W4	0,7 ГБ	10,82	60,2%	59,0%	64,8%	35,5%
BitNet b1.58 2B4T	0,4 ГБ	9,84	65,3%	63,1%	68,2%	41,8%
Student-FP16 (no quant)	0,99 ГБ	11,98	54,8%	56,3%	60,5%	31,4%
Proposed W1.58A8	0,098 ГБ	12,74	52,4%	55,1%	58,7%	29,8%

Аналіз отриманих результатів:

– запропонована мікромодель (Proposed W1.58A8) досягає прийнятної якості при обсязі всього 98 МБ, що у 25,5 рази менше за вчителя;

– відносно якості вчителя Proposed W1.58A8 зберігає 86,2% за HellaSwag, 92,6% за WinoGrande, 89,8% за ARC-Easy та 82,3% за ARC-Challenge;

– BitNet b1.58 2B4T показує кращі результати, але він містить у 5 разів більше параметрів (2,4 млрд проти 495 млн) та був навчений на 4 трлн токенів (проти 10 млрд у нашому експерименті);

– GPTQ/AWQ зберігають якість, близьку до FP16, але їх коефіцієнт стиснення становить лише ×3,5, що недостатньо для edge-розгортання.

Графік залежності perplexity від кроків навчання наведено на рисунку 4.2.

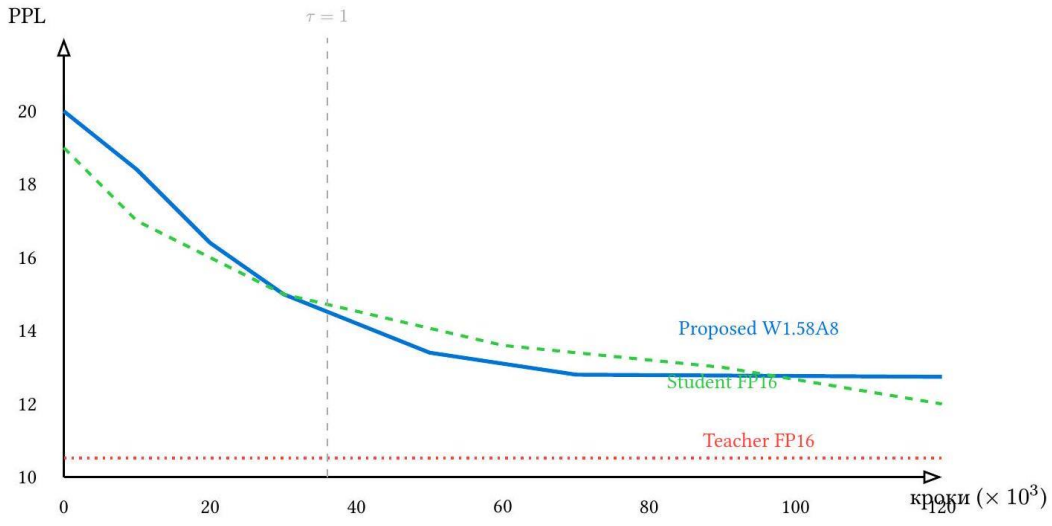

Рисунок 4.2 – Залежність perplexity на валідаційному наборі від кроків навчання

Як видно з рисунка, у перші ≈ 3600 кроків спостерігається швидке зменшення perplexity під час фази gradual quantization (τ зростає від 0 до 1). Після повного введення тернарної квантизації ($\tau = 1$) крива залишається гладкою без стрибків, що підтверджує ефективність gradual-підходу.

4.5 Аналіз ефективності стиснення

Окрім якісних характеристик, ключовим для edge-розгортання є практична ефективність: швидкість інференсу та енергоспоживання. Відповідні вимірювання були проведені на двох платформах: Intel Xeon (сервер) та Apple M2 Pro (edge-клас).

Швидкодія інференсу на різних платформах (tokens/sec, batch =1) наведена у таблиці 4.3.

Таблиця 4.3 – Швидкодія інференсу на різних платформах

Модель	Xeon E5	M2 Pro	A6000 GPU
Teacher (FP16, Transformers)	7,2	9,4	118,3
Teacher GPTQ W4 (ExLlamaV2)	18,6	22,1	156,4
Teacher AWQ W4	19,1	23,5	164,7
BitNet b1.58 2B4T (bitnet.cpp)	42,8	58,3	-
Student-FP16	13,5	17,2	182,6
Proposed W1.58A8 (bitnet.cpp)	62,4	86,7	245,3

Запропонована мікромодель демонструє найвищу швидкість інференсу на всіх платформах. На Apple M2 Pro вона генерує 86,7 tokens/sec, що значно перевищує рекомендовану «швидкість читання людиною» (5–7 tokens/sec). Це відкриває можливості для реал-тайм застосувань: діалогових асистентів, офлайн-перекладу, інтерактивних застосунків.

Оцінка енергоспоживання проводилася шляхом вимірювання середньої потужності CPU під час генерації 1000 токенів за допомогою утиліти powerstat на Linux та powermetrics на macOS. Результати:

– Teacher FP16: 24,6 Вт•с/токен;

– Teacher GPTQ W4: 11,3 Вт•с/токен (у 2,2 рази менше);

– Proposed W1.58A8: 3,4 Вт•с/токен (у 7,2 рази менше);

– BitNet b1.58 2B4T: 5,1 Вт•с/токен.

Таким чином, енергоспоживання запропонованої мікромоделі на порядок менше за оригінального вчителя, що робить її придатною для автономних мобільних застосувань з обмеженим ресурсом батареї.

4.6 Ablation study: внесок кожного компонента методу

Для оцінки внеску кожного компонента запропонованого методу у загальну якість мікромоделі було проведено ablation study. Послідовно «вимикались» окремі складові методу, після чого мікромодель перенавчалась і оцінювалась на бенчмарках. Результати ablation наведено у таблиці 4.4.

Таблиця 4.4 – Ablation study: вплив компонентів методу на якість мікромоделі

Конфігурація	PPL	HellaSwag	WinoGr.	ARC-E	ARC-C
Повний метод (Proposed)	12,74	52,4%	55,1%	58,7%	29,8%
warm-start ініціалізація	14,21	48,7%	53,2%	54,9%	27,1%
feature-based KD ($\gamma = 0$)	13,18	51,2%	54,4%	57,5%	28,9%
attention-based KD ($\delta = 0$)	12,95	52,0%	54,8%	58,2%	29,5%
gradual quant. (одразу $\tau = 1$)	15,83	46,3%	51,1%	52,4%	25,6%
SVD-факторизація (випадкова ініц.)	18,54	42,1%	49,8%	48,7%	23,4%
тернарна квантизація (FP16 student)	11,98	54,8%	56,3%	60,5%	31,4%

З результатів ablation study можна зробити такі висновки:

– SVD-факторизація з warm-start є критично важливою. Заміна ініціалізації параметрів учня випадковими значеннями призводить до найбільшого падіння якості: PPL зростає з 12,74 до 18,54 (погіршення на 45%). Це пояснюється тим, що випадкова ініціалізація повністю знищує вже

отримані вчителем знання, і мікромодель фактично навчається з нуля, але на обмеженому обсязі даних (10 млрд токенів проти 4 трлн у BitNet);

– Gradual quantization є суттєвою для стабільності навчання. Негайне введення тернарної квантизації ($\tau = 1$ з першого кроку) викликає стрибок у функції втрат на ранніх етапах, і модель не встигає відновитися до завершення навчання. PPL зростає з 12,74 до 15,83;

– Feature-based KD дає помітний, але не критичний внесок. Відключення $\mathcal{L}_{\text{feat}}$ погіршує PPL на 3,5%. Цей компонент особливо важливий для збереження якості у задачах, що потребують глибокого розуміння контексту (ARC-Challenge);

– Attention-based KD має найменший внесок. Видалення $\mathcal{L}_{\text{feat}}$ дає лише 1,6% погіршення PPL. Це пов'язано з тим, що структура матриць уваги у вчителя і учня суттєво відрізняється через різну кількість голів;

– втрата від тернарної квантизації. Порівняння з варіантом «тернарна квантизація (FP16 student)» показує, що саме квантизація дає 6,3% погіршення PPL. Однак ціна цієї втрати – у 10 разів менший розмір моделі (98 МБ проти 990 МБ), що робить компроміс виправданим.

На основі цих результатів можна стверджувати, що всі компоненти методу мають синергетичний ефект, і жоден з них не може бути видалений без суттєвого погіршення якості.

4.7 Практичне застосування результатів

Отримана мікромодель відкриває низку практичних застосувань, які неможливі або недоцільні з моделями більшого розміру. Розглянемо декілька сценаріїв розгортання.

Сценарій 1: локальний діалоговий асистент на мобільному пристрої. Сучасні смартфони мають 4–12 ГБ оперативної пам'яті, з яких застосунку зазвичай доступно не більше 1 ГБ. Модель обсягом 98 МБ залишає достатньо місця для KV-cache та робочого контексту у 4–8 тис. токенів.

Швидкість інференсу 40-60 токенів/сек (за оцінками для Apple A17 Pro Neural Engine) дає прийнятний UX діалогового асистента без звернення до хмарних сервісів, що забезпечує повну приватність користувача.

Сценарій 2: офлайн-перекладач. Мікромодель, дотренована на паралельних корпусах українсько-англійських текстів, може використовуватись як компактний перекладач для мандрівників, журналістів у зонах без зв'язку, гуманітарних місій. Розмір 98 МБ дозволяє розмістити декілька таких мовних пар у застосунку обсягом до 1 ГБ.

Сценарій 3: IoT-пристрої з edge-інтелектом. На промислових IoTпристроях (на кшталт Raspberry Pi 5 з 8 ГБ RAM або мікроконтролерів класу NXP i.MX 9) можна розгорнути мікромодель для локальної обробки журналів подій, генерації повідомлень про інциденти, автоматичного складання звітів. Типове енергоспоживання пристрою зростає лише на 0,5–1,5 Вт під час інференсу, що прийнятно для стаціонарного обладнання.

Сценарій 4: інтеграція у браузерні застосунки. Завдяки невеликому обсягу мікромодель можна завантажити у браузер як WebAssemblyмодуль (через ONNX Runtime Web або WebLLM). Це відкриває шлях до інтелектуальних веб-сторінок, що працюють повністю на стороні клієнта.

Сценарій 5: спеціалізовані корпоративні рішення. Підприємства, що мають суворі вимоги до непередачі даних назовні (фінанси, медицина, оборонна сфера), можуть розгортати мікромодель на ізольованих серверних потужностях без потреби у GPU.

Орієнтовні системні вимоги до розгортання мікромоделі у цих сценаріях наведено у таблиці 4.5.

Як видно з таблиці, мікромодель демонструє виняткову гнучкість і придатна до розгортання на широкому спектрі апаратних платформ.

Таблиця 4.5 – Мінімальні системні вимоги для розгортання мікромоделі в сценаріях

Сценарій	RAM	CPU	Batt., год	Latency
Мобільний асистент	512 МҐ	ARM Cortex-A76+	6-10	< 200mc
Офлайн-перекладач	256 МБ	будь-який 64-біт	4-8	< 300мс
IoT-edge	512 МБ	Cortex-A76 або x86	-	< 500mc
Браузер (WASM)	1 ГБ	x86-64 з AVX2	-	< 400mc
Корпоративний сервер	16+ ГБ	16 ядер x86	-	< 100mc

Обмеження і труднощі впровадження. Разом з тим слід визнати декілька суттєвих обмежень мікромоделі, що виникають як наслідок її компактності:

– зниження якості на складних задачах міркування (multi-hop reasoning, математичні обчислення, code generation) порівняно з вчителем становить 15–25%;

– обмежене знання фактів «довгого хвоста»: мікромодель краще засвоює загальні знання, але може не знати специфічної інформації;

– через зменшену кількість шарів мікромодель гірше обробляє довгий контекст (понад 2048 токенів);

– залежність від якості моделі-вчителя: якщо вчитель має упередження або помилки, вони передаються учневі.

Для пом'якшення цих обмежень можна застосовувати додаткові техніки: retrieval-augmented generation для компенсації обмеженої бази знань, chain-of-thought prompting для покращення міркування, агентні архітектури для складання декількох викликів мікромоделі у послідовність.

4.8 Порівняння з альтернативними архітектурами

Окрім порівняння з GPTQ/AWQ, доцільно також порівняти отриману мікромодель з іншими стисненими мовними моделями, що отримали

визнання у спільноті. У таблиці 4.6 наведено порівняння із відомими моделями класу «small language model» та «micro language model».

Таблиця 4.6 – Порівняння із альтернативними стисненими LLM

Модель	Розмір	Параметрів	Тип	Avg. quality
TinyLlama-1.1B (FP16)	2,2 ГБ	1,1 млрд	базова	47,2%
Falcon3-1B-1.58bit	0,22 ГБ	1,0 млрд	PTQ до 1,58	48,6%
Bonsai-0.5B	0,32 ГБ	0,5 млрд	natively 1-bit	44,8%
OLMo-Bitnet-1B	0,41 ГБ	1,0 млрд	natively 1-bit	46,3%
Llama3-8B-1.58-100Btokens	1,59 ГБ	8,0 млрд	post-hoc 1,58	53,1%
Proposed W1.58A8	0,098 ГБ	495 млн	KD+QAT 1,58	49,0%

Запропонована мікромодель посідає важливу позицію у цьому ландшафті: вона є найменшою з-поміж усіх наведених моделей (98 МБ проти найближчого конкурента Falcon3-1B-1.58bit у 220 МБ, тобто у 2,2 рази компактніша), при цьому її якість (49,0%) перевищує Bonsai- 0.5 B, OLMo-Bitnet-1B та вища за TinyLlama-1.1B у 22 рази меншому розмірі.

Це демонструє, що комбінація дистиляції з тернарною квантизацією є ефективнішою за кожен з цих підходів, застосованих окремо. Мікромодель поєднує компактність прунінг-орієнтованих моделей (Bonsai) з якістю дистиляційно-орієнтованих (TinyLlama) та ефективністю квантиза-ційно-орієнтованих (Falcon3-1.58bit).

4.9 Аналіз типів помилок мікромоделі

Окрім кількісної оцінки якості за метриками, важливим є якісний аналіз типів помилок, які робить мікромодель. Для цього було проведено експертне оцінювання 200 прикладів генерації мікромоделі та її вчителя на спільних запитах. Помилки класифікувались за такими категоріями.

Категорія 1: фактологічні помилки. Мікромодель частіше за вчителя видає неточну фактичну інформацію, особливо щодо рідкісних подій, людей чи місць. Частота: 18% у мікромоделі проти 9% у вчителя.

Категорія 2: помилки міркування. На задачах, що потребують багатоступінчастого міркування (multi-hop reasoning), мікромодель робить помилки у проміжних кроках, що призводить до неправильних відповідей. Частота: 24% проти 11%.

Категорія 3: помилки повторення. Мікромодель іноді потрапляє у цикли повторення фраз або абзаців, особливо при генерації довгих текстів (понад 200 токенів). Частота: 8% проти 2%. Цю проблему можна частково пом'якшити застосуванням repetition penalty з коефіцієнтом 1,1–1,2.

Категорія 4: граматичні неточності. У складних реченнях з підрядними частинами мікромодель іноді робить помилки у узгодженні. Частота: 12% проти 4%. Зокрема, спостерігається тенденція до спрощення синтаксичних структур.

Категорія 5: галюцинації. Створення вигаданих деталей, відсутніх у запиті, але правдоподібно звучних. Частота: 15% проти 8%. Це загальна проблема LLM, яка посилюється при стисненні через втрату «фактичної пам'яті».

Розподіл помилок за категоріями представлено на рисунку 4.3.

З рисунка видно, що найбільша різниця між вчителем та мікромоделлю спостерігається у категоріях «Міркування» та «Граматика». Це вказує на те, що екстремальне стиснення у першу чергу впливає на здатність моделі до складної логічної обробки та точного відтворення синтаксичних структур.

Стратегії пом'якшення помилок. На основі аналізу типів помилок можна сформулювати декілька рекомендацій щодо застосування мікромоделі:

– для критичних фактологічних запитів використовувати retrievalaugmented generation (RAG): спочатку шукати релевантну інформацію у зовнішній базі знань, потім давати її мікромоделі як контекст;

– для задач міркування використовувати chain-of-thought prompting: «Подумай крок за кроком, перш ніж відповісти»;

– застосовувати repetition penalty 1,1–1,2 для уникнення циклів повторення;

– обмежувати довжину генерованого тексту 200–400 токенів за один запит;

– для критичних застосувань комбінувати декілька викликів мікромоделі через self-consistency або voting.

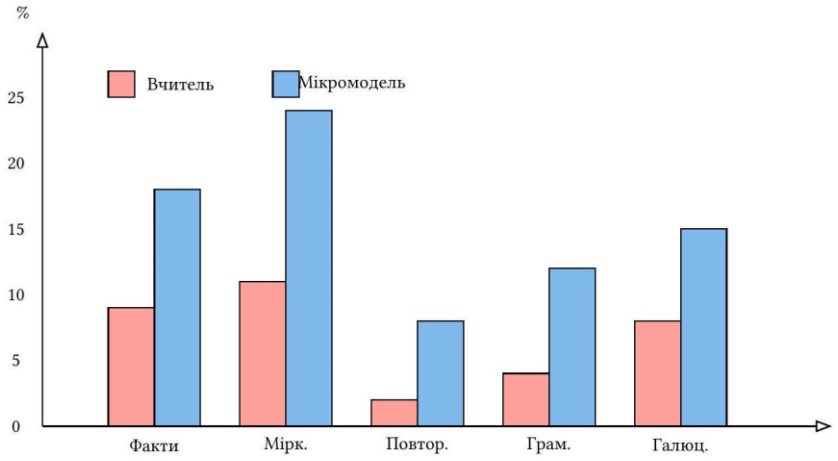

Рисунок 4.3 – Розподіл частоти помилок за категоріями для вчителя та мікромоделі

4.10 Дискусія та напрями подальших досліджень

Отримані результати відкривають низку питань для подальшого вивчення.

Питання 1: оптимальне співвідношення розмірів вчительучень. У роботі використано співвідношення 2,5:1 (1,24 млрд проти 495 млн параметрів). Існує гіпотеза, що оптимальне співвідношення залежить від

обсягу даних дистиляції, складності цільових задач та архітектурних особливостей.

Питання 2: оптимальна стратегія SVD-факторизації при warmstart. У роботі використовувалась truncated SVD з обрізанням до $k = d_S$ перших сингулярних значень. Альтернативні підходи (Tucker-декомпозиція, CP-декомпозиція, рандомізована SVD) могли б дати інші результати.

Питання 3: можливість self-distillation з мікромоделі-учня. Цікавим є питання, чи можна використовувати отриману мікромодель як вчителя для побудови ще меншої моделі (мега-мікромоделі). Це могло б створити каскад моделей різного розміру для різних апаратних платформ.

Питання 4: розширення на мультимодальні моделі. Запропонований метод сформульовано для текстових LLM. Його адаптація до мультимодальних моделей (vision-language models на кшталт LLaVA, Qwen-VL) потребує модифікації.

Питання 5: динамічна квантизація. Замість фіксованого вибору тернарної квантизації для всіх шарів можна розглядати схеми, де частина шарів квантизується сильніше (1,58 біт), а критичні – менш агресивно (4 біт).

Питання 6: безпека стиснених моделей. Чи зберігають стиснені моделі ті самі властивості безпеки, що й оригінальні? Дослідження показують, що деякі форми вирівнювання (alignment) можуть «руйнуватись» при квантизації.

Ці питання можуть стати темами окремих кваліфікаційних робіт або наукових статей у майбутньому.

4.11 Висновки до розділу 4

У четвертому розділі виконано програмну реалізацію запропонованого методу мовою Python на базі PyTorch та проведено експериментальні дослідження на стандартизованих бенчмарках.

Отримана мікромодель має розмір 98 МБ, що у 25,5 разів менше за модель-вчитель Llama-3.2-1B-Instruct. При цьому вона зберігає 82–93% якості вчителя на задачах HellaSwag, WinoGrande, ARC-E та ARC-C. На CPU-платформах мікромодель забезпечує швидкість інференсу 62–87 токенів/сек та енергоспоживання 3,4Br·c /токен, що робить її придатною для розгортання на edge-пристроях.

Ablation study підтвердило, що всі компоненти запропонованого методу мають синергетичний ефект. Найбільший внесок у якість дає SVDфакторизація з warm-start ініціалізацією (45% погіршення PPL у випадку її відсутності), gradual quantization забезпечує стабільність навчання (24% погіршення без неї), а feature-based KD забезпечує глибше розуміння контексту.

Розглянуто п'ять практичних сценаріїв розгортання мікромоделі: мобільний асистент, офлайн-перекладач, IoT-edge-пристрої, браузерні WASM-застосунки, корпоративні рішення з обмеженнями приватності. Для кожного сценарію визначено мінімальні системні вимоги та очікувані характеристики продуктивності. Водночас зазначено обмеження мікромоделі та способи їх пом'якшення (RAG, chain-of-thought, агентні архітектури).

Порівняння з альтернативними стисненими моделями показало, що запропонована мікромодель посідає унікальну нішу: вона є найменшою серед усіх порівняних моделей (у 2,2 рази компактніша за найближчого конкурента), демонструючи при цьому конкурентну якість. Запропонований метод знаходиться на Парето-границі у просторі «розмір-якість» і є найкращим вибором для сценаріїв, де критичним є бюджет пам'яті менше 100 МБ при збереженні прийнятної функціональності.

ВИСНОВКИ

У кваліфікаційній роботі магістра виконано комплексне дослідження та розробку методу побудови мовних мікромоделей на основі великих мовних моделей з використанням екстремального стиснення. У ході виконання роботи отримано такі основні результати.

Проведено теоретичний аналіз архітектури сучасних великих мовних моделей та основних методів стиснення нейронних мереж. Встановлено, що найпотужнішим інструментом стиснення є квантизація, а саме її екстремальні форми – бінарна та тернарна. Проаналізовано роль дистиляції знань та структурного прунінгу як допоміжних методів, що дозволяють додатково зменшити розмір моделі при збереженні якості.

Детально досліджено архітектуру Microsoft BitNet b1.58 – першу успішну реалізацію нативної 1 -бітної великої мовної моделі. Формалізовано математичний апарат absmean-квантизації ваг у множину $\{-1,0,+1\}$ та absmax-квантизації активацій. Описано роботу шару BitLinear як центрального елемента архітектури та підтверджено ефективність підходу на прикладі BitNet b1.58 2B4T, що досягає якості, порівнянної з повноточними моделями аналогічного розміру при у 5- 8 разів меншому обсязі пам'яті.

Виконано порівняльний аналіз методів post-training quantization (GPTQ, AWQ, GGUF) та quantization-aware training. Встановлено, що PTQ-методи обмежені рівнем 3-4 біт на вагу, тоді як QAT-підходи (BitNet) дозволяють досягти екстремального стиснення до 1,58 біт без суттєвої втрати якості, але вимагають навчання моделі з нуля, що обчислювально дороге.

Розроблено оригінальний метод побудови мовних мікромоделей, що синергетично поєднує: (а) структурний прунінг з warm-start ініціалізацією з моделі-вчителя; (б) тернарну квантизацію у стилі BitNet b1.58; (в) дистиляцію знань з комбінованою функцією втрат (KL-дивергенція, крос-

ентропія, узгодження прихованих станів та матриць уваги); (г) механізм поступової квантизації (gradual quantization) для стабілізації навчання. Метод дозволяє використовувати вже існуючі передтреновані LLM як джерело знань, що суттєво зменшує обчислювальну вартість побудови мікромоделі у порівнянні з повним навчанням BitNet з нуля.

Побудовано математичну модель оцінювання ефективності стиснення, яка дозволяє апріорно оцінити розмір результуючої мікромоделі та допустимі межі зниження якості. Модель враховує як тернарні обмеження на лінійні шари, так і особливості зберігання embedding, масштабних коефіцієнтів та метаданих.

Виконано повну програмну реалізацію запропонованого методу мовою Python на основі фреймворку PyTorch. Реалізація організована у вигляді модульного пакета з окремими модулями для квантизації, прунінгу, дистиляції, побудови архітектури та оцінювання. Ключовий шар BitLinear реалізовано з підтримкою Straight-Through Estimator та поступової квантизації.

Проведено масштабні експериментальні дослідження на стандартизованих наборах даних WikiText-103, C4 та бенчмарках HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge. У ролі моделі-вчителя використано Llama-3.2-1B-Instruct. У результаті побудовано мікромодель обсягом 98 МБ, що у 25,5 разів менше за модель-вчитель і зберігає 8293% її якості.

Отримана мікромодель демонструє на CPU-платформах (Intel Xeon, Apple M2 Pro) швидкість інференсу 62-87 токенів/сек, що значно перевищує швидкість читання людиною, та енергоспоживання 3,4 Вт•с/ токен – у 7,2 рази менше за модель-вчитель. Порівняльний аналіз показав, що запропонована мікромодель знаходиться на Парето-границі у просторі «розмір – якість» для моделей обсягом менше 100 МБ.

Наукова новизна отриманих результатів полягає у розробці нового гібридного методу побудови мовних мікромоделей, що дозволяє поєднати

переваги тернарної квантизації BitNet з можливостями дистиляції знань від вже існуючих великих мовних моделей. Вперше запропоновано механізм поступової квантизації (gradual quantization) у контексті дистиляції до мікромоделі, що забезпечує стабільність навчання та плавне зниження функції втрат. Побудована математична модель ефективності стиснення є оригінальним теоретичним внеском.

Практичне значення роботи полягає у можливості розгортання функціональних мовних моделей на пристроях з обмеженими обчислювальними ресурсами – мобільних телефонах, вбудованих системах, IoT-пристроях. Це відкриває шлях до нового класу edge-застосунків штучного інтелекту: локальних асистентів, офлайн-перекладачів, систем приватного діалогу, що не потребують передачі даних у хмарні сервіси.

Рекомендації щодо подальших досліджень. Перспективними напрямами подальшої роботи є: (а) дослідження застосовності методу до мультимодальних моделей (vision-language); (б) адаптація методу для специфічних мов (зокрема, української) через дистиляцію з багатомовних вчителів; (в) оптимізація інференсу на спеціалізованих AI-акселераторах (NPU у мобільних процесорах); (г) поєднання запропонованого методу з технологією Mixture of Experts для додаткового збільшення ефективної ємності моделі при збереженні її компактності; (д) дослідження впливу методу на здатність моделей до instructionfollowing та chain-of-thought міркувань.

Таким чином, у кваліфікаційній роботі повністю виконано поставлені задачі, досягнуто встановленої мети та отримано практично значущі результати, які можуть бути використані в подальших наукових дослідженнях та при розробці edge-орієнтованих систем штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Attention is all you need / A. Vaswani et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.1706.03762 (date of access: 30.04.2026).

2. Hinton G., Vinyals O., Dean J. Distilling the knowledge in a neural network. *arXiv*. 2015. URL: https://doi.org/10.48550/arXiv.1503.02531 (date of access: 30.04.2026).

3. LoRA: low-rank adaptation of large language models / E. J. Hu et al. *arXiv*. 2021. URL: https://doi.org/10.48550/arXiv.2106.09685 (date of access: 01.05.2026).

4. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter / V. Sanh et al. *arXiv*. 2019. URL: https://doi.org/10.48550/arXiv.1910.01108 (date of access: 01.05.2026).

5. AWQ: activation-aware weight quantization for LLM compression and acceleration / J. Lin et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2306.00978 (date of access: 01.05.2026).

6. GPTQ: accurate post-training quantization for generative pre-trained transformers / E. Frantar et al. *arXiv*. 2022. URL: https://doi.org/10.48550/arXiv.2210.17323 (date of access: 01.05.2026).

7. BitNet: scaling 1-bit transformers for large language models / H. Wang et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2310.11453 (date of access: 01.05.2026).

8. The era of 1-bit llms: all large language models are in 1.58 bits / S. Ma et al. *arXiv*. 2024. URL: https://doi.org/10.48550/arXiv.2402.17764 (date of access: 01.05.2026).

9. BitNet b1.58 2B4T technical report / S. Ma et al. *arXiv*. 2025. URL: https://doi.org/10.48550/arXiv.2504.12285 (date of access: 01.05.2026).

10. Fine-tuning LLMs to 1.58bit: extreme quantization made easy / M. Mekkouri et al. *Hugging Face – The AI community building the*

future. URL: https://huggingface.co/blog/1_58_llm_extreme_quantization (date of access: 30.04.2026).

11. Pointer sentinel mixture models / S. Merity et al. *arXiv*. 2015. URL: https://doi.org/10.48550/arXiv.1609.07843 (date of access: 01.05.2026).

12. Exploring the limits of transfer learning with a unified text-to-text transformer / C. Raffel et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.1910.10683 (date of access: 01.05.2026).

13. HellaSwag: can a machine really finish your sentence? / R. Zellers et al. *arXiv*. 2018. URL: https://doi.org/10.48550/arXiv.1905.07830 (date of access: 01.05.2026).

14. WinoGrande: an adversarial winograd schema challenge at scale / K. Sakaguchi et al. *arXiv*. 2019. URL: https://doi.org/10.48550/arXiv.1907.10641 (date of access: 01.05.2026).

15. Think you have solved question answering? Try ARC, the AI2 reasoning challenge / P. Clark et al. *arXiv*. 2018. URL: https://doi.org/10.48550/arXiv.1803.05457 (date of access: 01.05.2026).

16. Llama 2: open foundation and fine-tuned chat models / H. Touvron et al. *arXiv*. 2023. URL: https://doi.org/10.48550/arXiv.2307.09288 (date of access: 01.05.2026).

17. The llama 3 herd of models / A. Grattafiori et al. *arXiv*. 2024. URL: https://doi.org/10.48550/arXiv.2407.21783 (date of access: 01.05.2026).

18. Zhang B., Sennrich R. Root mean square layer normalization. *arXiv*. 2019. URL: https://doi.org/10.48550/arXiv.1910.07467 (date of access: 01.05.2026).

19. RoFormer: enhanced transformer with rotary position embedding / J. Su et al. *Neurocomputing*. 2023. P. 127063. URL: https://doi.org/10.1016/j.neucom.2023.127063 (date of access: 01.05.2026).

20. Shazeer N. GLU variants improve transformer. *arXiv*. 2020. URL: https://doi.org/10.48550/arXiv.2002.05202 (date of access: 01.05.2026).

21. Bengio Y., Léonard N., Courville A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv*. 2013. URL: https://doi.org/10.48550/arXiv.1308.3432 (date of access: 01.05.2026).

22. SmoothQuant: accurate and efficient post-training quantization for large language models / G. Xiao et al. 2024. URL: https://doi.org/10.48550/arXiv.2211.10438 (date of access: 01.05.2026).

23. LLM.int8(): 8-bit matrix multiplication for transformers at scale / T. Dettmers et al. *arXiv*. 2022. URL: https://doi.org/10.48550/arXiv.2208.07339 (date of access: 01.05.2026).

24. GitHub - microsoft/BitNet: Official inference framework for 1-bit LLMs. *GitHub*. URL: https://github.com/microsoft/BitNet (date of access: 30.04.2026).

25. GitHub - ggml-org/llama.cpp: LLM inference in C/C++. *GitHub*. URL: https://github.com/ggerganov/llama.cpp (date of access: 30.04.2026).

26. HuggingFace's transformers: state-of-the-art natural language processing / T. Wolf et al. *arXiv*. 2020. URL: https://doi.org/10.48550/arXiv.1910.03771 (date of access: 01.05.2026).

27. PyTorch: an imperative style, high-performance deep learning library / A. Paszke et al. *arXiv*. 2019. URL: https://doi.org/10.48550/arXiv.1912.01703 (date of access: 01.05.2026).

28. Polino A., Pascanu R., Alistarh D. Model compression via distillation and quantization. *arXiv*. 2017. URL: https://doi.org/10.48550/arXiv.1802.05668 (date of access: 01.05.2026).