НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____Оксана ТИМОЩУК
"__" _____2025 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Двонаправлена адаптація великих мовних моделей-декодерів для обробки української природної мови»

Виконав:
студент IV курсу, групи КА-12
Гавлицький Іван Вікторович _____

Керівник:
асистент кафедри ММСА,
Баздирев Антон Андрійович _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
канд. фіз.-мат. наук,
Статкевич Віталій Михайлович _____

Рецензент:
старший викладач кафедри СП, к.т.н.,
Кислий Роман Володимирович _____

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

Київ — 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Оксана ТИМОЩУК
"___" _____2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Гавлицькому Івану Вікторовичу

1. Тема роботи «Двонаправлена адаптація великих мовних моделей-декодерів для обробки української природної мови», керівник роботи асистент кафедри ММСА Баздирев Антон Андрійович, затверджені наказом по університету від
«26» травня 2025 р. №1759-с.
2. Термін подання студентом роботи: _____
3. Вихідні дані до роботи: Наявні корпуси текстів українською та англійською мовами.
4. Зміст роботи: Дослідження предметної області обробки текстової інформації, зокрема великими мовними моделями-декодерами. Адаптація відповідних моделей до задач, що вимагають розуміння двонаправленого контексту. Аналіз роботи та ефективності модифікованих моделей.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.
6. Консультанти розділів роботи[1]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

[1] Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

7.Дата видачі завдання 11.04.025

№ з/п	Назва етапів виконання бакалаврської дипломної роботи (БДР)	Термін виконання етапів роботи	Примітка
1	Затвердження теми БДР	14.04.2025 – 20.04.2025	Виконано
2	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	14.04.2025 – 20.04.2025	Виконано
3	Ознайомлення з ДСТУ	14.04.2025 – 20.04.2025	Виконано
4	Проведення дослідження за темою БДР під керівництвом керівника	14.04.2025 – 20.04.2025	Виконано
5	Завершення роботи над першим варіантом частини БДР	21.04.2025 – 27.04.2025	Виконано
6	Проведення роботи над програмним продуктом	28.04.2025 – 04.05.2025	Виконано
7	Проведення роботи над експериментальною частиною БДР	05.05.2025 – 11.05.2025	Виконано
8	Оформлення БДР та аналіз отриманих результатів	12.05.2025 – 18.05.2025	Виконано

Студент Іван ГАВЛИЦЬКИЙ

Керівник Антон БАЗДИРЕВ

РЕФЕРАТ

Дипломна робота: 148 с., 8 табл., 15 рис., 2 додатки, 32 джерела.

ВЕЛИКІ МОВНІ МОДЕЛІ, ДЕКОДЕР-ОНЛІ, ДВОНАПРАВЛЕНІСТЬ, УКРАЇНСЬКА МОВА, АДАПТАЦІЯ, ТРАНСФОРМЕР, МАРКУВАННЯ ПОСЛІДОВНОСТЕЙ, МАЛОРЕСУРСНІ МОВИ

Об'єкт дослідження – авторегресивні трансформерні моделі великої розміру (LLMs) декодерного типу.

Предмет дослідження – методи перетворення декодер-онлі моделей на інструмент, здатний працювати з двостороннім контекстом для задач аналізу української природної мови.

Мета роботи – розробити, дослідити та експериментально обґрунтувати ефективність підходів до двонаправленої адаптації авторегресивних мовних моделей шляхом архітектурних модифікацій і спеціалізованого донавчання, а також визначити вплив мовної специфіки та обраних стратегій адаптації на результати в задачах маркування послідовностей українською мовою.

Результатом роботи є розроблений фреймворк двонаправленої адаптації декодерних моделей на основі модифікації механізму уваги та цільового донавчання, а також демонстрація переваг запропонованого підходу на задачах маркування послідовностей для української мови. Проведено комплексний аналіз впливу різних стратегій донавчання та мовної специфіки корпусу, здійснено порівняння з сучасними енкодерними моделями, виявлено практичні обмеження існуючих метрик оцінки двонаправленості.

Подальший розвиток дослідження передбачає впровадження альтернативних позиційних енкодингів та схем пулінгу, розширення експериментального бенчмарку на задачі семантичних ембедингів, а також продовження аналізу внутрішніх метрик процесу адаптації трансформерних моделей.

ABSTRACT

Thesis: 148 pages, 8 tables, 15 figures, 2 appendices, 32 references.

LARGE LANGUAGE MODELS, DECODER-ONLY, BIDIRECTIONALITY, UKRAINIAN LANGUAGE, ADAPTATION, TRANSFORMER, SEQUENCE LABELING, LOW-RESOURCE LANGUAGES

The object of the study is autoregressive large-length transformer models (LLMs) of the decoder type.

The subject of the study is methods of transforming decoder-only models into a tool capable of working with a bilateral context for the tasks of analyzing Ukrainian natural language.

The aim of the study is to develop, investigate, and experimentally prove the effectiveness of approaches to bidirectional adaptation of autoregressive language models through architectural modifications and specialized retraining, as well as to determine the impact of language specificity and selected adaptation strategies on the results in Ukrainian sequence labeling tasks.

The result of the work is a framework for bidirectional adaptation of decoder models based on modification of the attention mechanism and targeted retraining, as well as a demonstration of the advantages of the proposed approach on sequence labeling tasks for the Ukrainian language. A comprehensive analysis of the impact of different retraining strategies and the language specifics of the corpus is carried out, a comparison with modern encoder models is made, and the practical limitations of existing bidirectionality assessment metrics are revealed.

The further development of the study involves the introduction of alternative positional encodings and pooling schemes, the extension of the experimental benchmark to the task of semantic embedding, as well as the continuation of the analysis of internal metrics of the process of adaptation of transformer models.

ЗМІСТ

ВСТУП

Великі мовні моделі декодерного типу сьогодні визначають темпи розвитку обробки природної мови. Їхня популярність пояснюється гнучкістю авторегресивного навчання, здатністю масштабуватися до сотень мільярдів параметрів і природним умінням генерувати зв'язний текст без додаткового донавчання. Проте низка прикладних завдань, такі як маркування послідовностей, вимагає двостороннього огляду контексту, що традиційно надається енкодерними моделями, подібними до BERT. Виникає методологічний розрив між універсальністю декодерів і потребою в двонаправленості, особливо відчутний у випадку малоресурсних мов, зокрема української.

Об'єктом цього дослідження є авторегресивні мовні моделі, а предметом – підходи до їхньої адаптації для завдань, де критичною є симетрична робота з контекстом. Йдеться про можливість «розблокувати» двонаправленість у вже навчених декодер-онлі архітектурах, не порушуючи їх корисних властивостей і не збільшуючи суттєво обчислювальні витрати. Складність проблеми полягає в тому, що каузальна маска, закладена в основу авторегресивного механізму уваги, обмежує модель бачити тільки попередні токени, тоді як багато аналітичних задач спираються на інформацію з обох боків цільного слова чи фрази.

Мета роботи полягає у створенні цілісної теоретико-прикладної основи, яка дозволила б перетворити декодер-онлі моделі на інструмент, придатний для енкодер-специфічних завдань без потреби будувати окремий сімейний ряд нейронних мереж. Для її досягнення послідовно аналізуються архітектурні обмеження авторегресивного увагового механізму, досліджуються способи пом'якшити ці обмеження та пропонуються критерії, що кількісно характеризують наявність або відсутність двонаправленого впливу між токе-

нами. У межах експериментальної частини здійснюється перевірка гіпотез на українських корпусах, що охоплюють завдання маркування послідовностей.

Наукова новизна дослідження полягає в системному розгляді проблеми двонаправленої адаптації саме для української мови, де дефіцит високоякісних даних робить пряме навчання великих енкодерів економічно недоцільним. У роботі пропонуються концептуальні засади оцінювання «симетричності» увагових матриць і демонструється, що навіть обмежене донавчання моделей після зняття каузальної маски може покращити метрики якості на послідовних завданнях.

Дослідження поєднує аналітичний огляд сучасної літератури з експериментальною перевіркою отриманих висновків. Використання відкритих корпусів, відтворюваних налаштувань навчання і публічних репозиторіїв робить результати придатними для подальшого розвитку, а висновки – коректними щодо їхньої узагальненості.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд великих мовних моделей (LLMs)

1.1.1 Розвиток обробки природної мови до епохи трансформерів

Еволюція обробки природної мови (NLP) характеризується послідовним переходом від чітко формалізованих граматичних систем до статистичних та нейронних методів навчання. Початкові програмні системи базувалися на регулярних виразах та ручно створених деревах синтаксичного розбору, що вимагало значної експертизи та не забезпечувало масштабованості для реальних обсягів мовних даних.

У 1990-х роках домінуючою парадигмою стали статистичні підходи, зокрема n-грамні ланцюги Маркова та приховані марковські моделі (HMM), які суттєво покращили якість машинного перекладу та розпізнавання мовлення. Проте ці методи обмежувалися локальним аналізом контексту та не могли ефективно обробляти довгострокові залежності в тексті. Подальший розвиток призвів до впровадження методів керованого навчання, включаючи метод опорних векторів (SVM) та умовні випадкові поля (CRF), які досягли високих результатів у класифікації текстів, однак потребували трудомісткої інженерії ознак.

Революційні зміни в галузі розпочалися з 2010-х років із появою глибинного навчання. Рекурентні нейронні мережі (RNN) та їх удосконалені ва-

ріанти LSTM і GRU продемонстрували здатність моделювати довгострокові залежності в послідовностях, ставши основою нейронних систем машинного перекладу. Попри значний прогрес, RNN-архітектури мали фундаментальні обмеження: низьку можливість паралелізації обчислень, проблему затухання та вибуху градієнтів, а також труднощі з обробкою віддалених контекстуальних залежностей.

1.1.2 Трансформерна революція в архітектурі моделей

Представлена у 2017 році архітектура трансформерів [1] стала фундаментальним проривом у галузі обробки природної мови. Революційність підходу полягала у повній відмові від рекурентних зв'язків на користь механізму самоуваги (self-attention), що дозволило моделі безпосередньо враховувати взаємозв'язки між усіма елементами послідовності незалежно від їх взаємного розташування.

Початкова архітектура була розроблена для задач машинного перекладу за принципом енкодер-декодер: енкодер здійснює паралельну обробку всього вхідного контексту, тоді як декодер авторегресивно генерує переклад на основі представлень, сформованих енкодером. Ця модель швидко стала стандартом для широкого спектру задач перетворення послідовностей, включаючи автоматичне реферування, перефразування та граматичне коригування тексту.

1.1.3 Енкодерні архітектури трансформерів

Модульна структура трансформерної архітектури створила передумови для розвитку спеціалізованих варіантів, орієнтованих на конкретні класи завдань. Одним із найвпливовіших представників енкодерних архітектур стала

модель BERT (Bidirectional Encoder Representations from Transformers) [2], яка впровадила інноваційну схему маскованого мовного моделювання (MLM).

Ключовою особливістю BERT є здатність враховувати як попередній, так і наступний контекст під час переднавчання, що радикально підвищило ефективність моделі в задачах, де критичним є двосторонній контекстуальний аналіз. До таких завдань належать розуміння тексту, аналіз тональності, розпізнавання іменованих сутностей та морфосинтаксичний аналіз. Енкодерні архітектури продемонстрували особливу ефективність у задачах класифікації та структурного аналізу тексту.

1.1.4 Декодерні трансформери

Виключно-декодерні архітектури трансформерів представляють окремий клас моделей глибокого навчання, орієнтованих на генеративні задачі обробки природної мови, найяскравішим представником якого є сімейство моделей GPT (Generative Pre-trained Transformer). Ці моделі реалізують авторегресивний підхід до генерації тексту через послідовне передбачення наступного токена на основі попереднього контексту з використанням каузального маскування, що відрізняє їх від енкодерних та енкодер-декодерних архітектур.

Ключовою характеристикою декодерних моделей є їх значний розмір, що суттєво перевищує енкодерні аналоги та налічує мільярди параметрів. Така різниця в масштабі обумовлена як простотою об'єктиву навчання, так і емпірично доведеною масштабованістю цих архітектур. Фундаментальне дослідження Kaplan et al. (2020) [3] встановило так звані «закони масштабування», які продемонстрували, що помилка мовної моделі зменшується за степеневим законом зі збільшенням кількості параметрів моделі, розміру навчального корпусу та обчислювальних ресурсів. Ці результати стали теоретичною основою для подальшого масштабування декодерних архітектур та ініціювали «гонку масштабів» у сфері великих мовних моделей, результатом якої стали моделі розміром у сотні мільярдів параметрів, включаючи GPT, Mistral,

LLaMA, Gemma та інші.

Зростання маштабів додатково породжувалося інтересом до штучного загального інтелекту (AGI), що посилився після дослідження феномену емерджентних властивостей нейронних мереж. Wei et al. (2022) [4] детально проаналізували виникнення якісно нових здібностей моделей при досягненні критичного масштабу, які характеризуються здатністю вирішувати завдання, що не були явно представлені в навчальних даних. Андрій Карпатий у своєму впливовому дослідженні емерджентних здібностей нейронних мереж сформулював ключовий принцип сучасного підходу до навчання великих моделей: «Ми не програмуємо алгоритми – ми створюємо середовище, у якому вони з'являються.» Це спостереження емерджентних властивостей стало центральним аргументом наукової спільноти, зокрема дослідників OpenAI, на користь продовження масштабування моделей як шляху до досягнення AGI.

1.2 Задачі маркування послідовності

1.2.1 Теоретичні основи та класифікація задач

Маркування послідовностей становить фундаментальний клас завдань обробки природної мови, де кожному елементу вхідної послідовності присвоюється мітка з попередньо визначеного скінченного набору категорій. Ця парадигма охоплює широкий спектр лінгвістичних завдань, від базового морфологічного аналізу до складних семантичних анотацій.

Класичними представниками цього класу задач є частиномовна розмітка (POS-tagging), де мітки відповідають граматичним категоріям слів; розпізнавання іменованих сутностей (NER), де мітки визначають типи сутностей або маркують їх відсутність; семантичне тегування з мітками семантичних ролей; а також морфологічний аналіз з мітками, що представляють набори

граматичних ознак словоформ.

Задачі маркування послідовності можна систематизувати відповідно до типу вихідних міток та характеру їх взаємозалежностей:

1. Задачі незалежної класифікації токенів: кожному токену відповідає автономна мітка без урахування міток сусідніх елементів (наприклад, POS-tagging). Для таких задач залежності між мітками мінімальні, а оцінка якості зазвичай проводиться на рівні окремих токенів з використанням метрик точності та F1-міри.
2. Задачі виявлення структурованих інтервалів: мітки визначають межі та типи тематичних сегментів з використанням схем типу BIO (Beginning-Inside-Outside). До цієї категорії належать NER, синтаксичне сегментування (chunking), виділення іменованих сутностей та ідентифікація мовних помилок. Критичним аспектом є оцінка цілісних спанів, де весь іменований об'єкт (наприклад, «Організація Об'єднаних Націй») повинен бути правильно ідентифікований як єдина сутність навіть при складній токенізації.
3. Ієрархічне маркування: завдання з вкладеною структурою міток, що можуть утворювати багаторівневі анотації (наприклад, вкладені сутності різних типів). Стандартні послідовні моделі з одним шаром міток часто неадекватні для таких завдань, що потребує застосування композитних схем маркування або спеціалізованих архітектур з множинними шарами передбачення.
4. Сегментація послідовності: спеціалізований випадок, де завдання полягає в розбитті послідовності на семантично або структурно значущі сегменти за визначеними правилами (сегментація на речення, морфемний аналіз). Ці задачі можуть бути зведені до стандартного маркування через позначення меж сегментів спеціальними мітками.

1.2.2 Порівняльний аналіз декодерних та енкодерних архітектур

Традиційно для розв'язання задач маркування послідовностей застосовувалися двонаправлені енкодерні архітектури – від бі-LSTM у ранніх дослідженнях до BERT та його варіантів у сучасних розробках. Перевага енкодерних моделей обумовлена їх здатністю аналізувати повний контекст токена, враховуючи інформацію з обох сторін послідовності, що часто є критичним для точної класифікації.

Наприклад, для визначення семантичного типу слова «Washington» (особа чи географічна локація) необхідний аналіз як попереднього, так і наступного контексту – можливо, назви штату або дієслова «said», що вказує на суб'єкт мовлення. Енкодерні моделі типу BERT обробляють всю послідовність одночасно, формуючи приховані представлення, що містять інформацію про весь контекст.

Натомість декодерні моделі в стандартному авторегресивному режимі не мають доступу до майбутнього контексту. При наївному застосуванні для класифікації токенів модель може визначати мітку y_i лише на основі попередніх токенів $x_{\leq i}$, не маючи інформації про подальші елементи $x_{i+1:n}$. Це може призводити до помилок класифікації в неоднозначних контекстах, де розрізнення категорій залежить від майбутнього контексту.

Проте декодерні моделі можуть застосовуватися для задач маркування через генеративний підхід – створення послідовності міток замість прямої класифікації токенів. Такий метод забезпечує повне розуміння контексту всієї послідовності перед початком процесу маркування. Однак генеративний підхід характеризується значно вищими обчислювальними витратами порівняно з токенною класифікацією, що є критичним недоліком в контексті масштабів великих мовних моделей та практичних обмежень застосування.

1.3 Двонаправлена адаптація декодерних моделей

Наукова спільнота лише нещодавно розпочала систематичне дослідження методів адаптації декодерних моделей до завдань, що традиційно вирішувалися енкодерними архітектурами та потребують двонаправленого розуміння контексту. Мотивація для такої адаптації обумовлена декількома факторами: по-перше, масштабні декодерні моделі демонструють вищу універсальність та здатність до transfer learning; по-друге, єдина архітектурна основа спрощує інфраструктуру розгортання; по-третє, декодерні моделі показують кращу ефективність в умовах обмежених навчальних даних завдяки потужному переднавчанню.

Водночас існують фундаментальні обмеження декодерних архітектур також і у генеративних завданнях. Зокрема, «прокляття реверсивності» (reversal curse) [5] демонструє неспроможність моделей засвоювати симетричні відношення: навчившись, що «A є B», модель не автоматично розуміє, що «B є A». Це обмеження особливо критичне для завдань, що потребують розуміння каузальних та логічних зв'язків у тексті.

Для подолання цих структурних обмежень дослідники запропонували різноманітні стратегії адаптації. Одним із найбільш перспективних альтернативних підходів є використання дифузійних мовних моделей (Large Language Diffusion Models) [6], що принципово відрізняються від авторегресивної парадигми генерації тексту. На відміну від традиційних декодерних архітектур, які генерують токени послідовно зліва направо, дифузійні моделі використовують ітеративний процес денойзингу, адаптуючи успішні принципи дифузійних моделей з комп'ютерного зору до задач обробки природної мови. Ключовою архітектурною перевагою такого підходу є можливість одночасного врахування всіх позицій у послідовності під час генерації, що повністю усуває фундаментальні обмеження каузальних масок та дозволяє природно враховувати двонаправлений контекст. Однак слід зазначити, що дифузійні підходи залишаються на стадії активних досліджень і потребують повного ретренування з нуля.

З огляду на практичні обмеження дифузійних підходів, більшість досліджень зосереджується на адаптації існуючих декодерних архітектур без кардинальної зміни базової парадигми. Ці методи варіюються від простих технік, що не потребують додаткового навчання, до комплексних модифікацій архітектури та багатостадійних схем переднавчання. Методи без додаткового навчання, попри спірну ефективність, забезпечують максимальну простоту імплементації. Springer et al. (2024) [7] продемонстрували, що просте повторення вхідного тексту може покращити якість ембедінгів. Fu et al. запропонували додавання ембедінга з попереднього шару до початку послідовності наступного шару, імітуючи псевдобідірекційний контекст [8].

Альтернативний напрямок досліджень зосереджений на модифікації механізму уваги під час тонкого налаштування моделі. Li et al. (2023) [9] дослідили повне видалення каузальної маски при адаптації LLaMA2 для задач класифікації та NER. Паралельно розглядалися підходи до селективної модифікації уваги лише в окремих шарах [10] або групах фінальних шарів [11]. Google Research [12] провели систематичну оцінку різних стратегій зняття каузальності на широкому спектрі завдань, встановивши оптимальні конфігурації для різних типів задач.

Серед підходів з додатковим переднавчанням слід виділити LLM2Vec [13] – метод, що передбачає двоетапне навчання для адаптації моделі до двонаправлених можливостей з подальшим тонким налаштуванням на цільових завданнях. Цей підхід демонструє компроміс між обчислювальними витратами та якістю адаптації.

1.4 NLP в малоресурсних мовах

Відповідність сучасним стандартам обробки природної мови залишається фундаментальною проблемою для малоресурсних мов, включаючи українську. Недостатня представленість цих мов у великомасштабних переднавчальних корпусах призводить до неможливості ефективної адаптації моделей

до специфічних мовних завдань, що ускладнюється браком якісних анотованих даних для конкретних прикладних задач.

Дослідники активно розробляють різноманітні підходи для подолання цих структурних обмежень. Joshi et al. (2024) [14] запропонували методи додаткового переднавчання на синтетичних корпусах, автоматично перекладених з високоресурсних мов. Gurgurov et al. (2024) [15] досліджували інтеграцію структурованих лінгвістичних знань через графи знань для покращення розуміння мовних структур.

Останні великомасштабні ініціативи суттєво покращили ситуацію з малоресурсними мовами. Проєкт Meta «No Language Left Behind» [16] та розробка багатомовних бенчмарків, зокрема XTREME [17], стимулювали технологічні компанії інвестувати в розширення мовної підтримки. Ці зусилля призвели до створення більш збалансованих багатомовних моделей та стандартизованих методів оцінки якості для різних мов.

Проте в умовах критичного дефіциту мовних ресурсів ефективність великих мовних моделей все ще значно поступається показникам для високоресурсних мов. Особливо це проявляється в завданнях, що потребують адаптації до нових предметних областей або глибокого контекстуального розуміння тексту. Ця проблема актуалізує необхідність розробки спеціалізованих методів адаптації великих моделей до українських мовних завдань.

1.5 Висновки до першого розділу

У розділі було проведено огляд еволюції NLP від традиційних підходів до сучасних трансформерних архітектур. Розглянуто задачі маркування послідовностей та проаналізовано особливості енкодерних і декодерних архітектур. Виявлено основні обмеження декодерних моделей для задач, що потребують двонаправленого контексту, та розглянуто сучасні методи їх адаптації. Додатково досліджено проблеми застосування NLP-технологій для малоресурсних мов, включаючи українську.

2 ТЕОРЕТИЧНІ ЗАСАДИ

2.1 Трансформерні архітектури та декодерні моделі

2.1.1 Механізм самоуваги

Архітектура трансформера революціонізувала область обробки природної мови, замінивши рекурентні та згорткові компоненти механізмом самоуваги. Фундаментальна інновація полягає у відмові від послідовної обробки елементів вхідної послідовності на користь паралельного обчислення взаємозв'язків між усіма парами позицій.

Нехай вхідна послідовність представлена матрицею $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$, де n означає довжину послідовності, а d_{model} – розмірність внутрішнього представлення моделі. Механізм самоуваги здійснює три лінійні перетворення вхідної матриці через навчальні матриці проекцій:

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}^Q, \quad \mathbf{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_k} \\ \mathbf{K} &= \mathbf{X}\mathbf{W}^K, \quad \mathbf{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_k} \\ \mathbf{V} &= \mathbf{X}\mathbf{W}^V, \quad \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \end{aligned}$$

де $\mathbf{Q}$ – матриця запитів (queries);
$\mathbf{K}$ – матриця ключів (keys);
$\mathbf{V}$ – матриця значень (values);
d_k – розмірність ключового простора;
d_v – розмірність значеннєвого простора.

Скалярна добуткова увага (scaled dot-product attention) обчислюється згідно з формулою:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}.$$

Масштабуючий фактор $\sqrt{d_k}$ відіграє критичну роль у забезпеченні обчислювальної стабільності. Без цього нормування, для великих значень d_k, скалярні добутки $\mathbf{q}_i^T\mathbf{k}_j$ можуть досягати значних величин, що призводить до насичення функції softmax і, як наслідок, до надзвичайно малих градієнтів.

Результуюча матриця має розмірність $n \times d_v$ і являє собою зважену суму рядків матриці значень $\mathbf{V}$, де ваги визначаються оцінками подібності між запитами та ключами. Формально, для i-го елемента результату:

$$\text{out}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{v}_j, \quad \alpha_{ij} = \frac{\exp(\mathbf{q}_i^T\mathbf{k}_j/\sqrt{d_k})}{\sum_{l=1}^{n}\exp(\mathbf{q}_i^T\mathbf{k}_l/\sqrt{d_k})}.$$

Багатоголова увага. Механізм багатоголової уваги (multi-head attention) розширює одноголову увагу шляхом паралельного обчислення H різних «голів уваги», кожна з яких фокусується на різних аспектах вхідної інформації. Математично це формулюється як:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)\mathbf{W}^O,$$

де кожна голова обчислюється як:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

з індивідуальними матрицями проекцій $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ та $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ для кожної голови, та фінальною матрицею $\mathbf{W}^O \in \mathbb{R}^{Hd_v \times d_{\text{model}}}$.

Зазвичай використовується $d_k = d_v = d_{\text{model}}/H$, що забезпечує постійну обчислювальну складність порівняно з одноголовою увагою при збереженні загальної розмірності моделі.

Обчислювальна складність та обмеження. Обчислювальна скла-

дність механізму уваги становить $\mathcal{O}(n^2 d_k + n d_k d_v)$, де домінуючим терміном є $\mathcal{O}(n^2 d_k)$ через необхідність обчислення всіх парних взаємодій запит-ключ. Це створює квадратичну залежність від довжини послідовності, що обмежує масштабованість архітектури для довгих текстів.

Просторова складність пам'яті для зберігання матриці уваги також становить $\mathcal{O}(n^2)$, що може стати критичним обмеженням для послідовностей довжиною понад кілька тисяч токенів.

Позиційне кодування. Оскільки механізм самоуваги є інваріантним відносно перестановок (permutation-invariant), для врахування порядку елементів у послідовності необхідне позиційне кодування. Класичний підхід використовує детерміністичні синусоїдальні функції.

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right),$$

де $pos \in \{0, 1, \ldots, n-1\}$ – абсолютна позиція токена;
$i \in \{0, 1, \ldots, \lfloor d_{\text{model}}/2 \rfloor - 1\}$ – індекс розмірності.

Ключові властивості синусоїдального кодування включають:

1. Унікальність: кожна позиція має унікальне представлення.
2. Детермінованість: кодування не залежить від навчальних параметрів.
3. Відносна позиційна інформація: PE_{pos+k} може бути виражено як лінійна комбінація PE_{pos}.
4. Екстраполяція: здатність узагальнювати на послідовності, довші за ті, що використовувалися під час навчання.

2.1.2 Архітектурні особливості декодерних моделей

Декодерні архітектури трансформерів, які лягли в основу сучасних великих мовних моделей, характеризуються використанням каузального (одно-

направленого) механізму самоуваги. На відмінну від енкодерних архітектур з двонаправленою увагою, декодери обмежують доступ кожного токена лише до попередніх позицій у послідовності.

Каузальне маскування. Каузальна увага реалізується через застосування нижньої трикутної маски до матриці оцінок уваги. Формально, маска визначається як:

$$\mathbf{M}_{ij} = \begin{cases} 0, & \text{якщо } j \leq i \\ -\infty, & \text{якщо } j > i \end{cases}.$$

Каузальна увага тоді обчислюється як:

$$\text{Attention}_{\text{causal}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}.$$

Застосування $-\infty$ у масці забезпечує, що після операції softmax відповідні елементи матриці уваги дорівнюють нулю, ефективно блокуючи інформацію з майбутніх позицій.

Авторегресивна генерація. Декодерна архітектура дозволяє навчання та інференс в авторегресивному режимі, де модель генерує послідовність токен за токеном:

$$P(x_1, x_2, \ldots, x_n) = \prod_{t=1}^{n} P(x_t | x_1, x_2, \ldots, x_{t-1}).$$

Під час інференсу, наступний токен визначається як:

$$\hat{x}_{t+1} = \arg\max_{v \in \mathcal{V}} P(v | x_1, \ldots, x_t),$$

де $\mathcal{V}$ – словник моделі.

2.1.3 Архітектурні інновації сучасних моделей

Rotary Position Embedding (RoPE). Rotary Position Embedding, запроваджений у роботі [18], представляє альтернативний підхід до позиційного кодування, що інтегрує позиційну інформацію безпосередньо в обчислення скалярного добутку через геометричні обертання у багатовимірному просторі.

Для двовимірного випадку, RoPE застосовує матрицю обертання:

$$\mathbf{R}_m^{(2)} = \begin{pmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{pmatrix},$$

де m – позиція токена;

θ – базова частота.

Для d-вимірного випадку (d парне), матриця має блочно-діагональну структуру:

$$\mathbf{R}_{\Theta,m}^{(d)} = \begin{pmatrix} \mathbf{R}_m^{(2)}(\theta_1) & & & \\ & \mathbf{R}_m^{(2)}(\theta_2) & & \\ & & \ddots & \\ & & & \mathbf{R}_m^{(2)}(\theta_{d/2}) \end{pmatrix},$$

де $\theta_i = 10000^{-2(i-1)/d}$ для $i = 1, 2, \ldots, d/2$.

Ключова властивість RoPE полягає у збереженні відносної позиційної інформації в скалярному добутку:

$$\langle \mathbf{R}_{\Theta,m}^{(d)}\mathbf{q}, \mathbf{R}_{\Theta,n}^{(d)}\mathbf{k} \rangle = \langle \mathbf{q}, \mathbf{R}_{\Theta,n-m}^{(d)}\mathbf{k} \rangle.$$

Це дозволяє моделі автоматично кодувати відносні відстані між позиціями незалежно від їхнього абсолютного розташування.

SwiGLU активаційна функція. SwiGLU [19] є варіантом Gated Linear Unit (GLU), що використовує Swish активацію замість традиційної ReLU. Для входу $\mathbf{x} \in \mathbb{R}^d$, SwiGLU обчислюється як:

$$\text{SwiGLU}(\mathbf{x}) = \text{Swish}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1) \odot (\mathbf{x}\mathbf{W}_2 + \mathbf{b}_2),$$

де $\text{Swish}(x) = x \cdot \sigma(\beta x)$, $\sigma(x) = (1 + e^{-x})^{-1}$ – сигмоїдна функція;
β – навчальний параметр (типово $\beta = 1$);
$\odot$ – поелементне множення;
$\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d \times d_{\text{ff}}}$ – матриці ваг.

Повна структура feed-forward блоку з SwiGLU:

$$\text{FFN}_{\text{SwiGLU}}(\mathbf{x}) = \text{SwiGLU}(\mathbf{x})\mathbf{W}_3 + \mathbf{b}_3,$$

де $\mathbf{W}_3 \in \mathbb{R}^{d_{\text{ff}} \times d}$ проектує результат назад до розмірності моделі.

Експериментальні дослідження показують, що SwiGLU демонструє кращу продуктивність порівняно з традиційними активаційними функціями, особливо в задачах мовного моделювання.

RMS Layer Normalization. Root Mean Square Layer Normalization [20] спрощує традиційну нормалізацію шарів шляхом усунення операції центрування:

$$\text{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\text{RMS}(\mathbf{x})} \odot \boldsymbol{\gamma},$$

де $\text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$;
$\boldsymbol{\gamma} \in \mathbb{R}^d$ – навчальні параметри масштабування.

Порівняно зі стандартною Layer Normalization:

$$\text{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$$

де $\mu(\mathbf{x}) = \frac{1}{d}\sum_{i=1}^{d} x_i$;
$\sigma(\mathbf{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu(\mathbf{x}))^2}$.

RMSNorm має декілька переваг:

1) знижені обчислювальні витрати через відсутність обчислення середнього;
2) менша кількість параметрів (відсутність $\boldsymbol{\beta}$);
3) покращена числова стабільність;
4) аналогічна ефективність у більшості практичних застосувань.

Архітектура без зміщень. Сучасні великі мовні моделі, такі як LLaMA та Gemma, повністю виключають параметри зміщення (bias) з лінійних перетворень:

$$\mathbf{y} = \mathbf{x}\mathbf{W}$$

замість традиційного:

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}.$$

Це архітектурне рішення мотивується кількома факторами.

1. Зменшення параметрів: для шару розмірності $d_{\text{in}} \times d_{\text{out}}$ виключення bias зменшує кількість параметрів на d_{out}.
2. Покращена стабільність навчання: RMSNorm ефективно центрує активації, роблячи bias менш необхідними.
3. Спрощення архітектури: менша складність реалізації та потенційно кращі властивості узагальнення.

2.1.4 Дистиляція знань у великих мовних моделях

Дистиляція знань [21] у контексті великих мовних моделей представляє процес передачі знань від великої моделі-учителя до компактнішої моделі-учня. Цей підхід особливо актуальний для створення ефективних версій великих моделей для промислового використання.

Математичні основи дистиляції. Нехай $f_T(\mathbf{x}; \boldsymbol{\theta}_T)$ та $f_S(\mathbf{x}; \boldsymbol{\theta}_S)$ позначають моделі-учителя та учня відповідно. Традиційна дистиляція мінімізує комбіновану функцію втрат:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{CE}}(\mathbf{y}, f_S(\mathbf{x})) + (1 - \alpha)\mathcal{L}_{\text{KD}}(f_T(\mathbf{x}), f_S(\mathbf{x})),$$

де $\mathcal{L}_{\text{CE}}$ – стандартна крос-ентропійна функція втрат;
$\mathcal{L}_{\text{KD}}$ – втрати дистиляції;
$\alpha \in [0, 1]$ – ваговий коефіцієнт.

Функція втрат дистиляції зазвичай формулюється як дивергенція Кульбака-Лейблера між розподілами:

$$\mathcal{L}_{\text{KD}} = \text{KL}(P_T^\tau || P_S^\tau) = \sum_i P_T^{(i)} \log \frac{P_T^{(i)}}{P_S^{(i)}},$$

де температурні розподіли обчислюються як:

$$P_T^{(i)} = \frac{\exp(z_T^{(i)}/\tau)}{\sum_j \exp(z_T^{(j)}/\tau)}$$
$$P_S^{(i)} = \frac{\exp(z_S^{(i)}/\tau)}{\sum_j \exp(z_S^{(j)}/\tau)}.$$

Параметр температури $\tau > 1$ згладжує розподіл імовірностей, роблячи менш впевнені прогнози учителя більш інформативними для учня.

Специфіка дистиляції в авторегресивних моделях. Для авторегресивних мовних моделей дистиляція відбувається на рівні розподілів наступного токена:

$$\mathcal{L}_{\text{AR-KD}} = \sum_{t=1}^{T} \text{KL}(P_T(\cdot|\mathbf{x}_{<t}) || P_S(\cdot|\mathbf{x}_{<t}))$$

Цей підхід використовується в таких моделях, як LLaMA та Gemma, де компактні версії (1B, 3B параметрів) отримуються дистиляцією з більших базових моделей (70B, 300B параметрів).

2.2 Каузальний анмаскінг

2.2.1 Теоретичні основи та мотивація

Каузальний анмаскінг (causal unmasking) представляє інноваційний підхід до адаптації авторегресивних декодерних моделей для задач, що традиційно вимагають двонаправленого контексту. Основна ідея полягає у контрольованому знятті каузальних обмежень під час спеціалізованого навчання, дозволяючи моделі використовувати інформацію з майбутніх позицій.

Формально, якщо стандартна каузальна увага обмежується маскою $\mathbf{M}_{\text{causal}}$, то анмаскована увага використовує модифіковану маску $\mathbf{M}_{\text{unmasked}}$:

$$\mathbf{M}_{\text{unmasked},ij} = \begin{cases} 0, & \text{для всіх дозволених пар } (i,j) \\ -\infty, & \text{для заборонених пар } (i,j) \end{cases}.$$

Градуальний анмаскінг. Пряме перемикання з каузальної на двонаправлену увагу може призвести до катастрофічного забування. Тому може бути застосовано градуальний підхід:

$$\mathbf{M}_{ij}^{(k)} = \begin{cases} 0, & \text{якщо } j \leq i + k \\ -\infty, & \text{якщо } j > i + k \end{cases},$$

де k поступово збільшується від 0 (повністю каузальна) до $n - 1$ (повністю двонаправлена).

На практиці градуальний анмаскінг застосовується рідко. Натомість аналогічні результати досягаються використанням методів warmup learning rate, які дозволяють моделі поступово адаптуватися до нових умов навчання без різких змін архітектури уваги. Крім того, механізм LoRA (Low-Rank Adaptation) природним чином забезпечує стабільний перехід до нових задач без катастрофічного забування (див. 2.3.2).

2.2.2 Задача Masked Language Modeling

Masked Language Modeling (MLM) прямо застосовується для декодерних архітектур без урахуванням їхніх особливостей. Нехай $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ – вхідна послідовність, а $\mathcal{M} \subseteq \{1, 2, \ldots, n\}$ – множина індексів маскованих позицій.

Модифікована послідовність утворюється як:

$$\tilde{x}_i = \begin{cases} [\text{MASK}], & \text{якщо } i \in \mathcal{M} \\ x_i, & \text{інакше} \end{cases}.$$

Функція втрат MLM для декодерної моделі:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log P_\theta(x_i | \tilde{\boldsymbol{x}}, \text{pos} = i).$$

Стратегії маскування. Вибір стратегії маскування критично впливає на ефективність навчання моделей.

1. Випадкове маскування: кожен токен маскується незалежно з імовірністю p (зазвичай $p = 0.15$). Формально:

$$P(i \in \mathcal{M}) = p, \quad \forall i \in \{1, 2, \ldots, n\},$$

   Цей підхід забезпечує рівномірний розподіл складності, але може призвести до фрагментації контексту.

2. Блочне маскування: маскуються суміжні блоки токенів довжини $l \sim \text{Poisson}(\lambda)$. Очікувана довжина блоку:

$$\mathbb{E}[l] = \lambda, \quad P(l = k) = \frac{\lambda^k e^{-\lambda}}{k!},$$

   Такий підхід краще моделює природні паттерни пропусків у тексті та підвищує складність завдання.

3. Структурне маскування: маскування відбувається з урахуванням лінгвістичної структури (слова, фрази, речення). Імовірність маскування залежить від синтаксичної ролі елемента:

$$P(i \in \mathcal{M}) = f(\text{POS}(x_i), \text{DEP}(x_i), \text{NER}(x_i)),$$

де POS – частиномовна сутність;
DEP – синтаксична сутність;
NER – іменована сутність.

2.2.3 Задача Masked Next Token Prediction

MNTP представляє адаптацію MLM, специфічно розроблену для декодерних архітектур. Ключова відмінність полягає у зсуві позиції прогнозування:

$$\mathcal{L}_{\text{MNTP}} = -\sum_{i \in \mathcal{M}, i>1} \log P_\theta(x_i | \tilde{\boldsymbol{x}}, \text{pos} = i-1).$$

Це дозволяє використовувати існуючу архітектуру мовної голови без додаткових модифікацій.

Теоретичне обґрунтування зсуву. Зсув прогнозування дозволяє моделі інтегрувати інформацію про маскований токен x_i при формуванні представлення для позиції $i-1$, використовуючи при цьому стандартний авторегресивний механізм генерації.

Математично, це еквівалентно оптимізації:

$$\arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x},\mathcal{M}} \left[\sum_{i \in \mathcal{M}} \log P_\theta(x_i | \mathbf{h}_{i-1}(\tilde{\boldsymbol{x}})) \right],$$

де $\mathbf{h}_{i-1}(\tilde{\boldsymbol{x}})$ – скрите представлення для позиції $i-1$ з урахуванням маскованої послідовності.

2.3 Тренування моделей за обмежених ресурсів

Навчання великих мовних моделей (Large Language Models, LLMs) потребує значних обчислювальних ресурсів. За останні роки моделі досягли значних розмірів (напр. LLaMA 4 із 2T параметрів).

Адаптуючись до таких змін, дослідницька спільнота розробила численні методи ефективного навчання націлені на оптимізацію пам'яті (квантування, pruning), адаптивне навчання (LoRA, AdaLoRA) та архітектурні інновації (MoE, sparse attention).

2.3.1 Квантування

Квантування представляє фундаментальний метод компресії нейронних мереж, який зменшує точність представлення параметрів моделі з метою економії пам'яті та прискорення обчислень [22; 23]. Ідейною основою квантування є принцип того, що багато параметрів нейронних мереж мають надлишкову точність, яка не є критичною для збереження їх функціональності.

Теоретичні основи квантування. Математично, квантування можна розглядати як відображення з простору дійсних чисел високої точності у простір цілих чисел обмеженої розрядності [24]:

$$Q : \mathbb{R} \to \mathbb{Z}_b = \{-2^{b-1}, -2^{b-1}+1, \ldots, 2^{b-1}-1\},$$

де b – кількість біт для представлення квантованого значення.

Фундаментальна теорема квантування стверджує, що для випадкової величини X з обмеженою дисперсією σ^2, мінімальна середньоквадратична помилка квантування досягається при використанні оптимального квантувача, що мінімізує [25]:

$$\text{MSE} = \mathbb{E}[(X - Q(X))^2].$$

Типи квантування за часом застосування.

1. Post-training quantization (PTQ) – квантування після завершення навчання без додаткової оптимізації параметрів. Цей підхід є найпростішим у реалізації, але може призводити до більших втрат якості.
2. Quantization-aware training (QAT) – навчання з урахуванням майбутнього квантування, коли квантизаційний шум моделюється під час forward pass для адаптації моделі до низької точності.
3. Dynamic quantization – квантування активацій динамічно під час inference з використанням статистик поточного batch'у.

Типи квантування за точністю представлення.

1. FP32 $\rightarrow$ INT8 квантування. Найпоширеніший підхід, що забезпечує 4x зменшення пам'яті:

$$w_{\text{int8}} = \text{round}\left(\frac{w_{\text{fp32}} - z}{s}\right),$$

де s – масштабуючий фактор;
z – zero-point.

2. INT4 квантування. Додатково зменшує розмір в 2 рази, часто використовується з груповаим квантуванням для збереження точності.
3. 1-bit квантування (бінаризація). Екстремальний випадок:

$$w_{\text{bin}} = \text{sign}(w_{\text{fp32}}) \cdot \alpha,$$

де α – додатковий масштабуючий фактор для компенсації втрат.

4. Mixed-precision quantization. Різні шари мають різну точність залежно від їх чутливості до квантизаційного шуму.

Алгоритм симетричного квантування. Симетричне квантування припускає, що розподіл ваг є приблизно симетричним відносно нуля. Для тензора ваг $\mathbf{W}$ з елементами $w_i \in [w_{\min}, w_{\max}]$.

1. Обчислення масштабуючого фактора:

$$s = \frac{\max(|w_{\min}|, |w_{\max}|)}{2^{b-1} - 1},$$

де b – кількість біт для представлення.

2. Квантування з обмеженням діапазону:

$$w_i^{(q)} = \text{clamp}\left(\text{round}\left(\frac{w_i}{s}\right), -2^{b-1}, 2^{b-1} - 1\right),$$

3. Деквантування для обчислень:

$$\tilde{w}_i = w_i^{(q)} \cdot s.$$

Асиметричне квантування. Асиметричне квантування є більш загальним підходом, що не робить припущень про симетричність розподілу і дозволяє кращого використання доступного діапазону представлення:

$$w_i^{(q)} = \text{clamp}\left(\text{round}\left(\frac{w_i - z \cdot s}{s}\right), 0, 2^b - 1\right),$$

де параметри обчислюються як:

$$s = \frac{w_{\max} - w_{\min}}{2^b - 1}$$
$$z = \text{round}\left(\frac{-w_{\min}}{s}\right).$$

Zero-point z представляє значення у квантованому просторі, що відповідає нулю в оригінальному просторі.

Груповане квантування. Для збереження точності при агресивному квантуванні (наприклад, INT4) використовується груповане квантування, де тензор поділяється на групи розміром G:

$$\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_{N/G}].$$

Кожна grupa має власні параметри квантування (s_j, z_j), що дозволяє

краще адаптуватися до локальних статистик розподілу ваг.

Аналіз втрати якості та квантизаційного шуму. Квантування неминуче призводить до погіршення якості моделі через введення квантизаційного шуму:

$$\epsilon_i = \tilde{w}_i - w_i.$$

Для рівномірного квантування з кроком s, квантизаційна помилка має рівномірний розподіл на інтервалі $[-s/2, s/2]$, що дає очікувану квадратичну помилку:

$$\mathbb{E}[\epsilon_i^2] = \frac{s^2}{12}$$

Теорема про накопичення квантизаційного шуму. Для глибокої мережі з L шарами, загальна дисперсія шуму на виході приблизно дорівнює:

$$\sigma^2_{\text{total}} \approx \sum_{l=1}^{L} \sigma_l^2 \prod_{j=l+1}^{L} \|\mathbf{W}_j\|_2^2,$$

де σ_l^2 – дисперсія квантизаційного шуму в l-тому шарі.

Експериментальні дослідження на великих мовних моделях показують наступні закономірності:

1) INT8 квантування: втрата точності $< 1\%$ для більшості задач;
2) INT4 квантування: втрата точності 2-5% залежно від архітектури;
3) INT4 з груповим квантуванням (G=128): втрата точності 1-3%;
4) змішана точність (FP16 для attention, INT8 для FFN): $< 0.5\%$ втрати.

2.3.2 LoRA та її варіанти

LoRA. Low-Rank Adaptation (LoRA) [26] представляє парадигмальний зсув у підході до fine-tuning великих моделей, базуючись на фундаментальній

гіпотезі про те, що зміни ваг під час адаптації до конкретної задачі мають внутрішню низькорангову структуру.

Математично, замість оновлення повної матриці ваг $\mathbf{W} \in \mathbb{R}^{d\times k}$, LoRA представляє зміни у вигляді низькорангової декомпозиції:

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{BA},$$

де $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$ – заморожені претреновані ваги;
$\mathbf{B} \in \mathbb{R}^{d\times r}$, $\mathbf{A} \in \mathbb{R}^{r\times k}$ – навчальні низькорангові матриці;
$r \ll \min(d, k)$ – ранг адаптації, гіперпараметр що контролює компроміс між ефективністю та якістю.

Формальний опис алгоритму покроково.

1. Ініціалізація параметрів:

$$\mathbf{A} \sim \mathcal{N}(0, \sigma^2), \quad \sigma^2 = \frac{1}{r}$$
$$\mathbf{B} = \mathbf{0}_{d\times r}.$$

Така ініціалізація забезпечує, що $\Delta\mathbf{W} = \mathbf{0}$ на початку навчання, зберігаючи поведінку претренованої моделі.

2. Forward pass з масштабуванням:

$$\mathbf{h} = \mathbf{xW}_0 + \mathbf{xBA} \cdot \frac{\alpha}{r},$$

де α – масштабуючий параметр, що дозволяє контролювати вплив LoRA адаптації незалежно від рангу r.

3. Навчання лише LoRA параметрів:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{A}}, \frac{\partial\mathcal{L}}{\partial\mathbf{B}} \neq 0, \quad \frac{\partial\mathcal{L}}{\partial\mathbf{W}_0} = 0.$$

QLoRA. Quantized LoRA (QLoRA) [27] представляє синергетичне поєднання квантування базової моделі з LoRA адаптерами, дозволяючи досягти максимальної ефективності пам'яті.

1. 4-bit квантування базових ваг:

$$\mathbf{W}_0^{(q)} = \text{Quantize}(\mathbf{W}_0, \text{NF4}),$$

де NF4 (Normal Float 4) – спеціальний формат квантування, оптимізований для нормально розподілених ваг.

2. Обчислення з деквантуванням у FP16/BF16:

$$\mathbf{h} = \mathbf{x}\text{Dequantize}(\mathbf{W}_0^{(q)}) + \mathbf{xBA} \cdot \frac{\alpha}{r}.$$

3. Подвійне квантування градієнтів: Для додаткової економії пам'яті, градієнти також квантуються:

$$\nabla_{\mathbf{A}}^{(q)} = \text{Quantize}(\nabla_{\mathbf{A}}, \text{INT8}).$$

2.4 Оцінювання та метрики

В попередніх дослідженнях, спрямованих на двонаправлена адаптацію декодерів (див. 1.3), явно не вводилися метрики для оцінки саме двонаправленості. Основною мірою якості залишалися метрики та їх приріст на потребуючих двонаправленості задачах. В рамках цієї роботи, спираючись на сучасні дослідження у сфері інтерпретовності машиного навчання, було розглянуто новітні метрики симетричності та «направленості» матриці уваги.

2.4.1 Метрики якості маркування послідовності

Для задач sequence labeling традиційно використовуються класифікаційні метрики – точність (precision), повнота (recall) та їх гармонійне середнє

F_1. Обчислення виконується щодо правильного виділення міток у послідовності. Математично ці метрики визначаються наступним чином:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN},$$

$$\mathrm{F}_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}},$$

де TP (True Positives) – кількість правильно ідентифікованих позитивних випадків;

FP (False Positives) – кількість хибно позитивних;

FN (False Negatives) – кількість пропущених позитивних випадків.

Проте якість, залежно від задачі (див. 1.2.1) можна вимірювати на різних рівнях деталізації, кожен з яких має свої переваги та обмеження.

1. На рівні токенів (Token-level evaluation). Вважаємо кожну позицію окремим класифікаційним випадком: істинна мітка vs передбачена. Цей підхід є найбільш гранулярним і дозволяє отримати детальну інформацію про помилки моделі. Для послідовності довжиною n з k класами міток, загальна кількість класифікаційних рішень становить n. Перевагою цього підходу є можливість отримання часткового зарахування за правильне розпізнавання частин сутностей. Недоліком може бути завищення оцінки для задач, де важливою є цілісність виділених фрагментів.
2. На рівні цілих інтервалів (Span-level evaluation). Використовується для задач типу NER, одиниця оцінки – весь інтервал токенів, що утворює сутність, разом з її типом. Правильним вважається тільки повний збіг інтервалу і мітки з еталоном. Часткові збіги зазвичай не рахуються як успіх.

   Формально, для сутності $e_i = (start_i, end_i, type_i)$ та передбачення $\hat{e}_j = (\hat{start}_j, \hat{end}_j, \hat{type}_j)$, збіг визначається як:

$$\text{match}(e_i, \hat{e}_j) = \mathbb{I}[start_i = \hat{start}_j \wedge end_i = \hat{end}_j \wedge type_i = \hat{type}_j]$$

Ця метрика більш сувора: модель отримує бал лише якщо маркування безпомилкове. Тому значення F_1 на рівні сутностей зазвичай нижче, ніж на рівні токенів (де модель могла б отримати частковий залік). Цей підхід краще відображає практичну корисність системи, оскільки частково виділені сутності часто не мають цінності для кінцевого користувача.

3. На рівні символів (Character-level evaluation). Інколи застосовується, коли межі сутностей можуть не збігатися з токенами або цікавить часткове перекриття. Суть: кожен символ тексту вважається одиницею, поміченою або ні.

   Для тексту довжиною L символів з бінарною розміткою $y \in \{0,1\}^L$ та передбаченням $\hat{y} \in \{0,1\}^L$, char-level F_1 обчислюється стандартним способом:

$$\text{char-F}_1 = \frac{2\sum_{i=1}^{L} y_i \hat{y}_i}{\sum_{i=1}^{L} y_i + \sum_{i=1}^{L} \hat{y}_i}.$$

   Підхід особливо корисний для задач, де важливе точне визначення меж, незалежно від токенізації.

Вибір рівня оцінювання залежить від специфіки задачі та вимог до системи. Для наукових досліджень часто використовують кілька рівнів одночасно, щоб отримати повне уявлення про якість моделі. У контексті адаптації декодерів особливо важливим є порівняння результатів на різних рівнях до та після донавчання, оскільки це може виявити специфічні аспекти покращення двонаправленої обробки.

2.4.2 Міра двонаправленості

Нещодавно було проведено фундаментальний аналіз навчання самоуваги для різних режимів тренування – авторегресивного (каузального) та маскованого (двонаправленого). У роботі [28] показано, що двонаправлене на-

вчання породжує симетричні структури у матрицях ваг самоуваги, тоді як авторегресивне – направлені, із домінуванням стовпців.

З теоретичного боку, цей результат має глибоке обґрунтування. При каузальному навчанні модель може використовувати інформацію лише з попередніх позицій, що призводить до асиметричних патернів уваги. Навпаки, при двонаправленому навчанні модель має доступ до повного контексту, що сприяє формуванню більш збалансованих та симетричних структур уваги.

Інтуїтивно це означає: при двонаправленій увазі модель вчиться приділяти рівномірну увагу контексту з обох боків, що проявляється як симетричність у матрицях ваг ($w_{ij} \approx w_{ji}$). При каузальній увазі формується перевага «дивитися назад» (на попередні токени), і деякі токени набувають особливо великої «ваги» як ключі (вони привертають увагу багатьох інших, що проявляється в матрицях ваг як стовпці великої норми).

Формальні визначення. Для формалізації введемо матрицю ваг $\mathbf{W} \in \mathbb{R}^{d\times d}$ певної голови самоуваги і її властивості. Автори дають точні визначення двох ключових коефіцієнтів: коефіцієнт симетрії $\mathcal{S}(\mathbf{W})$ та коефіцієнт напрямленості $\mathcal{D}(\mathbf{W})$.

Коефіцієнт симетрії. Симетрію визначають через канонічний розклад матриці $\mathbf{W}$ на симетричну і кососиметричну компоненти:

$$\mathbf{W}^{(s)} = \frac{\mathbf{W} + \mathbf{W}^T}{2} \quad \text{(симетрична компонента)},$$
$$\mathbf{W}^{(a)} = \frac{\mathbf{W} - \mathbf{W}^T}{2} \quad \text{(кососиметрична компонента)}.$$

Тоді коефіцієнт симетрії визначається як:

$$\mathcal{S}(\mathbf{W}) = \|\mathbf{W}^{(s)}\|_F^2 \; - \; \|\mathbf{W}^{(a)}\|_F^2 \,,$$

де $\|\cdot\|_F$ – норма Фробеніуса, що обчислюється як $\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$.

Геометрично, $\mathcal{S}(\mathbf{W})$ вимірює «баланс» між симетричною та антисиметричною складовими матриці. Додатне значення $\mathcal{S} > 0$ означає переважання симетричної складової (матриця близька до симетричної, що характерно для двонаправленої уваги), від'ємне значення $\mathcal{S} < 0$ – переважання кососиметрії

(що може свідчити про направлену, каузальну структуру уваги).

Коефіцієнт напрямленості. Напрямленість $\mathcal{D}(\mathbf{W})$ оцінює дисбаланс між впливом рядків та стовпців у матриці ваг, що відображає різні режими функціонування уваги.

Спочатку обчислюємо евклідові норми рядків та стовпців:

$$\|\mathbf{W}_{:j}\|_2 = \sqrt{\sum_i w_{ij}^2} \quad \text{(норма } j\text{-го стовпця)}$$
$$\|\mathbf{W}_{i:}\|_2 = \sqrt{\sum_j w_{ij}^2} \quad \text{(норма } i\text{-го рядка)}$$

Далі визначаємо статистики розподілу норм:

$$c_\mu = \frac{1}{d}\sum_{j=1}^{d} \|\mathbf{W}_{:j}\|_2, \quad c_\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(\|\mathbf{W}_{:j}\|_2 - c_\mu)^2}$$
$$r_\mu = \frac{1}{d}\sum_{i=1}^{d} \|\mathbf{W}_{i:}\|_2, \quad r_\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\|\mathbf{W}_{i:}\|_2 - r_\mu)^2}.$$

Використовуючи поріг λ стандартних відхилень (зазвичай $\lambda = 1$ або $\lambda = 2$), виділяємо "важкі" стовпці та рядки:

$$C = \sum_{j:\|\mathbf{W}_{:j}\|_2 > c_\mu + \lambda c_\sigma} \|\mathbf{W}_{:j}\|_2 \quad \text{(сумарна норма важких стовпців)}$$
$$R = \sum_{i:\|\mathbf{W}_{i:}\|_2 > r_\mu + \lambda r_\sigma} \|\mathbf{W}_{i:}\|_2 \quad \text{(сумарна норма важких рядків).}$$

Тоді коефіцієнт напрямленості визначається як:

$$\mathcal{D}(\mathbf{W}) = \frac{R - C}{R + C}.$$

Інтерпретація коефіцієнта напрямленості.

1. $\mathcal{D}(\mathbf{W}) > 0$: домінують рядки великої норми, що означає, що окремі нейрони (позиції запиту) сильно впливають на всі виходи.

Це характерно для моделей, які сильно покладаються на певні «центральні» позиції.

2. $\mathcal{D}(\mathbf{W}) < 0$: домінують стовпці великої норми, що означає column dominance – окремі позиції сильно притягують увагу від багатьох інших позицій. Це типово для каузальних декодерів, де деякі токени (наприклад, початок речення або ключові слова) стають «центрами уваги».
3. $\mathcal{D}(\mathbf{W}) \approx 0$: збалансована структура, характерна для двонаправленої уваги.

Емпіричні спостереження. В експериментах було показано систематичні відмінності між архітектурами.

1. У моделей-енкодерів (BERT, RoBERTa) коефіцієнт симетрії $\mathcal{S}(\mathbf{W})$ значно більший, часто додатний, що відображає симетричну природу двонаправленої уваги.
2. У моделей-декодерів (GPT, LLaMA) коефіцієнт симетрії зазвичай від'ємний або близький до нуля.
3. Коефіцієнт напрямленості $\mathcal{D}(\mathbf{W})$ у декодерів сильно від'ємний (яскраво виражена column dominance), тоді як у енкодерів близький до нуля або трохи додатний.

2.5 Висновки до другого розділу

У розділі було детально розглянуто теоретичні основи трансформерних архітектур та сучасних декодерних моделей, зокрема механізм самоуваги та архітектурні інновації. Проаналізовано принципи дистиляції знань, методи ефективного навчання LLM за обмежених ресурсів, зокрема квантування та LoRA. Окрему увагу приділено задачам каузального анмаскінгу та адаптації маскованого навчання для декодерних моделей. Розглянуто сучасні підходи до оцінювання якості та введено спеціалізовані метрики для аналізу двонаправленості.

3 ЕКСПЕРИМЕНТАЛЬНИЙ ПРОТОКОЛ

3.1 Основні дослідницькі запитання

Робота спрямоване на вирішення трьох ключових запитань у галузі адаптації авторегресивних мовних моделей до двонаправленого розуміння контексту.

Запитання 1: Ефективність двонаправленої адаптації. Чи може авторегресивна модель-декодер успішно адаптуватися до двонаправленого режиму роботи шляхом зняття каузальної маски та донавчання на завданнях маскованого моделювання мови? Це фундаментальне питання стосується архітектурної пластичності трансформерів та можливості перепрофілювання моделей, навчених для генеративних задач, на дискримінативні.

Запитання 2: Порівняння методів адаптації. Який із двох основних підходів до донавчання – Modified Next Token Prediction (MNTP) чи Masked Language Modeling (MLM) – є більш ефективним для досягнення двонаправленості? MNTP зберігає елементи авторегресивності при частковому знятті каузальних обмежень, тоді як MLM повністю переходить до енкодерної парадигми навчання.

Запитання 3: Вплив мови переднавчання. Чи впливає мова корпусу донавчання (англійська vs українська) на ефективність адаптації для задач обробки української мови? Це питання особливо актуальне в контексті cross-lingual transfer learning та оптимізації ресурсів для мов з обмеженими даними.

3.2 Дані та бенчмарки

Для комплексної оцінки ефективності двонаправленої адаптації використовується багаторівнева система оцінювання, що включає як внутрішні метрики (геометричні властивості матриць уваги), так і зовнішні завдання (performance на downstream задачах маркування послідовностей українською мовою).

3.2.1 Українські датасети маркування послідовності

Формалізація критеріїв відбору. При підготовці навчання і оцінки моделей для маркування послідовностей критично важливо використовувати надійні та різноманітні набори даних. Ситуація додатково ускладнюється обмеженістю ресурсів української мови порівняно з англійською. Для структурованого відбору датасетів було сформульовано наступні критерії:

1. Золотий стандарт розмітки. Корпус має бути розмічений виключно експертами-лінгвістами, а не автоматичними алгоритмами чи краудсорсингом. Це особливо важливо, оскільки досліджувані методи претендують на досягнення state-of-the-art результатів у відповідних задачах.
2. Стандартизоване розподілення даних. Публічно зафіксоване розподілення на train/validation/test множини дозволяє точно порівнювати різні підходи та унеможливлює data leakage чи оптимістичну оцінку результатів.
3. Актуальність та дослідницька релевантність. Датасети повинні відображати сучасний стан мови та відповідати поточним викликам у галузі обробки природної мови для української мови.
4. Достатній обсяг даних. Набори даних мають містити достатню кількість прикладів для надійного статистичного аналізу та уни-

кнення overfitting на тестовій множині.

Вибрані датасети. Спираючись на окреслені критерії, було обрано UNLP 2025 Shared Task [29] – найактуальніший датасет для виявлення маніпулятивних технік у цифрових медіа. Містить 9,500 дописів з Telegram, розмічених медіа-експертами згідно з таксономією маніпулятивних прийомів за стандартами European Digital Services Act. Особливість датасету полягає у необхідності character-level маркування маніпулятивних фрагментів, що представляє особливий виклик для моделей та дозволяє оцінити їх здатність до точної локалізації цільових елементів тексту.

3.2.2 Дані для переднавчання

При донавчанні великих мовних моделей критично важливим є вибір якісного та репрезентативного корпусу. Вікіпедія залишається одним з найкращих джерел для цієї мети завдяки:

1) високій якості редагування та фактчекінгу;
2) широкому тематичному охопленню;
3) стандартизованій структурі та форматуванню;
4) доступності для багатьох мов.

Ця робота не стала виключенням, переднавчання моделей з метою адаптації до двонаправленої уваги проводилося на корпусах української та англійської Вікіпедії [30].

3.3 Методологія оцінювання

3.3.1 Оцінка на задачах маркування послідовностей

Для забезпечення порівнянності з існуючими дослідженнями та відповідно до офіційного протоколу змагання UNLP 2025 Shared Task використовується char-level F1-score. Це обумовлено тим, що маніпулятивні фрагменти часто не співпадають з межами токенів або слів (наприклад, маніпулятивними можуть бути лише частини слів, пунктуація, чи специфічні символи).

3.3.2 Оцінка адаптації двонаправленості

Паралельно з оцінкою на downstream задачах досліджується ефективність використання метрик двонаправленості, описаних у розділі 2.4.2, для кількісного вимірювання трансформації внутрішньої структуры моделі після процедури адаптації.

Мотивація підходу. Традиційний підхід до оцінки якості двонаправленості покладається виключно на performance downstream задач, що має обмеження:

1) downstream performance може покращитися з причин, не пов'язаних з набуттям двонаправленості;
2) важко інтерпретувати чому модель стала краще працювати;
3) неможливо відстежити динаміку адаптації в процесі навчання.

Використання внутрішніх метрик дозволяє.

1. Пряме спостереження змін. замість опосередкованих висновків з downstream performance, безпосередньо вимірювати трансформації в архітектурі уваги.
2. Встановлення каузальних зв'язків: якщо покращення на мар-

куванні послідовностей корелює зі збільшенням симетрії ($\mathcal{S}$) та зменшенням домінації окремих колонок ($\mathcal{D}$), це підтверджує гіпотезу про набуття двонаправленості як причину покращення.

3. Гранулярний аналіз: різні шари та голови уваги можуть адаптуватися по-різному, і ці метрики дозволяють ідентифікувати, які компоненти архітектури зазнають найбільших змін.
4. Моніторинг процесу: відстеження метрик під час навчання дає інсайти про динаміку адаптації та оптимальний момент зупинки.

Протокол вимірювань. Для кожної моделі (протягом адаптації) обчислюються коефіцієнти симетрії $\mathcal{S}(\mathbf{W})$ та напрямленості $\mathcal{D}(\mathbf{W})$ згідно з формулами з розділу 2.4.2.

Оскільки адаптація проводиться з використанням LoRA адаптерів, аналіз внутрішніх змін здійснюється на двох рівнях.

1. Рівень LoRA адаптера. Обчислення метрик безпосередньо для низькорангових матриць $\Delta\mathbf{W} = \mathbf{BA}$, де $\mathbf{A} \in \mathbb{R}^{d\times r}$ та $\mathbf{B} \in \mathbb{R}^{r\times d}$ – навчувані матриці адаптера з ранком $r \ll d$. Це дозволяє проаналізувати, які саме трансформації вносить процес адаптації.
2. Рівень повної матриці після додавання адаптера до основних ваг. Обчислення метрик для результуючої матриці $\mathbf{W}_{merged} = \mathbf{W}_{original} + \alpha\Delta\mathbf{W}$, де α – коефіцієнт масштабування LoRA. Це показує кумулятивний ефект адаптації на загальну структуру уваги.

Такий подвійний аналіз дає можливість розділити внески базової моделі та адаптаційних змін, а також оцінити, наскільки ефективно низькорангова адаптація може модифікувати глобальні властивості матриць уваги.

Очікувані зміни та гіпотези. На основі теоретичних передбачень формулюються наступні гіпотези:

1. Гіпотеза симетрії: $\mathcal{S}(\mathbf{W})$ збільшиться від негативних значень (характерних для каузальних декодерів) до додатних (характерних для енкодерів): $\mathcal{S}_{post} > \mathcal{S}_{pre}$, при чому очікується $\mathcal{S}_{pre} < 0$ та $\mathcal{S}_{post} \geq 0$.
2. Гіпотеза збалансованості: $\mathcal{D}(\mathbf{W})$ зросте від сильно від'ємних зна-

чень (column dominance) до ближчих до нуля (збалансована структура): $\mathcal{D}_{post} > \mathcal{D}_{pre}$, при чому $\mathcal{D}_{pre} \ll 0$ та $\mathcal{D}_{post} \approx 0$.

3. Гіпотеза кореляції: зміни у внутрішніх метриках корелюють з покращенням downstream performance: $\Delta\text{F1} \propto (\mathcal{S}_{post} - \mathcal{S}_{pre}) - (\mathcal{D}_{pre} - \mathcal{D}_{post})$.

Обмеження та застереження. Застосування метрик двонаправленості для оцінки адаптації декодер-моделей має кілька принципових обмежень, які важливо враховувати при інтерпретації результатів.

1. Робота [28] встановила ці умови як необхідні для двонаправленості в контексті первинного навчання з нуля на різних режимах (каузальному vs маскованому). Застосування цих метрик до задачі адаптації вже навченої авторегресивної моделі є екстраполяцією, теоретичне обгрунтування якої не було детально досліджено. Процес адаптації може створювати гібридні структури уваги, які не підпадають під класичну дихотомію «енкодер vs декодер».
2. Оригінальне дослідження проводилося виключно на моделях з повною точністю, тоді як в рамках цієї роботи використовувалися прийоми квантизації для оптимізації ресурсів. Квантизація може вносити численні артефакти в матриці ваг, що потенційно впливає на геометричні властивості та може призводити до хибних висновків про ступінь двонаправленості.
3. Використання LoRA адаптерів створює додаткову складність: невідомо, чи зберігають низькорангові трансформації ті ж геометричні властивості, що й повнорангові зміни матриць уваги. Ранг адаптера може обмежувати можливі трансформації структури уваги.

3.4 Конфігурація експериментів

3.4.1 Використане програмне забезпечення та фреймворки

Під час виконання експериментальної частини роботи було ретельно підібрано набір сучасних інструментів для розробки та тестування моделей машинного навчання, зокрема у галузі обробки природної мови. Вибір кожного інструменту обґрунтовувався його надійністю, продуктивністю та широким визнанням у науковій спільноті.

1. PyTorch – провідна бібліотека для глибокого навчання, яка завдяки своїй гнучкості та інтуїтивному API забезпечує зручність реалізації та тренування складних моделей штучного інтелекту. Динамічний характер обчислювального графу PyTorch виявився особливо корисним для експериментування з різними архітектурами.
2. Hugging Face Transformers – найпопулярніша опенсорс бібліотека для роботи з великими мовними моделями, що дозволяє швидко завантажувати, адаптувати та тестувати існуючі архітектури. Особливо цінною виявилася можливість безшовної інтеграції з добре вибудованою Hugging Face інфраструктурою, що включає в себе бібліотеку Datasets для ефективної роботи з великими корпусами текстів.
3. Weights & Biases (W&B) – потужна платформа для логування експериментів, що забезпечила детальне відстеження метрик, візуалізацію процесу навчання та зручне порівняння результатів різних конфігурацій моделей. Інтерактивні дашборди W&B значно спростили аналіз та інтерпретацію експериментальних даних.
4. Hydra – елегантний інструмент для організації конфігурацій екс-

периментів за допомогою YAML файлів, що забезпечив гнучке керування параметрами та дозволив легко запускати різні сценарії експериментів без модифікації основного коду.

3.4.2 Опис обчислювальних ресурсів і експериментального середовища

Всі експерименти проводилися на хмарній платформі Vast AI, яка надала доступ до високопродуктивних графічних процесорів, необхідних для ефективного навчання великих мовних моделей. Вибір хмарної платформи був обумовлений потребою в масштабованості ресурсів та економічною доцільністю порівняно з придбанням власного обладнання.

Для забезпечення повної відтворюваності експериментів та усунення проблем з несумісністю середовищ використовувався єдиний Docker-образ, розміщений на DockerHub[1], який містив усі необхідні залежності та попередньо налаштоване середовище. Такий підхід гарантував ідентичність умов виконання для всіх експериментів незалежно від конкретного апаратного забезпечення.

Основним інтерпретатором обрано Python версії 3.10.12 завдяки його стабільності та широкій підтримці сучасних бібліотек машинного навчання. Для точного керування залежностями та запобігання конфліктам версій використовувався Poetry версії 2.1.3, що забезпечив детермінованість експериментального середовища. Детальну інформацію про версії ключових бібліотек наведено в табл. 3.1.

Обчислювальна інфраструктура була диференційована залежно від складності та ресурсоємності конкретних завдань.

1. Для основних тренувань моделей: використовувалися потужні системи з GPU NVIDIA A100 80GB, що забезпечували достатній обсяг відеопам'яті для навчання великих моделей, доповнені 128

[1] https://hub.docker.com/r/vastai/pytorch/

Таблиця 3.1. Версії основних бібліотек, використаних в експериментах.

Бібліотека	**Версія**
numpy	1.26.4
seaborn	0.12.2
tqdm	4.67.1
torch	2.5.1
lightning	2.5.0
transformers	4.51.3
sentencepiece	0.2.0
datasets	3.3.1
peft	0.14.0
tiktoken	0.9.0
hydra-core	1.3.2
omegaconf	2.3.0
wandb	0.19.10
evaluate	0.4.3
seqeval	1.2.2
bitsandbytes	0.45.5

ГБ оперативної пам'яті та швидким SSD-накопичувачем об'ємом 200 ГБ для мінімізації затримок при завантаженні даних.

2. Для розробки коду та попереднього тестування: застосовувалися більш економічні конфігурації з GPU NVIDIA RTX 3090 та 24 ГБ відеопам'яті, що виявилося достатнім для швидкого прототипування та валідації ідей перед запуском ресурсоємних експериментів.

Особлива увага приділялася забезпеченню детермінованості результатів – на початку кожного експерименту встановлювалися фіксовані seed-значення для всіх компонентів системи, що використовують псевдовипадкові генератори, включаючи PyTorch, NumPy та системні генератори Python.

3.4.3 Використані моделі

Вибір моделей для дослідження базувався на принципі оптимального балансу між продуктивністю, обчислювальною ефективністю та доступністю ресурсів. Після аналізу різних варіантів було обрано дистильовані версії провідних архітектур.

1. Llama 3 [31] — модель `meta-llama/Llama-3.2-3B`, що представляє останні досягнення Meta у галузі великих мовних моделей та демонструє відмінну продуктивність при компактному розмірі;
2. Gemma 3 [32] — модель `google/gemma-3-4b-pt`, розроблена Google та відзначена високою якістю розуміння української мови.

Дистильовані версії цих моделей були обрані свідомо, оскільки вони зберігають основні переваги повнорозмірних варіантів, але потребують значно менших обчислювальних ресурсів, що дозволило провести більшу кількість експериментів та детальніше дослідити різні підходи.

3.4.4 Деталі параметрів переднавчання

Конфігурація етапу переднавчання була ретельно підібрана на основі найкращих практик та попередніх досліджень у галузі адаптації великих мовних моделей. Ключові гіперпараметри включаючі наступні.

1. Квантування. Свідомо вирішено відмовитися від квантованих обчислень на етапі переднавчання для збереження максимальної точності та стабільності процесу навчання. Також з метою уникнення похибок обчислення метрик симетричності та направленості матриць уваги.
2. LoRA (Low-Rank Adaptation). Застосовувалася для кардинального зниження кількості тренованих параметрів без суттєвої втрати якості моделі. Параметри LoRA були налаштовані наступним

чином:

1) розмір low-rank матриць: $r = 64$і;
2) коефіцієнт масштабування: $\alpha = 128$;
3) dropout для LoRA: 0.05;
4) цільові модулі: охоплювали всі ключові компоненти уваги (`q_proj`, `k_proj`, `v_proj`, `o_proj`) та feed-forward мереж (`gate_proj`, `up_proj`, `down_proj`).

3. Параметри оптимізації налаштовані для забезпечення стабільної збіжності:
   1) швидкість навчання: 2×10^{-4} — агресивна, але контрольована швидкість для ефективного навчання;
   2) weight decay: 0.01 — помірна L2-регуляризація;
   3) планувальник: cosine annealing для плавного зниження швидкості навчання;
   4) максимальна кількість кроків: 2000 — достатньо для досягнення збіжності;
   5) розмір батчу: 16 з акумуляцією градієнтів через 2 кроки (фактичний розмір батчу = 32);
   6) оптимізатор: `adamw_8bit` для економії пам'яті без втрати якості.
4. Токенізація. Максимальна довжина обмежена 512 токенами з автоматичним усіченням довших текстів для забезпечення ефективності обчислень.

3.4.5 Постановка експерименту маркування послідовностей

Для забезпечення об'єктивності та повноти аналізу кожен експеримент було структуровано як систематичне порівняння різних підходів. Експериментальна матриця включала наступні конфігурації моделей.

1. Базова SOTA модель — `microsoft/deberta-v3-base` як еталон

для порівняння, що представляє найсучасніші досягнення в енкодерних моделях для задач розуміння мови.

2. Оригінальна каузальна LLM — модель без модифікацій у механізмі уваги, що демонструє базову продуктивність однонаправленого підходу.
3. Двонаправлена версія — модель зі зняття каузального маскування, але без додаткового переднавчання, для ізолювання ефекту зміни архітектури уваги.
4. Повністю адаптовані моделі — розмасковані та претреновані варіанти для всіх комбінацій експериментальних умов, що дозволило детально проаналізувати вплив різних факторів.

Параметри спеціалізованого навчання. Для етапу fine-tuning було обрано більш консервативний підхід:

1. Квантування. Застосовувався 4-bit формат NF4 через бібліотеку BitsAndBytes з обчисленнями в bfloat16 для оптімального балансу між швидкодією та точністю.
2. LoRA-конфігурація зберігалася ідентичною до етапу переднавчання для забезпечення консистентності підходу:
    1) $r = 64$, $\alpha = 128$, dropout $= 0.05$;
    2) ті самі цільові модулі для забезпечення однорідності адаптації.
3. Тренувальний режим:
    1) знижена швидкість навчання: 2×10^{-5} — для запобігання катастрофічному забуванню;
    2) cosine scheduler без warm-up для стабільної оптимізації;
    3) п'ять епох навчання — достатньо для адаптації без перенавчання;
    4) диференційовані розміри батчів: 16 для навчання, 4 для валідації через обмеження пам'яті;
    5) оптимізатор `adamw_torch` у форматі `bf16` для максимальної стабільності.

4. Токенізація. Максимальна довжина збільшена до 1 024 токенів для кращого охоплення контексту в задачах маркування.

3.5 Висновки до третього розділу

У розділі детально описано експериментальний протокол дослідження двонаправленої адаптації авторегресивних мовних моделей. Сформульовано ключові дослідницькі запитання, визначено критерії відбору даних та обрано релевантні українські і мультимовні корпуси. Описано підхід до оцінювання на основі зовнішніх (char-level F1) і внутрішніх метрик (симетрія та напрямленість матриць уваги), а також наведено гіпотези щодо очікуваних змін. Узагальнено програмні інструменти, конфігурацію обчислювального середовища та параметри моделей, що забезпечили відтворюваність і об'єктивність експериментів.

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Аналіз трансформації двонаправленості

4.1.1 Динаміка внутрішніх метрик під час переднавчання

Протягом усього процесу адаптації здійснювався систематичний моніторинг метрик симетричності $\mathcal{S}(\mathbf{W})$ та направленості $\mathcal{D}(\mathbf{W})$ матриць уваги на кожному шарі трансформера. Оскільки адаптація проводилась із використанням LoRA-адаптерів, аналіз здійснювався на двох рівнях (див. 3.3.2):

1. Рівень об'єднаних ваг: $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_{\text{LoRA}}$, де $\mathbf{W}_0$ представляють базові ваги оригінальної моделі;
2. Рівень LoRA-компоненти: окремий аналіз адаптаційних матриць $\Delta\mathbf{W}_{\text{LoRA}}$.

На рисунках 4.1 та 4.2 зображено еволюцію метрик симетричності та направленості для об'єднаних матриць уваги протягом етапу переднавчання. Де лінія показує середнє значення по шарах, зафарбована область — інтервал $\pm 1\sigma$.

Паралельно аналізувались метрики безпосередньо для LoRA-адаптерів, результати якого представлено на рисунках 4.3 та 4.4.

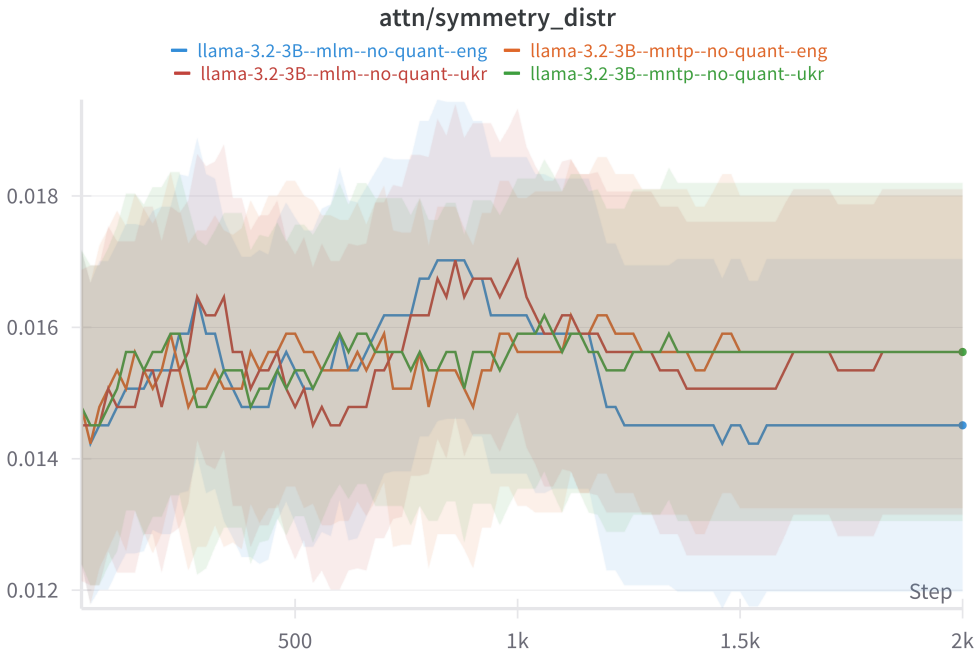


Рисунок 4.1. Еволюція $\mathcal{S}$ на об'єднаних матрицях $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_{\text{LoRA}}$ протягом етапу переднавчання.

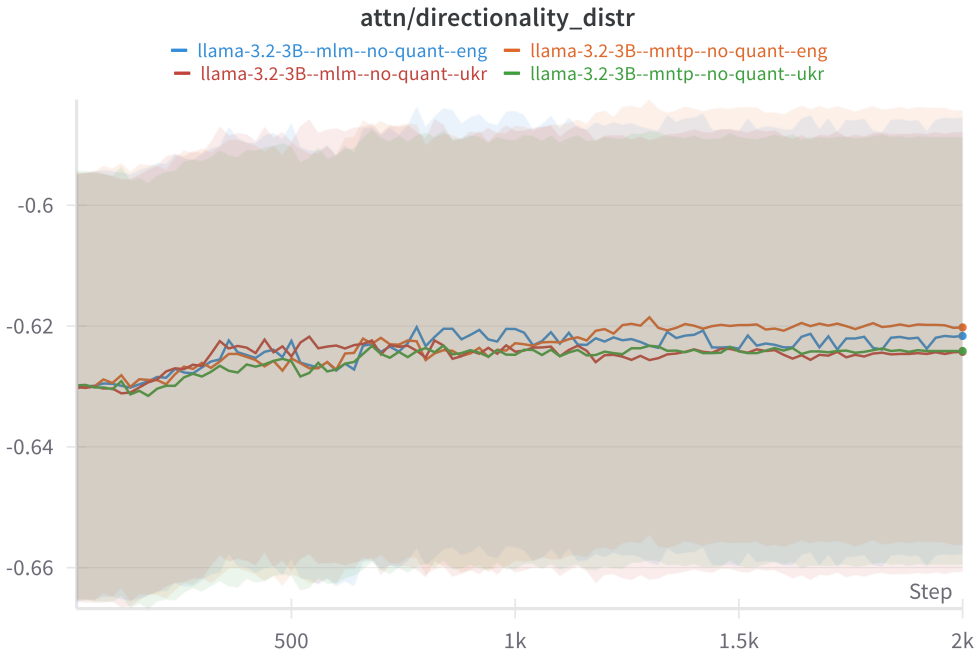


Рисунок 4.2. Еволюція $\mathcal{D}$ на об'єднаних матрицях $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_{\text{LoRA}}$ протягом етапу переднавчання.

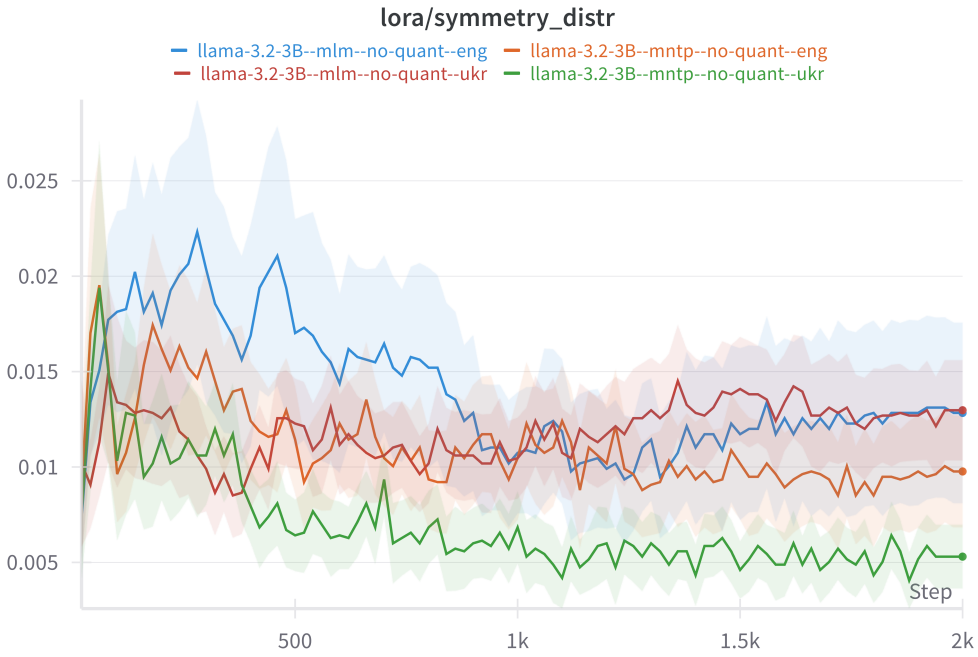


Рисунок 4.3. Еволюція $\mathcal{S}$ на LoRA-адаптерах $\Delta\mathbf{W}_{\text{LoRA}}$ протягом етапу переднавчання.

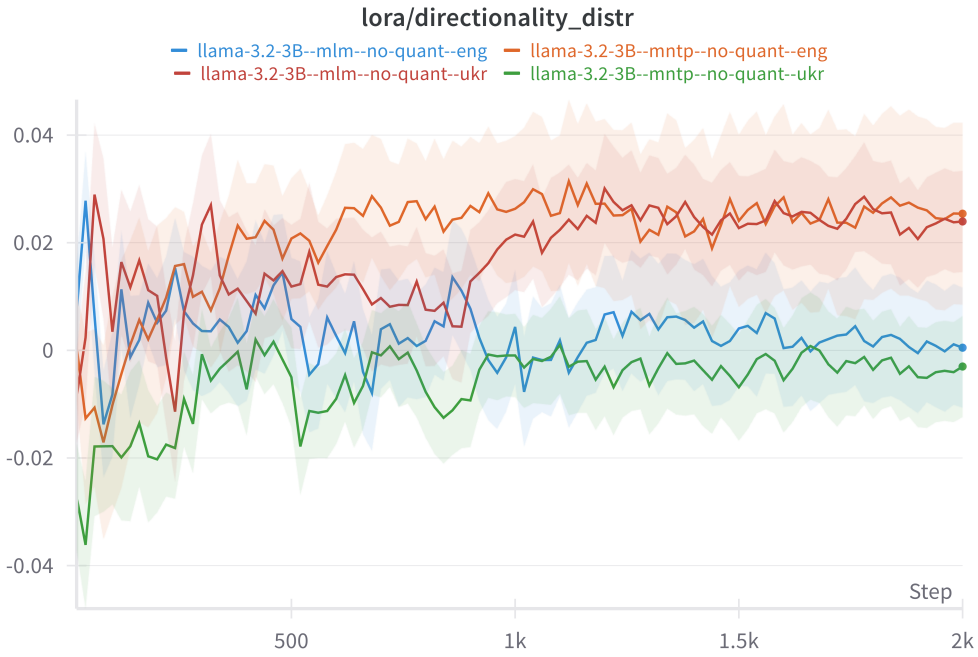


Рисунок 4.4. Еволюція $\mathcal{D}$ на LoRA-адаптерах $\Delta\mathbf{W}_{\text{LoRA}}$ протягом етапу переднавчання.

4.1.2 Результати спостереження метрик двонаправленості

Аналіз динаміки внутрішніх метрик під час етапу переднавчання виявив ряд несподіваних закономірностей, які суперечать початковим теоретичним очікуванням.

Стабільність метрик протягом навчання. На відміну від очікуваної еволюції у бік збільшення симетричності та зменшення направленості, обидві метрики демонстрували виражену стабільність протягом усього процесу адаптації. Це спостерігалося незалежно від:

1) типу навчальної задачі (MLM vs MNTP);
2) мовної специфіки корпусу (українська vs англійська);
3) глибини шару в архітектурі мережі.

Відсутність чутливості як критерію зупинки. Метрики $\mathcal{S}(\mathbf{W})$ та $\mathcal{D}(\mathbf{W})$ не продемонстрували достатньої варіативності для використання в якості надійного сигналу раннього припинення навчання. Це поставило під сумнів їхню практичну застосовність як автоматичного критерію збіжності процесу адаптації.

Вибір фінального чекпоїнту. З огляду на неінформативність динаміки внутрішніх метрик, для подальшого аналізу було обрано фінальні стани моделей після повного циклу переднавчання на 2000 кроків.

4.1.3 Критичне переосмислення теоретичних передбачень

Отримані результати вказують на суттєві обмеження в застосуванні існуючих метрик двонаправленості до контексту адаптації попередньо навчених авторегресивних моделей:

Питання достатності умов. Хоча дослідження [28] встановило математичні умови як необхідні для двонаправленості, питання їхньої **достатності** для гарантування справжньої двонаправленої поведінки залишається

відкритим та потребує додаткового дослідження.

Специфіка адаптаційного процесу. Процес адаптації може створювати гібридні структури уваги, які не підпадають під класичну дихотомію «енкодер vs декодер» та можуть вимагати розробки специфічних метрик оцінювання.

4.2 Емпіричне тестування на задачах маркування послідовностей

4.2.1 Результати на задачі UNLP 2025

Систематичне порівняння різних конфігурацій моделей на задачі виявлення маніпулятивних фрагментів тексту продемонструвало закономірності, представлені в таблиці 4.1.

Таблиця 4.1. Порівняльні результати на датасеті UNLP 2025 Shared Task.

Модель	**Переднавчання**	**Test F_1**
biLlama-3.2-3B	MNTP (Ukrainian)	**0.624**
biLlama-3.2-3B	MLM (Ukrainian)	0.621
biLlama-3.2-3B	MNTP (English)	0.616
biLlama-3.2-3B	MLM (English)	0.612
biLlama-3.2-3B	-	0.617
Llama-3.2-3B	-	0.593
biGemma-3-4B	MNTP (Ukrainian)	**0.633**
biGemma-3-4B	MLM (Ukrainian)	**0.633**
biGemma-3-4B	MNTP (English)	**0.633**
biGemma-3-4B	MLM (English)	0.629
biGemma-3-4B	-	0.539
Gemma-3-4B	-	0.605
mDeBERTaV3	-	0.604

4.2.2 Ключові спостереження

Ефективність архітектурної модифікації. Найбільш значущим результатом є демонстрація того, що навіть «порожня» двонаправлена версія без жодного додаткового переднавчання (biLlama-3.2-3B з результатом 0.617) суттєво перевищує як оригінальну однонаправлену модель (0.593), так і спеціалізовану енкодерну архітектуру mDeBERTaV3 (0.604). Цей результат підтверджує фундаментальну гіпотезу про важливість двонаправленого контексту для задач розуміння тексту.

Вплив мовної специфічності переднавчання Llama. Додаткове переднавчання на україномовному корпусі демонструє стабільне покращення результатів (+0.007 п.п. для MNTP, +0.004 п.п. для MLM порівняно з базовою двонаправленою версією). Натомість, використання англійського корпусу не лише не приносить користі, але навіть дещо погіршує показники (-0.001 п.п. для MNTP, -0.005 п.п. для MLM).

Контрастуючі результати архітектури Gemma. Експерименти з biGemma-3-4B демонструють принципово відмінну картину мовної адаптації. Зокрема, не спостерігається значних відмінностей між результатами додаткового переднавчання на українському (0.633 для MNTP і MLM) та англійському (0.633 для MNTP, 0.629 для MLM) корпусах. Дана особливість може пояснюватися початковим якісним мультилінгвальним переднавчанням базової моделі Gemma-3-4B, яке забезпечило кращу мовну універсальність порівняно з Llama-3.2-3B. Це припущення підтверджується і тим фактом, що базова Gemma-3-4B (0.605) показує кращі результати порівняно з Llama-3.2-3B (0.593) навіть без архітектурних модифікацій.

Критична важливість двонаправленості для Gemma. Водночас, результати biGemma-3-4B без додаткового переднавчання (0.539) демонструють найбільш драматичне падіння продуктивності серед усіх досліджуваних конфігурацій. Цей контраст підкреслює критичну важливість відповідного переднавчання для ефективного функціонування модифікованої архітектури і може свідчити про більшу чутливість Gemma до архітектурних змін.

Порівняння стратегій переднавчання. MNTP (Masked Next Token Prediction) демонструє незначну, але стабільну перевагу над MLM (Masked Language Modeling) в усіх експериментальних умовах для архітектури Llama. Різниця становить приблизно 0.003–0.004 п.п., що може свідчити про кращу сумісність MNTP з авторегресивною природою базової архітектури. Для архітектури Gemma ця закономірність є менш виразною, що може відображати кращу адаптованість даної моделі до різних стратегій навчання.

4.2.3 Аналіз внутрішніх метрик під час спеціалізованого навчання

Паралельно з оцінкою якості обробки задач маркування послідовностей здійснювався моніторинг метрик симетричності та направленості під час fine-tuning на цільовій задачі. Результати для об'єднаних матриць уваги представлено на рисунках 4.5 та 4.6; для LoRA-адаптерів на рисунках 4.7, 4.8.

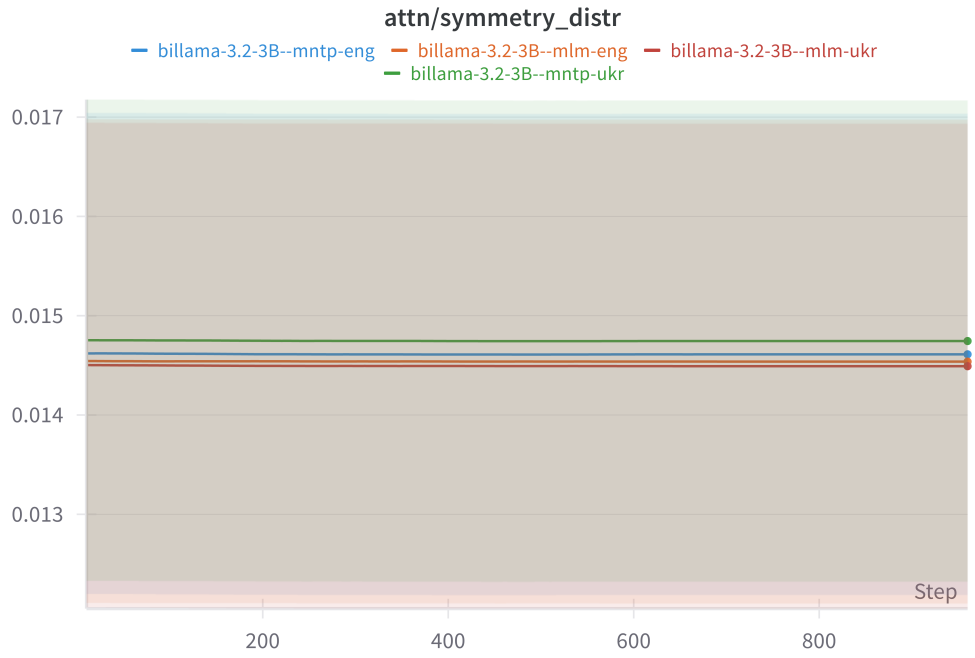


Рисунок 4.5. Еволюція $\mathcal{S}$ на об'єднаних матрицях $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_{\text{LoRA}}$ під час fine-tuning.

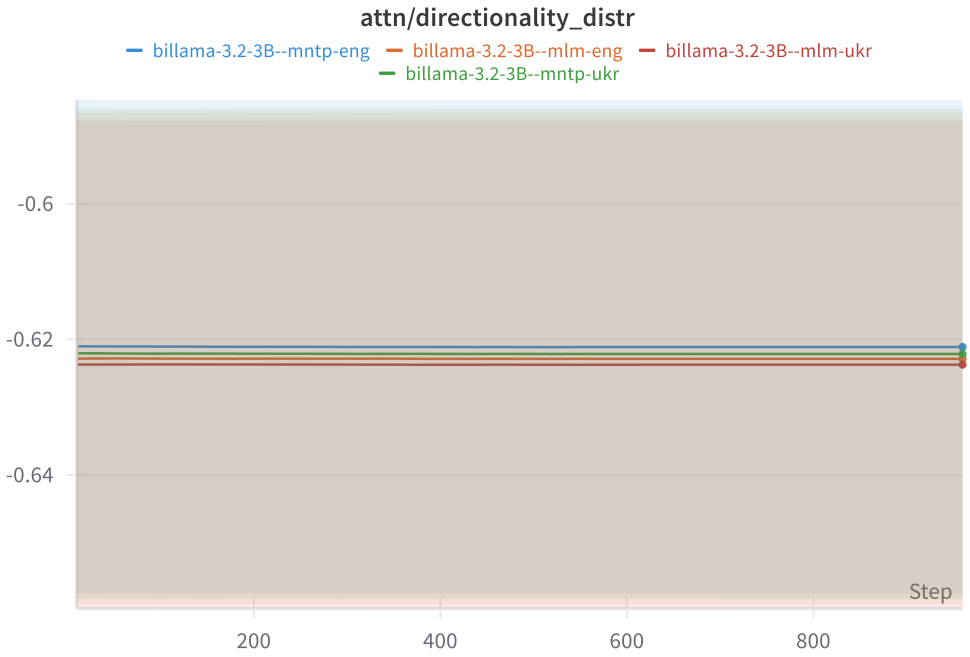


Рисунок 4.6. Еволюція $\mathcal{D}$ на об'єднаних матрицях $\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_{\text{LoRA}}$ під час fine-tuning.

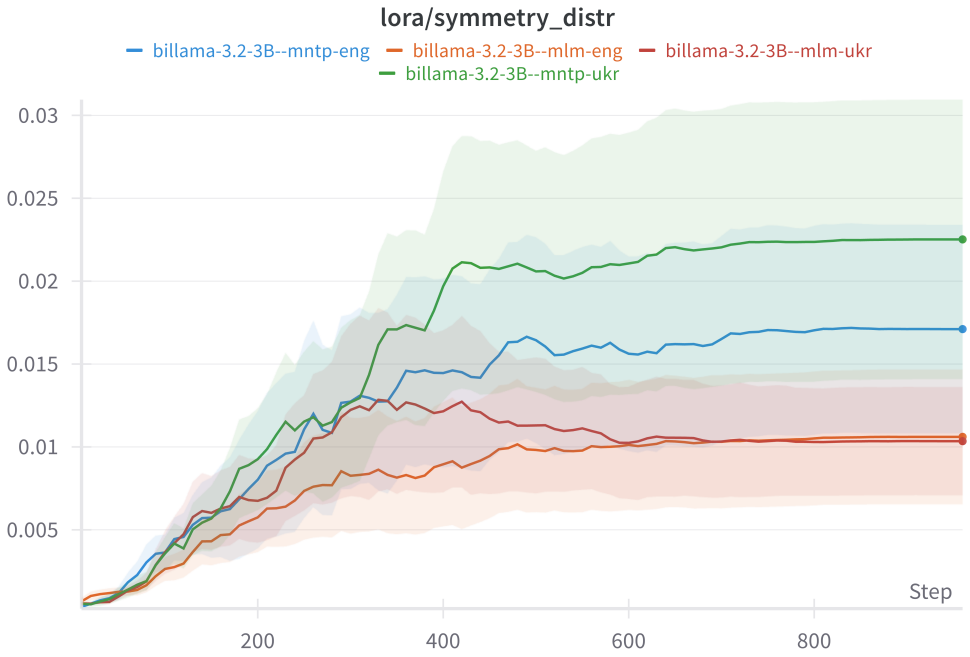


Рисунок 4.7. Еволюція $\mathcal{S}$ на LoRA-адаптерах $\Delta\mathbf{W}_{\text{LoRA}}$ під час fine-tuning.

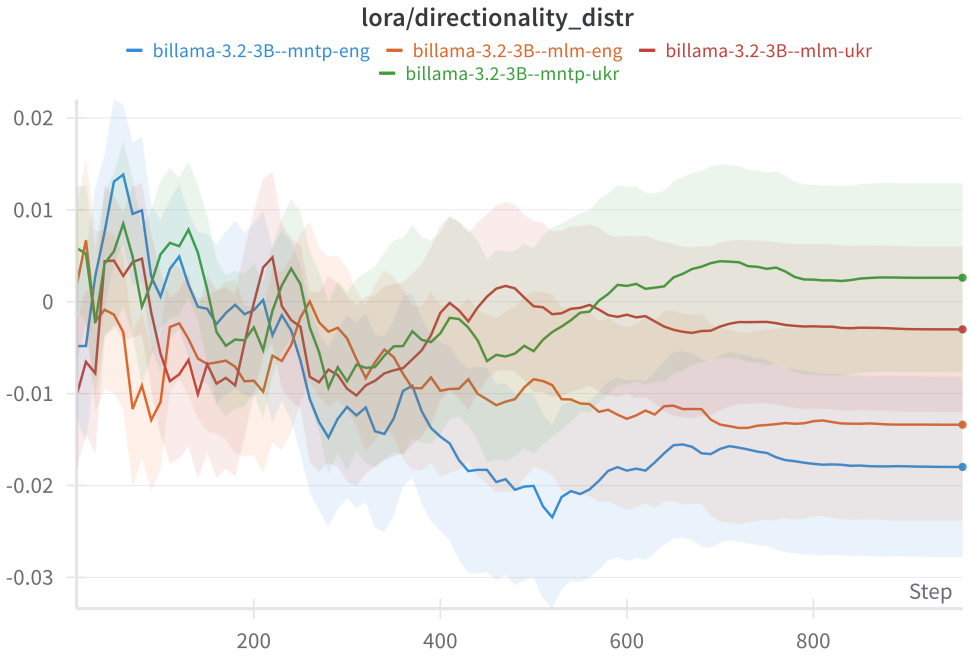


Рисунок 4.8. Еволюція $\mathcal{D}$ на LoRA-адаптерах $\Delta\mathbf{W}_{\text{LoRA}}$ під час fine-tuning.

Окремо було проаналізовано еволюцію метрики направленості $\mathcal{D}$ під час етапу fine-tuning для моделей, які не проходили додаткового переднавчання на задачах двонаправленого моделювання. Зокрема, здійснено порівняльний аналіз змін цього показника як для об'єднаних матриць уваги $\mathbf{W} = \mathbf{W}0 + \Delta\mathbf{W}_{\text{LoRA}}$, так і для самих LoRA-адаптерів $\Delta\mathbf{W}_{\text{LoRA}}$. На рисунках 4.9 та 4.10 наведено відповідні графіки, що відображають динаміку метрики $\mathcal{D}$ протягом fine-tuning у порівнянні між однонаправленою та двонаправленою версіями моделей без додаткового навчання.

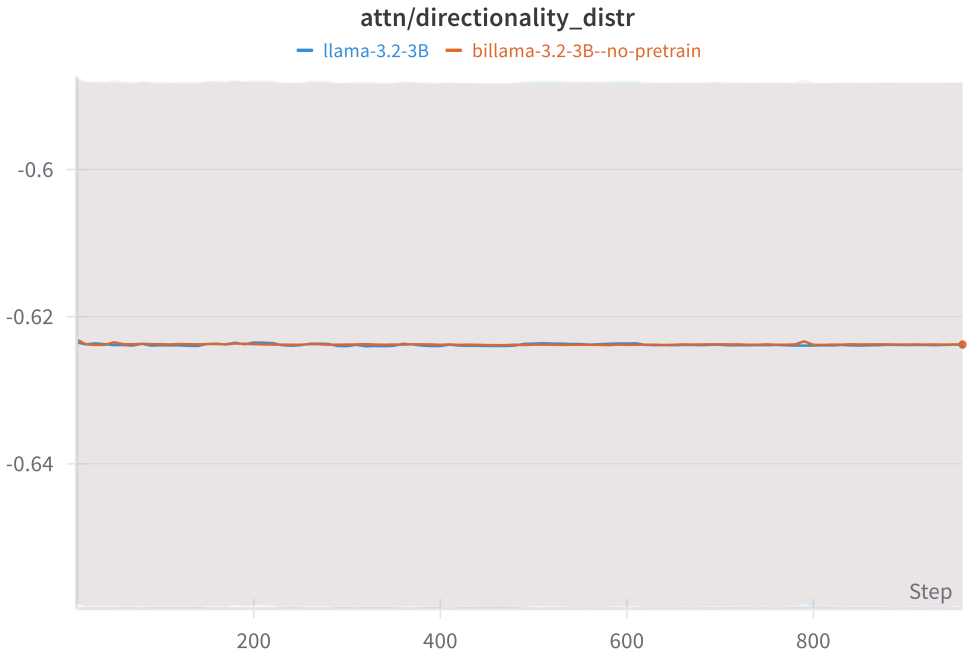


Рисунок 4.9. Еволюція $\mathcal{D}$ на об'єднаних матрицях для однонаправленої та двонаправленої версій без переднавчання.

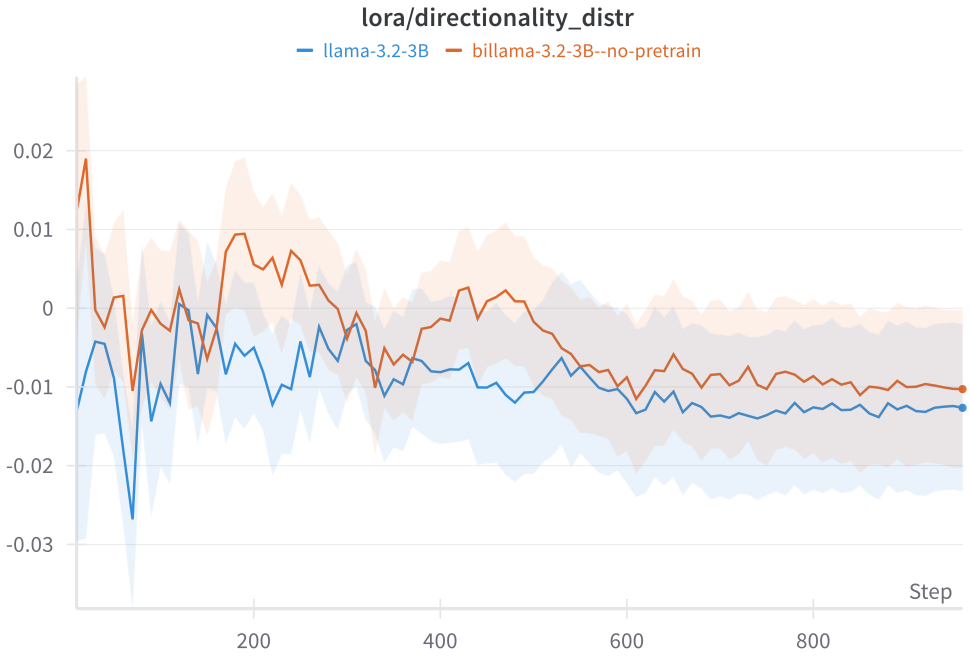


Рисунок 4.10. Еволюція $\mathcal{D}$ на LoRA-адаптерах для однонаправленої та двонаправленої версій без переднавчання.

Однак результати виявилися ще більш проблематичними порівняно з етапом переднавчання.

Вплив квантизації на метрики. Використання 4-bit квантизації NF4 в поєднанні з LoRA адаптерами призвело до численних артефактів у обчисленні метрик на окремих шарах мережі. Значна частина результатів була визнана некоректною та виключена з аналізу.

Відсутність кореляції з цільовою метрикою. Навіть у випадках, коли метрики симетричності та направленості демонстрували виражені зміни під час навчання, жодної значущої кореляції з F1-score на валідаційному наборі виявлено не було. Це ставить під сумнів практичну цінність цих метрик для моніторингу прогресу адаптації.

Подібність паттернів незалежно від архітектури. Порівняння еволюції метрик між однонаправленою та «порожньою» двонаправленою версіями не виявило суттєвих відмінностей, що додатково підкреслює обмеженість існуючого математичного апарату для характеризації двонаправленості.

4.3 Висновки та подальші дослідження

4.3.1 Узагальнені висновки експериментального дослідження

Підтвердження основної гіпотези. Експериментальні результати переконливо підтверджують основну гіпотезу дослідження: зняття каузального маскування в попередньо навчених авторегресивних моделях призводить до суттєвого покращення їхньої ефективності на задачах розуміння тексту. Навіть мінімальна архітектурна модифікація без додаткового навчання забезпечує значний приріст продуктивності для деяких з розглянутих моделей.

Важливість мовної специфічності. Результати демонструють критичну важливість узгодженості між мовою переднавчання та цільовими дани-

ми. Використання неспорідненого мовного корпусу може не лише не принести користі, але й погіршити результати, що має важливі практичні наслідки для багатомовних застосувань.

Обмеженість існуючих метрик двонаправленості. Дослідження виявило серйозні обмеження у застосовності існуючих математичних метрик для оцінки та моніторингу двонаправленості в контексті адаптації LLM.

1. Нечутливість до процесу адаптації: метрики залишаються стабільними навіть при очевидних змінах у поведінці моделі.
2. Вразливість до технічних артефактів: квантизація та низькорангова адаптація вносять значні спотворення в обчислення.
3. Відсутність кореляції з практичною ефективністю: внутрішні метрики не прогнозують покращення downstream performance.

4.3.2 Перспективи подальших досліджень

Отримані результати окреслюють кілька важливих напрямків для майбутніх досліджень.

1. Дослідження ефективності двонаправленої адаптації декодерів до задач побудови семантичних ембедингів.
2. Розробка альтернативних метрик. Необхідність створення нових математичних інструментів для характеризації двонаправленості, які враховували б специфіку адаптаційних процесів та були б стійкими до технічних артефактів сучасних методів оптимізації.
3. Аналіз інформаційного потоку. Перспективним підходом може бути дослідження патернів передачі інформації між токенами під час виконання задач, що критично залежать від двонаправленого контексту.
4. Вивчення гібридних архітектур. Подальше дослідження структур уваги, що виникають в процесі адаптації та можуть поєдну-

вати властивості енкодерів та декодерів у нових конфігураціях.

4.4 Висновки до четвертого розділу

У розділі було проведено детальний аналіз змін внутрішніх метрик та результатів на практичних задачах після двонаправленої адаптації авторегресивних моделей. Встановлено, що зняття каузального маскування забезпечує помітне покращення якості на задачах розуміння тексту, а найбільший ефект спостерігається при відповідності мови переднавчання цільовим даним. Разом з тим, розглянуті метрики симетричності та напрямленості не продемонстрували достатньої чутливості до процесу адаптації і не корелювали з практичною ефективністю моделей.

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі. Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

5.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

1) функціонування на персональних комп'ютерах із стандартним набором компонентів;
2) зручність та зрозумілість для користувача;
3) швидкість обробки даних та доступ до інформації в реальному часі;
4) можливість зручного масштабування та обслуговування;
5) мінімальні витрати на впровадження програмного продукту.

5.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

1) F_1 – вибір мови програмування;
2) F_2 – вибір операційної системи;
3) F_3 – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

1. Функція F_1:
    1) Python;
    2) R.
2. Функція F_2:
    1) MacOS;
    2) Linux.
3. Функція F_3:
    1) VSCode;
    2) PyCharm.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

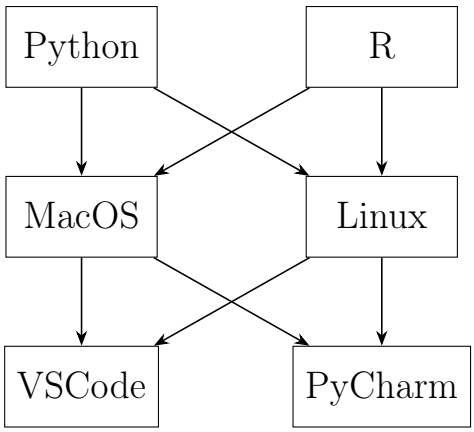


Рисунок 5.1. Морфологічна карта.

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 5.1.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

1. Функція F_1. Перевагу даємо мові програмування Python. Для спрощення роботи по написанню коду варіант В має бути відкинутий.

Таблиця 5.1. Позитивно-негативна матриця.

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Багата екосистема ML бібліотек, легка інтеграція з vast.ai.	Повільна швидкодія, проблеми з пам'яттю.
	B	Потужна статистика, хороша візуалізація.	Обмежені DL бібліотеки, складна робота з GPU.
F_2	A	Стабільність, інтеграція з Apple Silicon.	Обмежена CUDA підтримка, вища вартість.
	B	Нативна CUDA підтримка, краща сумісність з vast.ai.	Складніше налаштування.
F_3	A	Універсальність, багато плагінів, підтримка Jupyter.	Ресурсоємний, повільний запуск.
	B	Потужні Python інструменти, вбудований debugger.	Важкий, платна повна версія.

2. Функція F_2. Програма допускає обрання обох варіантів. Можливо використати варіанти A чи B.
3. Функція F_3. Реалізація першого варіанту є сприйнятливою для програми. Це варіант A.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1A - F_2A - F_3A$$
$$F_1A - F_2B - F_3A$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

1) X_1 – швидкодія мови програмування;
2) X_2 – об'єм пам'яті для обчислень та збереження даних;
3) X_3 – час навчання даних;
4) X_4 – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 5.2.

Таблиця 5.2. Основні параметри програмного продукту.

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X_1	оп/мс	60	80	110
Об'єм пам'яті	X_2	Мб	60	50	30
Час попередньої обробки даних	X_3	мс	80	70	60
Потенційний об'єм програмного коду	X_4	кількість рядків коду	35	25	20

За даними таблиці 5.2 будуються графічні характеристики параметрів – (рисунки 5.2 – 5.5).

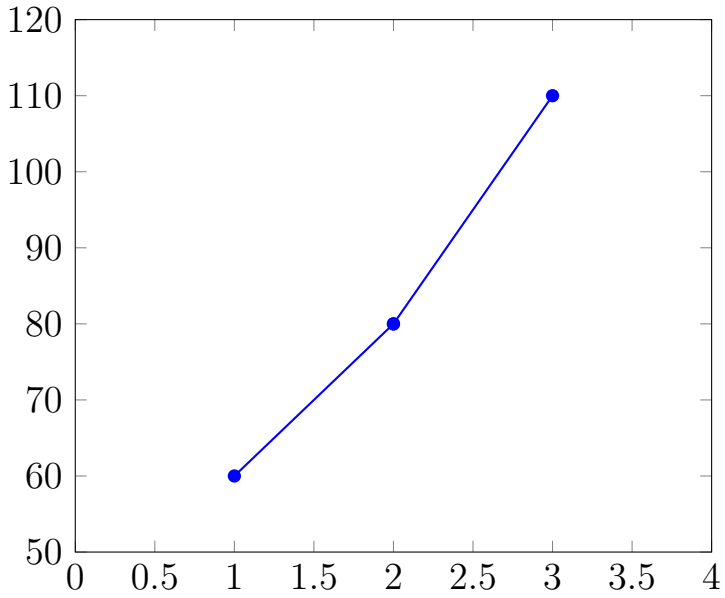


Рисунок 5.2. X_1, швидкодія мови програмування.

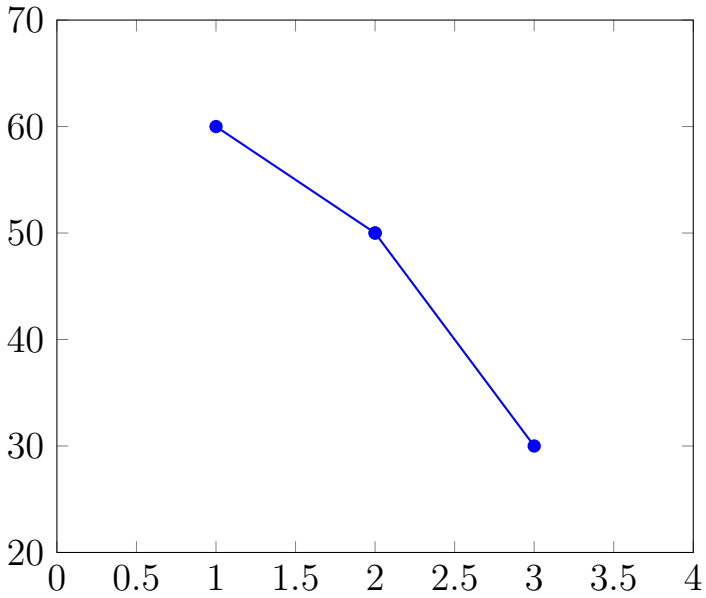


Рисунок 5.3. X_2, об'єм пам'яті.

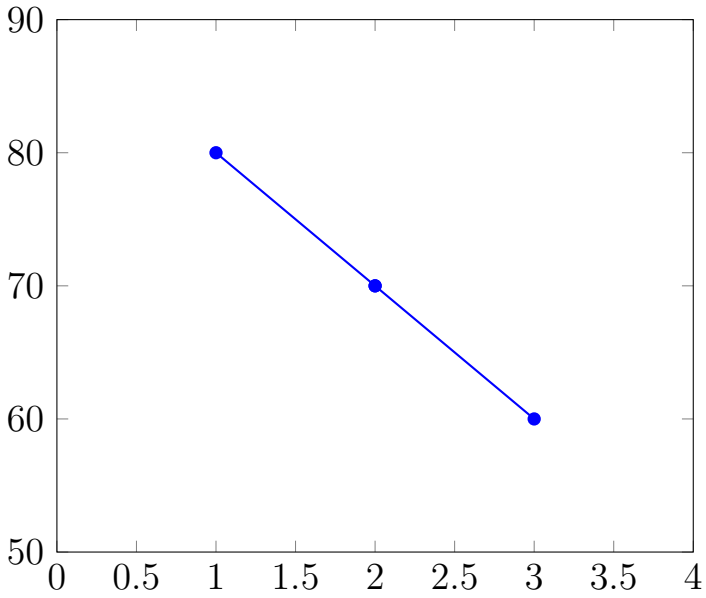


Рисунок 5.4. X_3, час попередньої обробки даних.

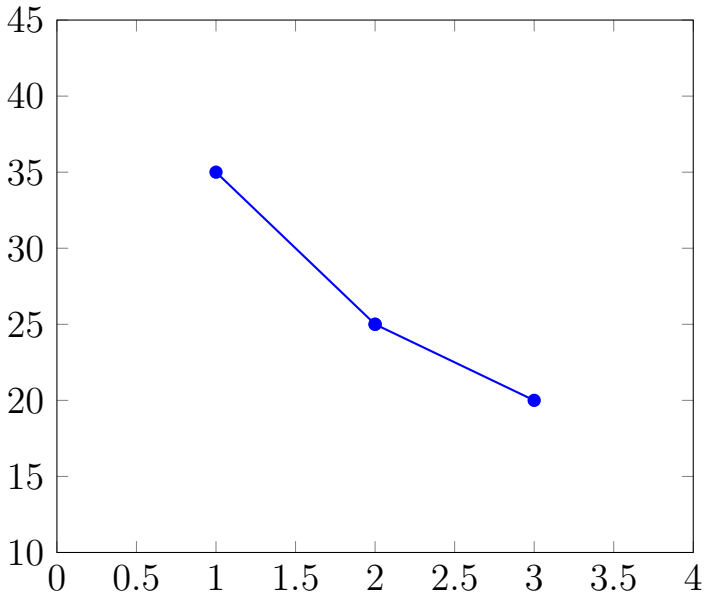


Рисунок 5.5. X_4, потенційний об'єм програмного коду.

5.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;
3) визначення оцінки попарного пріоритету параметрів;
4) обробку результатів та визначення коефіцієнту значимості.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри.

1. Сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = N\frac{n(n+1)}{2} = 70$$

де N – число експертів;
n – кількість параметрів.

2. Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5.$$

3. Відхилення суми рангів кожного параметра від середньої суми

рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0.

4. Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0,754 > W_k = 0,67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Результати експертного ранжування наведені у таблиці 5.3.

Таблиця 5.3. Результати ранжування параметрів

Параметр	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δi	Δi^2
			1	2	3	4	5	6	7			
X_1	Швидкодія мови програмування	Оп/мс	1	2	2	1	1	1	2	10	-7,5	56,25
X_2	Об'єм пам'яті	Mb	3	4	3	3	4	3	4	24	6,5	42,25
X_3	Час попередньої обробки даних	мс	2	1	1	2	2	2	1	11	-6,5	42,25
X_4	Потенційний об'єм програмного коду	Кількість рядків коду	4	3	4	4	3	4	3	25	7,5	56,25
Разом			10	10	10	10	10	10	10	70	0	197

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.4.

Таблиця 5.4. Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X_1 і X_2	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0,5
X_1 і X_3	$<$	$>$	$>$	$<$	$<$	$<$	$>$	$<$	0,5
X_1 і X_4	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0,5
X_2 і X_3	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	1,5
X_2 і X_4	$<$	$>$	$<$	$<$	$>$	$<$	$<$	$<$	0,5
X_3 і X_4	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	0,5

Числове значення, що визначає ступінь переваги i–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 & \text{при } X_i > X_j \\ 1.0 & \text{при } X_i = X_j \\ 0.5 & \text{при } X_i < X_j \end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю $A = ||a_{ij}||$.

Для кожного параметра зробимо розрахунок вагомості Кві за наступними формулами:

$$\text{К}_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$b'_i = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5. Розрахунок вагомості параметрів

Параметри X_i	Параметри X_j				Перша іт.		Друга іт.		Третя іт.	
	X_1	X_2	X_3	X_4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X_1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X_2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X_3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X_4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього :					16	1	59	1	213	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X$_2$ (об'єм пам'яті), X_3 (час попередньої обробки даних) та $X4$ (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.6):

$$K_K(j) = \sum_{i=1}^{n} \text{K}_{\text{ві},j} B_{i,j}$$

де n — кількість параметрів;

$K_{\text{ві}}$ — коефіцієнт вагомості і–го параметра;

B_i — оцінка i–го параметра в балах.

За даними з таблиці 5.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \cdots + K_{\text{ТУ}}[F_{zk}]$$

Таблиця 5.6. Розрахунок показників рівня якості варіантів реалізації основних функцій ПП.

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	A	X_1	100	25	$0,16$	4
F_3	A	X_2	87	29	$0,28$	$8,12$
F_4	B	X_3	27	19	$0,20$	$3,8$
	A	X_4	25	23	$0,36$	$8,28$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4 + 8.12 + 8.28 = 20.4;$$
$$K_{K2} = 4 + 3.8 + 8.28 = 16.08.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1) розробка проекту програмного продукту;

2) розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_{\text{П}} \cdot K_{\text{СК}} \cdot K_{\text{М}} \cdot K_{\text{СТ}} \cdot K_{\text{СТ.М}}$$

де T_P – трудомісткість розробки ПП;

$K_{\text{П}}$ – поправочний коефіцієнт;

$K_{\text{П}}$ – коефіцієнт на складність вхідної інформації;

$K_{\text{СК}}$ – коефіцієнт рівня мови програмування;

$K_{\text{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{\text{СТ.М}}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_{\text{Р}} = 37$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\text{П}} = 1.8$ Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{\text{СК}} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{\text{СТ}} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість.

1. $T_I = (59,94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852$ людино-днів.
2. $T_{II} = (59,94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868,88$ людино-днів.

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 17000 грн., один аналітик в області даних з окладом 19000. Визначимо середню зарплату за

годину за формулою:

$$C_{\text{Ч}} = \frac{M}{T_m \cdot t}\text{грн.},$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$C_{\text{Ч}} = \frac{17000 + 17000 + 19000}{3 \cdot 21 \cdot 8}\text{грн.},$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{ЗП}} = C_{\text{Ч}} \cdot T_i \cdot K_{\text{Д}}$$

де $C_{\text{Ч}}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$K_{\text{Д}}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $C_{\text{ЗП}} = 105.16 \cdot 852 \cdot 1.2 = 107515,58$ грн.

II. $C_{\text{ЗП}} = 105.16 \cdot 868.88 \cdot 1.2 = 109645,7$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 107515,58 \cdot 0.22 = 23653,4$ грн.

II. $C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 109645,7 \cdot 0.22 = 24122,06$ грн.

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 17000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{\text{Г}} = 12 \cdot M \cdot K_{\text{З}} = 12 \cdot 17000 \cdot 0,2 = 40800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_{\text{Г}} \cdot (1 + K_{\text{З}}) = 40800 \cdot (1 + 0.2) = 48960 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 48960 \cdot 0,22 = 10771,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$C_{\text{А}} = K_{\text{ТМ}} \cdot K_{\text{А}} \cdot \text{Ц}_{\text{ПР}} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн.},$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;
$K_{\text{А}}$ – річна норма амортизації;
$\text{Ц}_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_{\text{Р}} = K_{\text{Т}}\text{М} \cdot \text{Ц}_{\text{П}}\text{Р} \cdot K_{\text{Р}} = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн.},$$

де $K_{\text{Р}}$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (\text{Д}_{\text{К}} - \text{Д}_{\text{В}} - \text{Д}_{\text{С}} - \text{Д}_{\text{Р}}) \cdot t_{\text{З}} \cdot K_{\text{В}} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.35 = 627,2 \text{ години},$$

де $\text{Д}_{\text{К}}$ – календарна кількість днів у році;
$\text{Д}_{\text{В}}, \text{Д}_{\text{С}}$ – відповідно кількість вихідних та святкових днів;
$\text{Д}_{\text{Р}}$ – кількість днів планових ремонтів устаткування;
t – кількість робочих годин в день;
$K_{\text{В}}$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_{\text{З}} \cdot \text{Ц}_{\text{ЕН}} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 9,43 = 354,9 \text{ грн.},$$

де N_C – середньо-споживча потужність приладу;
$K_{\text{З}}$ – коефіцієнтом зайнятості приладу;
$\text{Ц}_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = \text{Ц}_{\text{ПР}} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}$$
$$C_{\text{ЕКС}} = 48960 + 10771,2 + 3500 + 1120 + 354,9 + 6700 = 71406,1 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М–Г}} = C_{\text{ЕКС}}/T_{\text{ЕФ}} = 71406,1/627,2 = 113,85 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{\text{М–Г}} \cdot T$$

I. $C_M = 113,85 \cdot 852 = 97000,2$ грн.

II. $C_M = 113,85 \cdot 868,88 = 98921,99$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н}} = C_{\text{ЗП}} \cdot 0,67$$

I. $C_{\text{Н}} = 107515,58 \cdot 0,67 = 72035,45$ грн.

II. $C_{\text{Н}} = 109645,70 \cdot 0,67 = 73462,6$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}$$

I. $C_{\text{ПП}} = 107515,58 + 23653,4 + 97000,2 + 72035,45 =$
$= 300204,63$ грн.

II. $C_{\text{ПП}} = 109645,7 + 24122,06 + 98921,99 + 73462,6 =$
$= 306152,35$ грн.

5.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{Kj}/C_{\text{ф}j},$$
$$K_{\text{ТЕР1}} = 20,4/300204,63 = 6,795 \cdot 10^{-5},$$
$$K_{\text{ТЕР2}} = 16,08/306152,35 = 5,252 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР1}} = 6,795 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 6,795 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

1) Вибір мови програмування – Python;
2) Вибір операційної системи – MacOS;
3) Вибір середовища розробки – VSCode.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

5.8 Висновки до п'ятого розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У комплексному дослідженні, присвяченому адаптації декодер-онлі мовних моделей до завдань, що вимагають двостороннього аналізу контексту, вирішено низку теоретичних і практичних проблем, які раніше стримували використання авторегресивних трансформерів у дискримінативних задачах, що вимагають двостороннього контексту. Насамперед було проаналізовано фундаментальне обмеження каузальної маски: будучи необхідною складовою для авторегресивної генерації, вона водночас позбавляє модель здатності враховувати інформацію з майбутніх позицій, що є критичним для задач маркування послідовностей. У роботі продемонстровано, що повне усунення цієї маски у поєднанні з низькоранговою адаптацією задачею Masked Next-Token Prediction забезпечує стійкий приріст точності.

Запропоновано формально-математичний апарат для кількісного вимірювання ступеня «симетризації» уваги після зняття каузального обмеження. Введені показники симетрії $\mathcal{S}$ та напрямленості $\mathcal{D}$ що хоч і не дали очікуваних результатів, проте породили ряд додаткових запитань стосовно природи двонаправленої адаптації декодерних моделей. Емпірична перевірка на українському корпусі UNLP-2025 засвідчила приріст макро-F1 у порівнянні з базовою каузальною моделлю, що є значущим з огляду на високу насиченість цього датасетів складними лінгвістичними явищами.

З методологічного погляду робота демонструє, що навіть без істотного розширення навчального корпусу можна досягти релевантних покращень у задачах маркування послідовностей, якщо правильно скоригувати архітектурні обмеження та підібрати цільове донавчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is all you need. Advances in Neural Information Processing Systems. 2017. Vol. 30.

[2] Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT). Minneapolis. 2019. P. 4171–4186.

[3] Kaplan J., McCandlish S., Henighan T., Brown T. B., Chess B., Child R., Gray S., Radford A., Wu J., Amodei D. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361. 2020.

[4] Wei J., Tay Y., Bommasani R., Raffel C., Zoph B., Borgeaud S., Yogatama D., Bosma M., Zhou D., Metzler D., et al. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682. 2022.

[5] Lv A., Zhang K., Xie S., Tu Q., Chen Y., Wen J.-R., Yan R. An analysis and mitigation of the reversal curse. arXiv preprint arXiv:2311.07468. 2023.

[6] Nie S., Zhu F., You Z., Zhang X., Ou J., Hu J., Zhou J., Lin Y., Wen J.-R., Li C. Large language diffusion models. arXiv preprint arXiv:2502.09992. 2025.

[7] Springer J. M., Kotha S., Fried D., Neubig G., Raghunathan A. Repetition improves language model embeddings. arXiv preprint arXiv:2402.15449. 2024.

[8] Fu Y., Cheng Z., Jiang Z., Wang Z., Yin Y., Li Z., Gu Q. Token prepending: A training-free approach for eliciting better sentence embeddings from LLMs. arXiv preprint arXiv:2412.11556. 2024.

[9] Li Z., Li X., Liu Y., Xie H., Li J., Wang F., Li Q., Zhong X. Label-supervised LLaMA fine-tuning. arXiv preprint arXiv:2310.01208. 2023.

[10] Li X., Li J. BELLM: Backward dependency-enhanced large language model for sentence embeddings. arXiv preprint arXiv:2311.05296. 2023.

[11] Dukić D., Šnajder J. Looking right is sometimes right: Investigating the capabilities of decoder-only LLMs for sequence labeling. arXiv preprint arXiv:2401.14556. 2024.

[12] Suganthan P., Moiseev F., Yan L., Wu J., Ni J., Han J., Zitouni I., Alfonseca E., Wang X., Dong Z. Adapting decoder-based language models for diverse encoder downstream tasks. arXiv preprint arXiv:2503.02656. 2025.

[13] BehnamGhader P., Adlakha V., Mosbach M., Bahdanau D., Chapados N., Reddy S. LLM2Vec: Large language models are secretly powerful text encoders. arXiv preprint arXiv:2404.05961. 2024.

[14] Joshi R., Singla K., Kamath A., Kalani R., Paul R., Vaidya U., Chauhan S. S., Wartikar N., Long E. Adapting multilingual LLMs to low-resource languages using continued pre-training and synthetic corpus. arXiv preprint arXiv:2410.14815. 2024.

[15] Gurgurov D., Hartmann M., Ostermann S. Adapting multilingual LLMs to low-resource languages with knowledge graphs via adapters. arXiv preprint arXiv:2407.01406. 2024.

[16] Costa-Jussà M. R., Cross J., Çelebi O., Elbayad M., Heafield K., Heffernan K., Kalbassi E., Lam J., Licht D., Maillard J., et al. No language left behind: Scaling human-centered machine translation. arXiv preprint arXiv:2207.04672. 2022.

[17] Hu J., Ruder S., Siddhant A., Neubig G., Firat O., Johnson M. XTREME: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. Proceedings of the 37th International Conference on Machine Learning (ICML). 2020. P. 4411–4421.

[18] Su J., Ahmed M., Lu Y., Pan S., Bo W., Liu Y. RoFormer: Enhanced Transformer with rotary position embedding. Neurocomputing. 2024. Vol. 568. Article 127063.

[19] Shazeer N. GLU variants improve Transformer. arXiv preprint arXiv:2002.05202. 2020.

[20] Zhang B., Sennrich R. Root mean square layer normalization. Advances in Neural Information Processing Systems. 2019. Vol. 32.

[21] Hinton G., Vinyals O., Dean J. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531. 2015.

[22] Nagel M., Fournarakis M., Amjad R. A., Bondarenko Y., Van Baalen M., Blankevoort T. A white paper on neural network quantization. arXiv preprint arXiv:2106.08295. 2021.

[23] Guo Y., Yao A., Chen Y. Dynamic network surgery for efficient DNNs. Advances in Neural Information Processing Systems. 2016. Vol. 29.

[24] Krishnamoorthi R. Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342. 2018.

[25] Jacob B., Kligys S., Chen B., Zhu M., Tang M., Howard A., Adam H., Kalenichenko D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City. 2018. P. 2704–2713.

[26] Hu E. J., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang S., Wang L., Chen W., et al. LoRA: Low-rank adaptation of large language models. International Conference on Learning Representations (ICLR). 2022. Vol. 1. 3 p.

[27] Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient fine-tuning of quantized LLMs. Advances in Neural Information Processing Systems. 2023. Vol. 36. Pp. 10088–10115.

[28] Saponati M., Sager P., Aceituno P. V., Stadelmann T., Grewe B. The underlying structures of self-attention: Symmetry, directionality, and emergent dynamics in Transformer training. arXiv preprint arXiv:2502.10927. 2025.

[29] UNLP 2025 shared task on detecting social media manipulation. GitHub repository. 2025. Available at: https://github.com/unlp-workshop/unlp-2025-shared-task (last accessed 09.06.2025).

[30] Wikimedia – Wikipedia dumps. Hugging Face datasets. 2024. Available at: https://huggingface.co/datasets/wikimedia/wikipedia (last accessed 09.06.2025).

[31] AI @ Meta. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783. 2024.

[32] Gemma Team, Kamath A., Ferret J., Pathak S., Vieillard N., Merhej R., Perrin S., Matejovicova T., Ramé A., Rivière M., et al. Gemma 3 technical report. arXiv preprint arXiv:2503.19786. 2025.

А ЛІСТИНГ ПРОГРАМНОГО МОДУЛЮ

```
================================================
FILE: config/config.yaml
================================================
hydra:
  run:
    dir: "hydra_logs"
  output_subdir: ${now:%Y-%m-%d_%H-%M-%S}

defaults:
  - _self_
  - data: data
  - model: model
  - train: train


seed: 42
experiment_name: gemma-3-4b--${train.task}--no-quant--eng


wandb:
  project: bidirectional-decoder
  entity: havlytskyi-thesis
  name: ${experiment_name}
#  log_model: all



================================================
FILE: config/data/data.yaml
================================================
dataset:
  path: wikimedia/wikipedia
  name: 20231101.en
  split: train

sample:
```

```
  num_samples: 70000
  seed: ${seed}

tokenize:
  max_length: 512
  truncation: True


================================================
FILE: config/model/model.yaml
================================================
model_family: gemma3


model:
  pretrained_model_name_or_path: google/gemma-3-4b-pt
  torch_dtype: bfloat16


# quantization:
#   load_in_4bit: True
#   bnb_4bit_quant_type: nf4
#   bnb_4bit_use_double_quant: False
#   bnb_4bit_compute_dtype: float16


lora:
  r: 64  # the dimension of the low-rank matrices
  lora_alpha: 128 # scaling factor for LoRA activations vs pre-trained weight
      activations
  lora_dropout: 0.05
  bias: none
  inference_mode: False
  task_type: CAUSAL_LM
  target_modules: ['o_proj', 'v_proj', "q_proj", "k_proj", "gate_proj", "
     down_proj", "up_proj"]


================================================
FILE: config/train/train.yaml
================================================
task: mntp # mlm | mntp

# We should use some existing token
mask_token: "_"
```

```
training_args:
    learning_rate: 2e-4
    weight_decay: 0.01
    lr_scheduler_type: cosine
    warmup_ratio: 0.0
    max_steps: 2000
    per_device_train_batch_size: 8
    gradient_accumulation_steps: 4
    do_train: True
    do_eval: False
    bf16: True
    report_to: wandb
    optim: adamw_8bit
    save_strategy: steps
    logging_steps: 20
    save_steps: 200


================================================
FILE: scripts/train.py
================================================
import os
import hydra
from omegaconf import DictConfig, OmegaConf
import wandb
from src.utils.other import set_seeds

import torch
from transformers import (
    AutoTokenizer,
    DataCollatorForLanguageModeling,
    Trainer,
    TrainingArguments,
    BitsAndBytesConfig
)
from peft import get_peft_model, LoraConfig, TaskType
from datasets import load_dataset
from src.models import MODELS_MAPPING
from src.callbacks.partial_grad_norm import PartialGradNormCallback
from src.callbacks.directionality import AttentionGeometryCallback

@hydra.main(version_base=None, config_path="../config", config_name="config")
def main(cfg: DictConfig):
```

```
set_seeds(cfg.seed)

model_class = MODELS_MAPPING.get(
    cfg.model.model_family, {}
    ).get(cfg.train.task)
assert model_class, f"Model family {cfg.model.model_family} and task {cfg
   .model.task} not supported."

tokenizer = AutoTokenizer.from_pretrained(
    cfg.model.model.pretrained_model_name_or_path
)
if not tokenizer.pad_token:
    tokenizer.pad_token = tokenizer.eos_token

if hasattr(cfg.model, "quantization"):
    quant_cfg = OmegaConf.to_container(cfg.model.quantization, resolve=
       True)
    quant_cfg["bnb_4bit_compute_dtype"] = getattr(torch, quant_cfg["
       bnb_4bit_compute_dtype"])
    quant_config = BitsAndBytesConfig(
        **quant_cfg
    )
else:
    quant_config = None


model_cfg = OmegaConf.to_container(cfg.model.model, resolve=True)
model_cfg['torch_dtype'] = getattr(torch, model_cfg['torch_dtype'])
model = model_class.from_pretrained(
    **model_cfg,
    quantization_config=quant_config,
)

if quant_config:
    model = prepare_model_for_kbit_training(model)

if hasattr(cfg.model, "lora"):
    lora_cfg = OmegaConf.to_container(cfg.model.lora, resolve=True)
    lora_config = LoraConfig(**lora_cfg)
    model = get_peft_model(model, lora_config)

    model.print_trainable_parameters()

train_args = TrainingArguments(
    output_dir=f'./checkpoints/{cfg.wandb.name}',
```

```
        logging_dir=f'./logs/{cfg.wandb.name}',
        **cfg.train.training_args
    )

    # DATASET
    ds = load_dataset(**cfg.data.dataset)
    sampled_ds = ds.shuffle(
        seed=cfg.data.sample.seed
        ).select(range(cfg.data.sample.num_samples)
        ).select_columns(["text"])

    tokenized_dataset = sampled_ds.map(
        lambda examples: tokenizer(examples["text"], **cfg.data.tokenize),
        batched=True, num_proc=10
        ).remove_columns(["text"])


    tokenizer.mask_token = cfg.train.mask_token
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer, mlm=True, mlm_probability=0.2
    )

    wandb_run = wandb.init(
        **cfg.wandb,
        config={'hydra': OmegaConf.to_container(cfg, resolve=True,
            throw_on_missing=True)}
    )
    wandb.define_metric("*", summary="none")

    trainer = Trainer(
        model=model,
        args=train_args,
        data_collator=data_collator,
        train_dataset=tokenized_dataset,
        # callbacks=callbacks
    )

    trainer.train()

    wandb_run.finish()

if __name__ == "__main__":
    main()
```

```
================================================
FILE: scripts/train.sh
================================================
python scripts/train.py
python scripts/train.py train.task=mlm
python scripts/train.py data.dataset.name="20231101.uk" experiment_name="
   gemma-3-4b--mntp--no-quant--ukr"
python scripts/train.py train.task=mlm data.dataset.name="20231101.uk"
   experiment_name="gemma-3-4b--mlm--no-quant--ukr"


================================================
FILE: scripts/wandb_to_hf.sh
================================================
#!/usr/bin/env bash

# Check if required arguments are provided
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <RUN_ID> <VERSION>"
    echo "Example: $0 hrui1936 v9"
    exit 1
fi

#           CONFIG


# Weights & Biases config
ENTITY="havlytskyi-thesis"
PROJECT="bidirectional-decoder"

RUN_ID="$1"
VERSION="$2"
ARTIFACT="model-${RUN_ID}:${VERSION}"         # replace artifact name:version


BASE_DIR="data/pretrained_models" # replace with your base directory

# Kaggle config for saving old versions to locan main folder
HF_REPO_ID="GPT2Vec"
HF_PATH="."


#           1) DERIVE DESTINATION FOLDER FROM RUN NAME
```

```
RUN_NAME=$(python3 <<EOF
import wandb, re
ENTITY="${ENTITY}"
PROJECT="${PROJECT}"
RUN_ID="${RUN_ID}"
api = wandb.Api()
run = api.run(f"{ENTITY}/{PROJECT}/{RUN_ID}")
name = run.name or run.id
# slugify: replace  n o n alphanumeric  with underscore
slug = re.sub(r"\W+", "_", name)
print(slug)
EOF
)

DEST_DIR="${BASE_DIR}/${RUN_ID}_${VERSION}_${RUN_NAME}"

# #           2) PREP


# echo ">>> Preparing directory structure"
# rm -rf "$DEST_DIR"
# mkdir -p "$DEST_DIR"

#           3) DOWNLOAD ARTIFACT


echo ">>> Downloading artifact into $DEST_DIR"
wandb artifact get "$ENTITY/$PROJECT/$ARTIFACT" --root "$DEST_DIR"


#           5) UPLOAD TO HUGGINGFACE


echo ">>> Uploading to Hugging Face"
huggingface-cli upload $HF_REPO_ID $DEST_DIR "${RUN_NAME}/" --commit-message
   "${RUN_ID}:${VERSION}" --private


echo "RUN_NAME=${RUN_NAME}"


================================================
FILE: src/callbacks/directionality.py
================================================
import numpy as np
```

```
import torch
from typing import Dict, Any, Iterable

from transformers import TrainerCallback
from transformers.integrations import WandbCallback
from src.metrics.directionality import (
    AttentionExtractor,
    SymmetryScore,
    DirectionalityScore,
)


class AttentionGeometryCallback(WandbCallback):
    def __init__(
        self,
        q_path: str,
        k_path: str,
        layers: Iterable[int] | int | None = None,
        is_lora: bool = False,
        merge_lora: bool = False,
        adapter_name: str | None = None,
        is_quantized: bool = False,
        attention_type: str | None = None,
    ):
        super().__init__()
        self.q_path = q_path
        self.k_path = k_path
        self.layers = [layers] if isinstance(layers, int) else layers

        self.extractor_kwargs = dict(
            attention_type=attention_type,
            is_lora=is_lora,
            merge_lora=merge_lora,
            adapter_name=adapter_name,
            is_quantized=is_quantized,
        )


    def on_log(self, args, state, control, logs=None, **kwargs):
        if logs is None:          # HF passes an empty dict sometimes
            return

        # run only on rank-0 to avoid duplicate logs under DDP / DS
        if getattr(args, "local_rank", -1) not in (-1, 0):
            return
```

```
        model = kwargs["model"]

        with torch.no_grad():     # absolute safety: never touch graph
            extractor = AttentionExtractor(
                model,
                self.q_path,
                self.k_path,
                **self.extractor_kwargs,
            )
            sym  = SymmetryScore(extractor)(self.layers)
            dirc = DirectionalityScore(extractor)(self.layers)


        # HF Trainer will push whatever is inside `logs`
        if self.layers is None:
            self.layers = range(len(sym))

        name = "attn"
        if self.extractor_kwargs['merge_lora'] is False:
            name = "lora_delta"

        tmp_logs = {}
        for layer in self.layers:
            tmp_logs[f"{name}/symmetry/layer_{layer}"] = sym[layer]
            tmp_logs[f"{name}/directionality/layer_{layer}"] = dirc[layer]

        tmp_logs[f"{name}/avg_symmetry"] = np.array(sym).mean()
        tmp_logs[f"{name}/avg_directionality"] = np.array(dirc).mean()

        self._wandb.log(tmp_logs, step=state.global_step)


================================================
FILE: src/callbacks/partial_grad_norm.py
================================================
import re
from typing import Iterable, List, Optional, Sequence, Union

import torch
from transformers import TrainerCallback, TrainingArguments, TrainerState,
   TrainerControl


class PartialGradNormCallback(TrainerCallback):
```

```
def __init__(
    self,
    model: torch.nn.Module,
    target_modules: Union[str, Sequence[str], None] = None,
    target_layers: Optional[Iterable[int]] = None,
    norm_type: float = 2.0,
    log_key: str = "partial_grad_norm",
):
    super().__init__()
    self.model = model
    if target_modules is None:
        target_modules = []
    if isinstance(target_modules, str):
        target_modules = [target_modules]

    self._regexes: List[re.Pattern] = [re.compile(p) for p in
       target_modules]
    self._target_layers = (
        set(int(i) for i in target_layers) if target_layers is not None
           else None
    )
    self.norm_type = norm_type
    self.log_key = log_key
    self._cached_norm: Optional[float] = None


def _is_selected(self, param_name: str) -> bool:
    if self._regexes:  # empty list -> keep everything
        if not any(r.search(param_name) for r in self._regexes):
            return False

    if self._target_layers is not None:
        m = re.search(r"\.layer[s]?\.([0-9]+)\.", param_name)
        if m is None or int(m.group(1)) not in self._target_layers:
            return False

    return True

def _compute_grad_norm(self) -> float:
    norms = []
    for name, p in self.model.named_parameters():
        if not p.requires_grad or p.grad is None:
            continue
        if self._is_selected(name):
            norms.append(p.grad.detach().norm(self.norm_type))
```

```
        if len(norms) == 0:
            return 0.0

        stacked = torch.stack(norms)
        return stacked.norm(self.norm_type).item()


    # ----------------------- Trainer hooks ----------------------- #

    def on_optimizer_step(
        self,
        args: TrainingArguments,
        state: TrainerState,
        control: TrainerControl,
        **kwargs,
    ):
        # Grads exist at this point (before optimizer.step / zero_grad)
        if control.should_log:
            self._cached_norm = self._compute_grad_norm()
        else:
            self._cached_norm = None

    def on_log(
        self,
        args: TrainingArguments,
        state: TrainerState,
        control: TrainerControl,
        logs=None,
        **kwargs,
    ):
        if self._cached_norm is not None and logs is not None:
            logs[self.log_key] = self._cached_norm



================================================
FILE: src/callbacks/uter.py
================================================
import torch
from transformers import TrainerCallback, TrainingArguments, TrainerState,
   TrainerControl
from typing import Dict, Any
from src.metrics.uter import UTERScore
```

```
class UTERCallback(TrainerCallback):
    def __init__(self, device: str | None = None):
        self.device = device
        self.metric = UTERScore()

    def on_train_batch_end(
        self,
        args: TrainingArguments,
        state: TrainerState,
        control: TrainerControl,
        **kwargs: Dict[str, Any]
    ):
        inputs = kwargs["inputs"]
        outputs = kwargs["outputs"]
        model = kwargs["model"]
        device  = self.device or next(model.parameters()).device

        if isinstance(outputs, dict) and "attentions" in outputs:
            attentions = outputs["attentions"]
        else:
            # run a quick no-grad forward pass to obtain attentions
            with torch.no_grad():
                re_out = model(**{k: v.to(device) for k, v in inputs.items()
                    },
                               output_attentions=True,
                               return_dict=True)
            attentions = re_out.attentions

        attn_mask = inputs["attention_mask"].to(device)
        self.metric.add_batch(attentions, attn_mask)


    def on_step_end(
        self,
        args: TrainingArguments,
        state: TrainerState,
        control: TrainerControl,
        **kwargs
    ):
        layer_means = self.metric.compute()
        if layer_means:
            logs = {f"uter/L{i}": v for i, v in enumerate(layer_means)}
            logs["uter/mean"] = sum(layer_means) / len(layer_means)
            self.log(logs)
```

```
        self.metric.reset()


================================================
FILE: src/losses/losses.py
================================================
import torch.nn as nn
from transformers.loss.loss_utils import fixed_cross_entropy, ForMaskedLMLoss

# Loss is similar to ForMaskedLMLoss, but we need to shift the labels
def ForMaskedNTPLoss(
    logits,
    labels,
    vocab_size: int,
    num_items_in_batch: int = None,
    ignore_index: int = -100,
    shift_labels=None,
    **kwargs,
):
    # Upcast to float if we need to compute the loss to avoid potential
       precision issues
    logits = logits.float()
    labels = labels.to(logits.device)

    if shift_labels is None:
        labels = labels.to(logits.device)
        # Shift so that tokens < n predict n
        labels = nn.functional.pad(labels, (0, 1), value=ignore_index)
        shift_labels = labels[..., 1:].contiguous()

    # Flatten the tokens
    logits = logits.view(-1, vocab_size)
    shift_labels = shift_labels.view(-1)
    # Enable model parallelism
    shift_labels = shift_labels.to(logits.device)
    loss = fixed_cross_entropy(logits, shift_labels, num_items_in_batch,
       ignore_index, **kwargs)
    return loss


LOSS_MAPPING = {
    "mlm": ForMaskedLMLoss,
    "mntp": ForMaskedNTPLoss
}
```

```
================================================
FILE: src/metrics/directionality.py
================================================
import torch
import functools

import bitsandbytes as bnb
from bitsandbytes.functional import dequantize_4bit

from abc import ABC, abstractmethod
from typing import Literal, Dict, Any, Optional, List, Iterable



def rgetattr(obj, attr, *args):
    def _getattr(obj, attr):
        return getattr(obj, attr, *args)
    return functools.reduce(_getattr, [obj] + attr.split('.'))


class AttentionExtractor:
    def __init__(
        self,
        model,
        q_path: str,
        k_path: str,
        attention_type: Optional[Literal["grouped"]] = None,

        is_lora: bool = False,
        merge_lora: bool = False,
        adapter_name: Optional[str] = None,

        is_quantized: bool = False,
    ):
        self.model = model
        self.q_path, self.k_path = q_path, k_path
        self.attention_type = attention_type

        self.is_lora = is_lora
        self.merge_lora = merge_lora
        self.adapter_name = adapter_name  # default: merge *all* loaded
           adapters
```

```
    self.is_quantized = is_quantized

    self.layer_count = model.config.num_hidden_layers
    self.d  = model.config.hidden_size
    self.dh = self.d // model.config.num_attention_heads

# def layer_count(self) -> int:
#     prefix, _ = self.q_path.split("[layer_idx].")
#     return len(rgetattr(self.model, prefix))


def _get_weight(self, lin: torch.nn.Linear | bnb.nn.Linear4bit) -> torch.
   Tensor:
    """
    Get the weights of a linear layer, handling quantization if necessary
       .
    """
    if self.is_quantized:
        assert isinstance(lin, bnb.nn.Linear4bit), \
            f"Expected Linear4bit, got {type(lin)}"

        packed = lin.weight.data
        qs = lin.weight.quant_state
        weight = dequantize_4bit(packed, qs)

    else:
        assert isinstance(lin, torch.nn.Linear), \
            f"Expected Linear, got {type(lin)}"

        weight = lin.weight

    return weight


def matrix(self, layer_idx: int) -> torch.Tensor:
    """Return  Q   K  for the chosen layer as a *detached* tensor."""
    Wq = self._raw(self.q_path, layer_idx).T.detach()
    Wk = self._raw(self.k_path, layer_idx).T.detach()

    if self.attention_type == "grouped":
        # "Grouped" attention (Mistral-like) -> reshape keys before mm
        Wk = Wk.view(Wk.shape[0], self.dh, Wk.shape[1] // self.dh)
        rep = (Wq.shape[0] // self.dh) // Wk.shape[-1]
        Wk = Wk.repeat_interleave(rep, 0).view(Wq.shape[0], Wq.shape[0])
```

```
    return Wq @ Wk.T


def _raw(self, path: str, idx: int) -> torch.Tensor:
    """
    path: str
        Path to the matrix in the model, e.g. "model.layers[layer_idx].
            self_attn.q_proj".
        The [layer_idx] part will be replaced with the actual layer index
            .
    """

    layers_path, matrix_path = path.split("[layer_idx].")
    layer_module = rgetattr(self.model, layers_path)[idx]
    proj_module = rgetattr(layer_module, matrix_path)

    if not self.is_lora:
        assert not self._is_lora_layer(proj_module), \
            f"Module {proj_module} is a LoRA layer, but is_lora is False
                ."

        return self._get_weight(proj_module)

    else:
        delta_lora = self._delta_lora_weight(proj_module)

        if self.merge_lora:
            return self._get_weight(proj_module.base_layer) + delta_lora
        else:
            return delta_lora


@staticmethod
def _is_lora_layer(module) -> bool:
    try:
        from peft.tuners.lora import LoraLayer
        return isinstance(module, LoraLayer)
    except ImportError:
        return hasattr(module, "lora_A") and hasattr(module, "lora_B")


def _delta_lora_weight(self, module) -> torch.Tensor:
    delta = torch.zeros_like(self._get_weight(module.base_layer))

    if not self._is_lora_layer(module):
```

```
            return delta

        adapters = (
            [self.adapter_name] if self.adapter_name is not None
            else module.lora_A.keys()
        )
        for name in adapters:
            A = module.lora_A[name].weight # [r, in]
            B = module.lora_B[name].weight # [out, r]
            r = A.size(0)
            alpha = (
                module.lora_alpha[name]
                if isinstance(module.lora_alpha, dict)
                else module.lora_alpha
            )
            delta += (B @ A) * (alpha / r)

        return delta



class AttentionScorer(ABC):
    def __init__(self, extractor: AttentionExtractor):
        self.extractor = extractor

    def __call__(self, layers: Iterable[int] | int | None = None) -> List[
       float] | float:
        if layers is None:
            layers = range(self.extractor.layer_count)
        if isinstance(layers, int):
            layers = [layers]

        with torch.no_grad():
            scores = [self._score(self.extractor.matrix(i)) for i in layers]
        return scores if len(scores) > 1 else scores[0]

    @abstractmethod
    def _score(self, A: torch.Tensor) -> float: ...


class SymmetryScore(AttentionScorer):
    def _score(self, A: torch.Tensor) -> float:
        sym = 0.5 * (A + A.T)

        # somehow we should handle the case when A is zero
```

```
        denominator = (A ** 2).sum()
        if denominator == 0:
            return 1.0

        score = (sym ** 2).sum() / denominator

        # like in paper
        score = 2 * score - 1

        return score.detach().item()


class DirectionalityScore(AttentionScorer):
    def _score(self, A: torch.Tensor, *, num_std: int = 2) -> float:
        row, col = torch.norm(A, dim=1), torch.norm(A, dim=0)
        rt, ct = row.mean() + num_std*row.std(), col.mean() + num_std*col.std
           ()
        r_exc, c_exc = torch.sum(row[row > rt] - rt), torch.sum(col[col > ct]
            - ct)
        total = r_exc + c_exc
        score = 0.0 if total == 0 else (c_exc - r_exc) / total

        # like in paper
        score = -1 * score

        return score.detach().item()


================================================
FILE: src/metrics/uter.py
================================================
import torch
from typing import Tuple, List


class UTERScore:
    """
    Computes the positional Upper Triangle Energy Ratio (pUTER) score for
       attention layers.
    """
    def __init__(self):
        self._buffer: List[List[float]] = []

    def add_batch(self,
                  attentions: Tuple[torch.Tensor, ...],
```

```
                attn_mask:   torch.Tensor):
        self._buffer.append(self._uter_per_layer(attentions, attn_mask))


    def compute(self) -> List[float]:
        if not self._buffer:
            return []
        stack = torch.tensor(self._buffer)         # (micro_batches, L)
        return stack.mean(dim=0).tolist()          # layer means


    def reset(self):
        self._buffer.clear()


    @staticmethod
    def _uter_per_layer(
            attentions: Tuple[torch.Tensor, ...],
            attn_mask: torch.Tensor
    ) -> List[float]:
        mask_q = attn_mask.unsqueeze(1).unsqueeze(-1)
        mask_k = attn_mask.unsqueeze(1).unsqueeze(-2)
        valid  = (mask_q & mask_k).to(attentions[0].dtype)

        eps, uters = 1e-9, []
        for A in attentions:
            tri_u = torch.triu(A, diagonal=1)
            tri_valid = torch.triu(valid, diagonal=1)

            num   = ((tri_u * tri_valid) ** 2).sum(dim=(-1, -2))
            denom = ((A * valid) ** 2).sum(dim=(-1, -2))
            uters.append((num / (denom + eps)).mean().item())
        return uters


================================================
FILE: src/models/__init__.py
================================================
from .llama import biLlamaForMaskedLM, biLlamaForMaskedNTP
from .gemma3 import biGemma3ForMaskedLM, biGemma3ForMaskedNTP


MODELS_MAPPING = {
    'llama': {
        'mlm': biLlamaForMaskedLM,
```

```
        'mntp': biLlamaForMaskedNTP
    },
    'gemma3': {
        'mlm': biGemma3ForMaskedLM,
        'mntp': biGemma3ForMaskedNTP
    }
}


================================================
FILE: src/models/gemma3.py
================================================
from typing import Optional, Union
import copy

import torch
from torch import nn

from transformers.utils import logging
from transformers.cache_utils import Cache, StaticCache, HybridCache
from transformers.modeling_outputs import (
    BaseModelOutputWithPast,
    CausalLMOutputWithPast,
    MaskedLMOutput,
    TokenClassifierOutput
)
from transformers.processing_utils import Unpack

from transformers import Gemma3PreTrainedModel, Gemma3TextModel,
   Gemma3TextConfig, Gemma3ForCausalLM
from transformers.models.gemma3.modeling_gemma3 import (
    Gemma3TextScaledWordEmbedding,
    Gemma3DecoderLayer,
    Gemma3Attention,
    Gemma3MLP,
    Gemma3RMSNorm,
    Gemma3RotaryEmbedding, # TODO: try to customize this
)

from transformers.utils import (
    is_torch_flex_attn_available,
    logging,
)

from src.losses.losses import ForMaskedLMLoss, ForMaskedNTPLoss
```

```
logger = logging.get_logger(__name__)



class biGemma3Attention(Gemma3Attention):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.is_causal = False


class biGemma3DecoderLayer(Gemma3DecoderLayer):
    def __init__(self, config: Gemma3TextConfig, layer_idx: int):
        nn.Module.__init__(self)
        self.config = config
        self.hidden_size = config.hidden_size
        self.layer_idx = layer_idx

        self.self_attn = biGemma3Attention(config=config, layer_idx=layer_idx
            )

        self.mlp = Gemma3MLP(config)
        self.input_layernorm = Gemma3RMSNorm(self.hidden_size, eps=config.
            rms_norm_eps)
        self.post_attention_layernorm = Gemma3RMSNorm(self.hidden_size, eps=
            config.rms_norm_eps)
        self.pre_feedforward_layernorm = Gemma3RMSNorm(self.hidden_size, eps=
            config.rms_norm_eps)
        self.post_feedforward_layernorm = Gemma3RMSNorm(self.hidden_size, eps
            =config.rms_norm_eps)
        self.is_sliding = self.self_attn.is_sliding
        self.sliding_window = config.sliding_window


class biGemma3TextModel(Gemma3TextModel):
    def __init__(self, config: Gemma3TextConfig):
        Gemma3PreTrainedModel.__init__(self, config)
        self.padding_idx = config.pad_token_id
        self.vocab_size = config.vocab_size

        # Gemma3 downcasts the below to bfloat16, causing sqrt(3072)=55.4256
            to become 55.5. See https://github.com/huggingface/transformers/
            pull/29402
        self.embed_tokens = Gemma3TextScaledWordEmbedding(
```

```
            config.vocab_size, config.hidden_size, self.padding_idx,
                embed_scale=self.config.hidden_size**0.5
        )
        self.layers = nn.ModuleList(
            [biGemma3DecoderLayer(config, layer_idx) for layer_idx in range(
                config.num_hidden_layers)]
        )
        self.norm = Gemma3RMSNorm(config.hidden_size, eps=config.rms_norm_eps
            )
        self.rotary_emb = Gemma3RotaryEmbedding(config=config)
        self.gradient_checkpointing = False

        # TODO: raushan fix this after RoPE refactor. For now we hack it by
            reassigning thetas
        # when we want to create a local RoPE layer. Config defaults should
            hold values for global RoPE
        config = copy.deepcopy(config)
        config.rope_theta = config.rope_local_base_freq
        config.rope_scaling = {"rope_type": "default"}
        self.rotary_emb_local = Gemma3RotaryEmbedding(config=config)

        # Initialize weights and apply final processing
        self.post_init()


    @torch.no_grad()
    def _update_causal_mask(
        self,
        attention_mask: torch.Tensor,
        input_tensor: torch.Tensor,
        cache_position: torch.Tensor,
        past_key_values: HybridCache,
        output_attentions: bool,
    ):
        # Flash Attention currently doesn't support static cache but
            Gemma3Text work only with static cache.
        # So we will pass in attention mask as is in any case, not only when
            ther's padding. Then we'll use its shape
        # to cut out keys/values trailing 0 used in static cache. This
            workaround should be compile compatible
        # as it doesn't cause dynamic control issues.
        if self.config._attn_implementation == "flash_attention_2":
            return attention_mask

        dtype, device = input_tensor.dtype, input_tensor.device
```

```
    sequence_length = input_tensor.shape[1]
    if isinstance(past_key_values, (HybridCache, StaticCache)):
        target_length = past_key_values.get_max_cache_shape()
    else:
        target_length = attention_mask.shape[-1] if attention_mask is not
            None else input_tensor.shape[1]

    # In case the provided `attention` mask is 2D, we generate a causal
       mask here (4D).
    causal_mask = self.
       _prepare_4d_causal_attention_mask_with_cache_position(
        attention_mask,
        sequence_length=sequence_length,
        target_length=target_length,
        dtype=dtype,
        device=device,
        cache_position=cache_position,
        batch_size=input_tensor.shape[0],
    )
    return causal_mask


@staticmethod
def _prepare_4d_causal_attention_mask_with_cache_position(
    attention_mask: torch.Tensor,
    sequence_length: int,
    target_length: int,
    dtype: torch.dtype,
    cache_position: torch.Tensor,
    batch_size: int,
    **kwargs,
):
    min_dtype = torch.finfo(dtype).min

    # in original implementation:
    # causal_mask = torch.full(
    #     (sequence_length, target_length), fill_value=min_dtype, dtype=
       dtype, device=cache_position.device
    # )
    causal_mask = torch.zeros(
        (sequence_length, target_length), dtype=dtype, device=
            cache_position.device
    )

    # Commenting out next 2 lines to disable causal masking
```

```
# if sequence_length != 1:
#     causal_mask = torch.triu(causal_mask, diagonal=1)

causal_mask *= torch.arange(target_length, device=cache_position.
   device) > cache_position.reshape(-1, 1)
causal_mask = causal_mask[None, None, :, :].expand(batch_size, 1, -1,
    -1)
if attention_mask is not None:
    causal_mask = causal_mask.clone() # copy to contiguous memory for
        in-place edit

    # original implementation for 2D attention mask
    if attention_mask.dim() == 2:
        mask_length = attention_mask.shape[-1]
        padding_mask = causal_mask[:, :, :, :mask_length] +
           attention_mask[:, None, None, :].to(
            causal_mask.device
        )
        padding_mask = padding_mask == 0
        causal_mask[:, :, :, :mask_length] = causal_mask[:, :, :, :
           mask_length].masked_fill(
            padding_mask, min_dtype
        )

    # custom implementation for 4D attention mask
    elif attention_mask.dim() == 4:
        # backwards compatibility: we allow passing a 4D attention
           mask shorter than the input length with
        # cache. In that case, the 4D attention mask attends to the
           newest tokens only.
        if attention_mask.shape[-2] < cache_position[0] +
           sequence_length:
            offset = cache_position[0]
        else:
            offset = 0
        mask_shape = attention_mask.shape
        mask_slice = (attention_mask.eq(0.0)).to(dtype=dtype) *
           min_dtype
        causal_mask[
            : mask_shape[0],
            : mask_shape[1],
            offset : mask_shape[2] + offset,
            : mask_shape[3],
        ] = mask_slice
```

```
        return causal_mask


# initialized with a random head
class biGemma3ForMaskedLM(Gemma3PreTrainedModel):
    config_class = Gemma3TextConfig
    base_model_prefix = "language_model"

    def __init__(self, config):
        super().__init__(config)
        self.model = biGemma3TextModel(config)
        self.vocab_size = config.vocab_size
        self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=
            False)

        # Initialize weights and apply final processing
        self.post_init()

    def get_input_embeddings(self):
        return self.model.embed_tokens

    def set_input_embeddings(self, value):
        self.model.embed_tokens = value

    def get_output_embeddings(self):
        return self.lm_head

    def set_output_embeddings(self, new_embeddings):
        self.lm_head = new_embeddings

    def set_decoder(self, decoder):
        self.model = decoder

    def get_decoder(self):
        return self.model

    def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        past_key_values: Optional[Cache] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
        use_cache: Optional[bool] = None,
```

```
    output_attentions: Optional[bool] = None,
    output_hidden_states: Optional[bool] = None,
    cache_position: Optional[torch.LongTensor] = None,
    logits_to_keep: Union[int, torch.Tensor] = 0,
    **kwargs,
) -> MaskedLMOutput:
    output_attentions = output_attentions if output_attentions is not
       None else self.config.output_attentions
    output_hidden_states = (
        output_hidden_states if output_hidden_states is not None else
           self.config.output_hidden_states
    )

    # decoder outputs consists of (dec_features, layer_state, dec_hidden,
        dec_attn)
    outputs: BaseModelOutputWithPast = self.model(
        input_ids=input_ids,
        attention_mask=attention_mask,
        position_ids=position_ids,
        past_key_values=past_key_values,
        inputs_embeds=inputs_embeds,
        use_cache=use_cache,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        cache_position=cache_position,
        **kwargs,
    )

    hidden_states = outputs.last_hidden_state
    # Only compute necessary logits, and do not upcast them to float if
       we are not computing the loss
    slice_indices = slice(-logits_to_keep, None) if isinstance(
       logits_to_keep, int) else logits_to_keep
    logits = self.lm_head(hidden_states[:, slice_indices, :])

    loss = None
    if labels is not None:
        loss = ForMaskedLMLoss(logits=logits, labels=labels, vocab_size=
           self.config.vocab_size, **kwargs)

    return MaskedLMOutput(
        loss=loss,
        logits=logits,
        hidden_states=outputs.hidden_states,
        attentions=outputs.attentions,
```

```
        )


# with trained head
class biGemma3ForMaskedNTP(Gemma3ForCausalLM):
    def __init__(self, config):
        Gemma3PreTrainedModel.__init__(self, config)
        self.model = biGemma3TextModel(config)
        self.vocab_size = config.vocab_size
        self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=
            False)

        # Initialize weights and apply final processing
        self.post_init()

    def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        past_key_values: Optional[Cache] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
        use_cache: Optional[bool] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        cache_position: Optional[torch.LongTensor] = None,
        logits_to_keep: Union[int, torch.Tensor] = 0,
        **kwargs,
    ) -> CausalLMOutputWithPast:
        output_attentions = output_attentions if output_attentions is not
            None else self.config.output_attentions
        output_hidden_states = (
            output_hidden_states if output_hidden_states is not None else
                self.config.output_hidden_states
        )

        # decoder outputs consists of (dec_features, layer_state, dec_hidden,
            dec_attn)
        outputs: BaseModelOutputWithPast = self.model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            position_ids=position_ids,
            past_key_values=past_key_values,
```

```
            inputs_embeds=inputs_embeds,
            use_cache=use_cache,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            cache_position=cache_position,
            **kwargs,
        )

        hidden_states = outputs.last_hidden_state
        # Only compute necessary logits, and do not upcast them to float if
           we are not computing the loss
        slice_indices = slice(-logits_to_keep, None) if isinstance(
           logits_to_keep, int) else logits_to_keep
        logits = self.lm_head(hidden_states[:, slice_indices, :])

        loss = None
        if labels is not None:
            loss = ForMaskedNTPLoss(logits=logits, labels=labels, vocab_size=
               self.config.vocab_size, **kwargs)

        return CausalLMOutputWithPast(
            loss=loss,
            logits=logits,
            past_key_values=outputs.past_key_values,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


class biGemma3ForTokenClassification(Gemma3PreTrainedModel):
    config_class = Gemma3TextConfig
    base_model_prefix = "language_model"

    def __init__(self, config: Gemma3TextConfig):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.model = biGemma3TextModel(config)
        if getattr(config, "classifier_dropout", None) is not None:
            classifier_dropout = config.classifier_dropout
        elif getattr(config, "hidden_dropout", None) is not None:
            classifier_dropout = config.hidden_dropout
        else:
            classifier_dropout = 0.1
        self.dropout = nn.Dropout(classifier_dropout)
        self.score = nn.Linear(config.hidden_size, config.num_labels)
```

```
    # Initialize weights and apply final processing
    self.post_init()

def get_input_embeddings(self):
    return self.model.embed_tokens

def set_input_embeddings(self, value):
    self.model.embed_tokens = value


def forward(
    self,
    input_ids: Optional[torch.LongTensor] = None,
    attention_mask: Optional[torch.Tensor] = None,
    position_ids: Optional[torch.LongTensor] = None,
    past_key_values: Optional[Cache] = None,
    inputs_embeds: Optional[torch.FloatTensor] = None,
    labels: Optional[torch.LongTensor] = None,
    use_cache: Optional[bool] = None,
    output_attentions: Optional[bool] = None,
    output_hidden_states: Optional[bool] = None,
    **kwargs
) -> TokenClassifierOutput:
    r"""
    labels (`torch.LongTensor` of shape `(batch_size,)`, *optional*):
        Labels for computing the sequence classification/regression loss.
            Indices should be in `[0, ...,
        config.num_labels - 1]`. If `config.num_labels == 1` a regression
            loss is computed (Mean-Square loss), If
        `config.num_labels > 1` a classification loss is computed (Cross-
           Entropy).
    """

    outputs: BaseModelOutputWithPast = self.model(
        input_ids,
        attention_mask=attention_mask,
        position_ids=position_ids,
        past_key_values=past_key_values,
        inputs_embeds=inputs_embeds,
        use_cache=use_cache,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
    )
    sequence_output = outputs.last_hidden_state
```

```
        sequence_output = self.dropout(sequence_output)
        logits = self.score(sequence_output)

        loss = None
        if labels is not None:
            loss = self.loss_function(logits, labels, self.config)

        return TokenClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


================================================
FILE: src/models/llama.py
================================================
from typing import Optional, Union

import torch
from torch import nn

from transformers.utils import logging
from transformers.cache_utils import Cache
from transformers.modeling_outputs import (
    BaseModelOutputWithPast,
    CausalLMOutputWithPast,
    MaskedLMOutput
)
from transformers.processing_utils import Unpack

from transformers import LlamaModel, LlamaForCausalLM, LlamaPreTrainedModel,
   LlamaConfig
from transformers.models.llama.modeling_llama import (
    LlamaDecoderLayer,
    LlamaAttention,
    LlamaMLP,
    LlamaRMSNorm,
    LlamaRotaryEmbedding, # TODO: try to customize this
    LlamaForTokenClassification,
    KwargsForCausalLM
)

from transformers.modeling_attn_mask_utils import AttentionMaskConverter
```

```
from transformers.utils import (
    is_torch_flex_attn_available,
    logging,
)
if is_torch_flex_attn_available():
    from torch.nn.attention.flex_attention import BlockMask
    from transformers.integrations.flex_attention import
       make_flex_block_causal_mask

from src.losses.losses import ForMaskedLMLoss, ForMaskedNTPLoss


logger = logging.get_logger(__name__)


class biLlamaAttention(LlamaAttention):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.is_causal = False


class biLlamaDecoderLayer(LlamaDecoderLayer):
    def __init__(self, config: LlamaConfig, layer_idx: int):
        nn.Module.__init__(self)
        self.hidden_size = config.hidden_size

        self.self_attn = biLlamaAttention(config=config, layer_idx=layer_idx)

        self.mlp = LlamaMLP(config)
        self.input_layernorm = LlamaRMSNorm(config.hidden_size, eps=config.
           rms_norm_eps)
        self.post_attention_layernorm = LlamaRMSNorm(config.hidden_size, eps=
           config.rms_norm_eps)


class biLlamaModel(LlamaModel):
    def __init__(self, config: LlamaConfig):
        LlamaPreTrainedModel.__init__(self, config)
        self.padding_idx = config.pad_token_id
        self.vocab_size = config.vocab_size

        self.embed_tokens = nn.Embedding(config.vocab_size, config.
           hidden_size, self.padding_idx)
        self.layers = nn.ModuleList(
            [biLlamaDecoderLayer(config, layer_idx) for layer_idx in range(
```

```
            config.num_hidden_layers)]
        )
        self.norm = LlamaRMSNorm(config.hidden_size, eps=config.rms_norm_eps)
        self.rotary_emb = LlamaRotaryEmbedding(config=config)
        self.gradient_checkpointing = False

        # Initialize weights and apply final processing
        self.post_init()


    def _update_causal_mask(
        self,
        attention_mask: Union[torch.Tensor, "BlockMask"],
        input_tensor: torch.Tensor,
        cache_position: torch.Tensor,
        past_key_values: Cache,
        output_attentions: bool = False,
    ):
        if self.config._attn_implementation == "flash_attention_2":
            if attention_mask is not None and (attention_mask == 0.0).any():
                return attention_mask
            return None
        if self.config._attn_implementation == "flex_attention":
            if isinstance(attention_mask, torch.Tensor):
                attention_mask = make_flex_block_causal_mask(attention_mask)
            return attention_mask

        # For SDPA, when possible, we will rely on its `is_causal` argument
           instead of its `attn_mask` argument, in
        # order to dispatch on Flash Attention 2. This feature is not
           compatible with static cache, as SDPA will fail
        # to infer the attention mask.
        past_seen_tokens = past_key_values.get_seq_length() if
           past_key_values is not None else 0
        using_compilable_cache = past_key_values.is_compileable if
           past_key_values is not None else False

        # When output attentions is True, sdpa implementation's forward
           method calls the eager implementation's forward
        # if self.config._attn_implementation == "sdpa" and not
           using_compilable_cache and not output_attentions:
        #     if AttentionMaskConverter._ignore_causal_mask_sdpa(
        #         attention_mask,
        #         inputs_embeds=input_tensor,
        #         past_key_values_length=past_seen_tokens,
```

```
        #         is_training=self.training,
        #     ):
        #         return None

        dtype = input_tensor.dtype
        sequence_length = input_tensor.shape[1]
        if using_compilable_cache:
            target_length = past_key_values.get_max_cache_shape()
        else:
            target_length = (
                attention_mask.shape[-1]
                if isinstance(attention_mask, torch.Tensor)
                else past_seen_tokens + sequence_length + 1
            )

        # In case the provided `attention` mask is 2D, we generate a causal
           mask here (4D).
        causal_mask = self.
           _prepare_4d_causal_attention_mask_with_cache_position(
            attention_mask,
            sequence_length=sequence_length,
            target_length=target_length,
            dtype=dtype,
            cache_position=cache_position,
            batch_size=input_tensor.shape[0],
        )

        if (
            self.config._attn_implementation == "sdpa"
            and attention_mask is not None
            and attention_mask.device.type in ["cuda", "xpu", "npu"]
            and not output_attentions
        ):
            # Attend to all tokens in fully masked rows in the causal_mask,
               for example the relevant first rows when
            # using left padding. This is required by F.
               scaled_dot_product_attention memory-efficient attention path.
            # Details: https://github.com/pytorch/pytorch/issues/110213
            min_dtype = torch.finfo(dtype).min
            causal_mask = AttentionMaskConverter._unmask_unattended(
               causal_mask, min_dtype)

        return causal_mask

    @staticmethod
```

```
def _prepare_4d_causal_attention_mask_with_cache_position(
    attention_mask: torch.Tensor,
    sequence_length: int,
    target_length: int,
    dtype: torch.dtype,
    cache_position: torch.Tensor,
    batch_size: int,
    **kwargs,
):
    min_dtype = torch.finfo(dtype).min

    # in original implementation:
    # causal_mask = torch.full(
    #     (sequence_length, target_length), fill_value=min_dtype, dtype=
       dtype, device=cache_position.device
    # )
    causal_mask = torch.zeros(
        (sequence_length, target_length), dtype=dtype, device=
           cache_position.device
    )

    # Commenting out next 2 lines to disable causal masking
    # if sequence_length != 1:
    #     causal_mask = torch.triu(causal_mask, diagonal=1)

    causal_mask *= torch.arange(target_length, device=cache_position.
       device) > cache_position.reshape(-1, 1)
    causal_mask = causal_mask[None, None, :, :].expand(batch_size, 1, -1,
        -1)
    if attention_mask is not None:
        causal_mask = causal_mask.clone() # copy to contiguous memory for
            in-place edit

        # original implementation for 2D attention mask
        if attention_mask.dim() == 2:
            mask_length = attention_mask.shape[-1]
            padding_mask = causal_mask[:, :, :, :mask_length] +
               attention_mask[:, None, None, :].to(
                causal_mask.device
            )
            padding_mask = padding_mask == 0
            causal_mask[:, :, :, :mask_length] = causal_mask[:, :, :, :
               mask_length].masked_fill(
                padding_mask, min_dtype
            )
```

```
            # custom implementation for 4D attention mask
            elif attention_mask.dim() == 4:
                # backwards compatibility: we allow passing a 4D attention
                    mask shorter than the input length with
                # cache. In that case, the 4D attention mask attends to the
                    newest tokens only.
                if attention_mask.shape[-2] < cache_position[0] +
                    sequence_length:
                    offset = cache_position[0]
                else:
                    offset = 0
                mask_shape = attention_mask.shape
                mask_slice = (attention_mask.eq(0.0)).to(dtype=dtype) *
                    min_dtype
                causal_mask[
                    : mask_shape[0],
                    : mask_shape[1],
                    offset : mask_shape[2] + offset,
                    : mask_shape[3],
                ] = mask_slice

        return causal_mask


# initialized with a random head
class biLlamaForMaskedLM(LlamaPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.model = biLlamaModel(config)
        self.vocab_size = config.vocab_size
        self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=
            False)

        # Initialize weights and apply final processing
        self.post_init()

    def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        past_key_values: Optional[Cache] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
```

```
    use_cache: Optional[bool] = None,
    output_attentions: Optional[bool] = None,
    output_hidden_states: Optional[bool] = None,
    cache_position: Optional[torch.LongTensor] = None,
    logits_to_keep: Union[int, torch.Tensor] = 0,
    **kwargs: Unpack[KwargsForCausalLM],
) -> MaskedLMOutput:
    output_attentions = output_attentions if output_attentions is not
       None else self.config.output_attentions
    output_hidden_states = (
        output_hidden_states if output_hidden_states is not None else
           self.config.output_hidden_states
    )

    # decoder outputs consists of (dec_features, layer_state, dec_hidden,
        dec_attn)
    outputs: BaseModelOutputWithPast = self.model(
        input_ids=input_ids,
        attention_mask=attention_mask,
        position_ids=position_ids,
        past_key_values=past_key_values,
        inputs_embeds=inputs_embeds,
        use_cache=use_cache,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        cache_position=cache_position,
        **kwargs,
    )

    hidden_states = outputs.last_hidden_state
    # Only compute necessary logits, and do not upcast them to float if
       we are not computing the loss
    slice_indices = slice(-logits_to_keep, None) if isinstance(
       logits_to_keep, int) else logits_to_keep
    logits = self.lm_head(hidden_states[:, slice_indices, :])

    loss = None
    if labels is not None:
        loss = ForMaskedLMLoss(logits=logits, labels=labels, vocab_size=
           self.config.vocab_size, **kwargs)

    return MaskedLMOutput(
        loss=loss,
        logits=logits,
        hidden_states=outputs.hidden_states,
```

```
            attentions=outputs.attentions,
        )


# with trained head
class biLlamaForMaskedNTP(LlamaForCausalLM):
    def __init__(self, config):
        LlamaPreTrainedModel.__init__(self, config)
        self.model = biLlamaModel(config)
        self.vocab_size = config.vocab_size
        self.lm_head = nn.Linear(config.hidden_size, config.vocab_size, bias=
            False)

        # Initialize weights and apply final processing
        self.post_init()

    def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        past_key_values: Optional[Cache] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
        use_cache: Optional[bool] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        cache_position: Optional[torch.LongTensor] = None,
        logits_to_keep: Union[int, torch.Tensor] = 0,
        **kwargs: Unpack[KwargsForCausalLM],
    ) -> CausalLMOutputWithPast:
        output_attentions = output_attentions if output_attentions is not
            None else self.config.output_attentions
        output_hidden_states = (
            output_hidden_states if output_hidden_states is not None else
                self.config.output_hidden_states
        )

        # decoder outputs consists of (dec_features, layer_state, dec_hidden,
            dec_attn)
        outputs: BaseModelOutputWithPast = self.model(
            input_ids=input_ids,
            attention_mask=attention_mask,
            position_ids=position_ids,
            past_key_values=past_key_values,
```

```
            inputs_embeds=inputs_embeds,
            use_cache=use_cache,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            cache_position=cache_position,
            **kwargs,
        )

        hidden_states = outputs.last_hidden_state
        # Only compute necessary logits, and do not upcast them to float if
           we are not computing the loss
        slice_indices = slice(-logits_to_keep, None) if isinstance(
           logits_to_keep, int) else logits_to_keep
        logits = self.lm_head(hidden_states[:, slice_indices, :])

        loss = None
        if labels is not None:
            loss = ForMaskedNTPLoss(logits=logits, labels=labels, vocab_size=
               self.config.vocab_size, **kwargs)

        return CausalLMOutputWithPast(
            loss=loss,
            logits=logits,
            past_key_values=outputs.past_key_values,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


class biLlamaForTokenClassification(LlamaForTokenClassification):
    def __init__(self, config):
        LlamaPreTrainedModel.__init__(self, config)
        self.num_labels = config.num_labels
        self.model = biLlamaModel(config)
        if getattr(config, "classifier_dropout", None) is not None:
            classifier_dropout = config.classifier_dropout
        elif getattr(config, "hidden_dropout", None) is not None:
            classifier_dropout = config.hidden_dropout
        else:
            classifier_dropout = 0.1
        self.dropout = nn.Dropout(classifier_dropout)
        self.score = nn.Linear(config.hidden_size, config.num_labels)

        # Initialize weights and apply final processing
        self.post_init()
```

```
================================================
FILE: src/utils/other.py
================================================
import os
import random
import numpy as np
import torch

def set_seeds(seed):
    """Set seeds for reproducibility """
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)


================================================
FILE: sequence_labeling/unlp-2025/gemma/bi-gemma-mntp-eng.ipynb
================================================
# %% [markdown]
# # Setup

# %% [code]
import sys
sys.path.append('..')

# %% [code]
from dataclasses import dataclass
from tqdm.autonotebook import tqdm
from utils.utils import set_seeds
import wandb

import numpy as np
import torch
from datasets import Dataset
from transformers import (
    AutoModelForTokenClassification,
    AutoTokenizer,
    TrainingArguments
)
```

```
set_seeds(seed=42)
tqdm.pandas()

# %% [markdown]
# # Config

# %% [code]
@dataclass
class Config:
    data_path: str = "../../../data/unlp-2025/"
    cv_path: str = "../../../data/unlp-2025/cv_split.csv"

    pretrained: str = "google/gemma-3-4b-pt"
    adapter_args = {
        'run_id': "jyrcv5wz",
        'version': "latest"
    }
    max_length: int = 1024

    wandb_init_args = {
        'project': "sl-unlp-2025",
        'entity': "havlytskyi-thesis",
        'name': "gemma-3-4B--mntp-eng"
    }

config = Config()

# %% [markdown]
# # Training Arguments

# %% [code]
training_args = TrainingArguments(
    output_dir=f'./checkpoints/{config.wandb_init_args["name"]}',
    logging_dir=f'./logs/{config.wandb_init_args["name"]}',
    learning_rate=2e-5,
    weight_decay=0.01,
    lr_scheduler_type='cosine',
    warmup_ratio=0.0,
    num_train_epochs=5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=4,
    # gradient_accumulation_steps=1,
    bf16=True,
    report_to="wandb",
```

```
    optim='adamw_torch',
    eval_strategy='steps',
    save_strategy="steps",
    eval_steps=100,
    logging_steps=10,
    save_steps=100,
    save_total_limit=10,
    metric_for_best_model='eval_f1',
    greater_is_better=True,
    load_best_model_at_end=True,
)

# %% [markdown]
# # Instantiate the tokenizer & model

# %% [markdown]
# ## Base Model

# %% [code]
from src.models.gemma3 import biGemma3ForTokenClassification
from transformers import BitsAndBytesConfig
from peft import prepare_model_for_kbit_training


tokenizer = AutoTokenizer.from_pretrained(config.pretrained)
tokenizer.pad_token = tokenizer.eos_token


quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=False,
    bnb_4bit_compute_dtype=torch.bfloat16
)


base_model = biGemma3ForTokenClassification.from_pretrained(
    config.pretrained,
    id2label={0: 0, 1: 1},
    label2id={0: 0, 1: 1},
    quantization_config=quant_config
)
base_model = prepare_model_for_kbit_training(base_model)

# %% [markdown]
```

```
# ## Pretrained LoRA

# %% [code]
# to HF

out = !source ../../../scripts/wandb_to_hf.sh {config.adapter_args['run_id']}
    {config.adapter_args['version']}

hf_path_line = [l for l in out if l.startswith("RUN_NAME=")][0]
hf_path = hf_path_line.split("=", 1)[1]

config.adapter_args['subfolder'] = hf_path
hf_path

# %% [code]
from peft import PeftModel

model = PeftModel.from_pretrained(
    base_model,
    "nuinashco/GPT2Vec",
    subfolder=config.adapter_args['subfolder']
)

model = model.merge_and_unload()

# %% [markdown]
# ## New LoRA

# %% [code]
from peft import get_peft_model, LoraConfig, TaskType

lora_config = LoraConfig(
    r=64,  # the dimension of the low-rank matrices
    lora_alpha=128, # scaling factor for LoRA activations vs pre-trained
       weight activations
    lora_dropout=0.05,
    bias='none',
    inference_mode=False,
    task_type=TaskType.CAUSAL_LM,
    target_modules=['o_proj', 'v_proj', "q_proj", "k_proj", "gate_proj", "
       down_proj", "up_proj"]
)

model = get_peft_model(model, lora_config)
# Trainable Parameters
```

```
model.print_trainable_parameters()


# %% [markdown]
# # Data

# %% [code] {"scrolled":true}
import pandas as pd

df = pd.read_parquet(config.data_path + "train.parquet")
cv = pd.read_csv(config.cv_path)
df = df.merge(cv, on='id', how='left')

df_test = pd.read_csv(config.data_path + "test.csv")

# %% [code]
from utils.data import preprocess_df

df.trigger_words = df.trigger_words.apply(lambda x: [] if x is None else x)

is_valid_mask = (df.fold == 4)
df_train = df[~is_valid_mask].copy()
df_valid = df[is_valid_mask].copy()


df_train = preprocess_df(df_train, tokenizer=tokenizer, max_length=config.
    max_length)
df_valid = preprocess_df(df_valid, tokenizer=tokenizer, max_length=None)
df_test = preprocess_df(df_test, tokenizer=tokenizer, max_length=None)

# %% [code]
train_columns = list(df_train.seq_labels.iloc[0].keys()) +\
                ['content', 'trigger_words']
test_columns = list(df_train.seq_labels.iloc[0].keys()) + ['content']

ds_train = Dataset.from_pandas(df_train[train_columns].reset_index(drop=True)
    )
ds_valid = Dataset.from_pandas(df_valid[train_columns].reset_index(drop=True)
    )
ds_test = Dataset.from_pandas(df_test[test_columns].reset_index(drop=True))

# %% [markdown]
# # Train

# %% [code]
```

```
from itertools import chain

train_labels = df_train.labels.tolist() + df_valid.labels.tolist()
positive_class_balance = pd.Series(list(chain(*train_labels))).mean()

positive_class_balance

# %% [code]
from transformers import DataCollatorForTokenClassification
from utils.trainer import SpanIdentificationTrainer

data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)

trainer = SpanIdentificationTrainer(
    model=model,
    args=training_args,
    train_dataset=ds_train,
    eval_dataset=ds_valid,
    data_collator=data_collator,
    tokenizer=tokenizer,
    desired_positive_ratio=positive_class_balance
)

# %% [code] {"scrolled":true}
wandb.init(**config.wandb_init_args)

trainer.train()

# %% [markdown]
# # Inference

# %% [markdown]
# ## Checkpoint

# %% [code]
from utils.metric import score as char_f1
from utils.utils import inference_aggregation

FINETUNED_MODEL = f'./checkpoints/{config.wandb_init_args["name"]}/checkpoint
    -500'

# %% [code]
trainer._load_from_checkpoint(FINETUNED_MODEL)

# %% [markdown]
```

```
# ## Threshold Selection

# %% [code]
valid_preds = trainer.predict(ds_valid)
valid_metrics = trainer.compute_metrics((valid_preds.predictions, valid_preds
    .label_ids))

valid_metrics

# %% [code]
from utils.utils import find_class_balance_threshold

test_preds = trainer.predict(ds_test)
test_probabilities = torch.softmax(torch.tensor(test_preds.predictions), dim
    =-1).cpu().numpy()

test_distr_th = find_class_balance_threshold(
    desired_positive_ratio=positive_class_balance,
    probabilities=test_probabilities,
    labels=test_preds.label_ids
    )

print(test_distr_th)

# %% [code]
final_th = valid_metrics['thold']

# %% [markdown]
# ## CV-Score

# %% [code]
valid_probabilities = torch.softmax(torch.tensor(valid_preds.predictions),
    dim=-1).cpu().numpy()
valid_results = inference_aggregation(
    probabilities=valid_probabilities,
    labels=valid_preds.label_ids,
    offset_mappings=ds_valid['offset_mapping'],
    thold=final_th
)

# %% [code]
from copy import deepcopy

df_valid_gt = df[df.fold==4][['id', 'trigger_words']].reset_index(drop=True)
df_valid = deepcopy(df_valid_gt)
```

```
df_valid['trigger_words'] = valid_results

cv_score = char_f1(df_valid_gt, df_valid, row_id_column_name='id')
cv_score

# %% [markdown]
# ## Predict Test

# %% [code]
test_results = inference_aggregation(
    probabilities=test_probabilities,
    labels=test_preds.label_ids,
    offset_mappings=ds_test['offset_mapping'],
    thold=final_th
)

# %% [code]
df_test_gt = pd.read_csv(config.data_path + 'solution.csv')[['id', '
   trigger_words']]
df_test = deepcopy(df_test_gt)
df_test['trigger_words'] = test_results

test_score = char_f1(df_test_gt, df_test, row_id_column_name='id')
test_score
```

В ГРАФІЧНІ МАТЕРІАЛИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
Кафедра математичних методів системного аналізу
Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»

Двонаправлена адаптація великих мовних моделей-декодерів для обробки української природної мови

Керівник:
асистент кафедри ММСА,
Баздирев Антон Андрійович

Виконав:
студент IV курсу, групи КА-12
Гавлицький Іван Вікторович

Актуальність

- **Текст - головний формат даних**
  Стрімке зростання пошукових систем, чат-ботів та інших NLP-сервісів спричиняє експоненційне зростання попиту на моделі, здатні ефективно обробляти та враховувати контекст.

- **Фінансування та масштабування декодерних архітектур**
  Популярність генеративних систем (таких як ChatGPT), а також концепція емерджентності нейронних мереж як ключового напрямку досягнення AGI призводить до концентрації фінансових ресурсів саме на односпрямованих (декодерних) моделях, дозволяючи їм досягати масштабів до трильйонів параметрів.

- **Економічна неефективність тренування енкодерів з нуля**
  Навчання еквівалентних за потужністю енкодерних моделей за допомогою задачі MLM вимагає значно більшої кількості навчальних кроків через розрідженість маскування токенів, а також через відсутність готових чекпойнтів аналогічних масштабів.

Ідея: Перетворення вже навчених декодерних моделей на енкодерні забезпечуючи масштаб, двосторонній контекст та мінімальні додаткові витрати.

Цілі роботи

Об'єкт дослідження:
Моделі обробки природної мови на основі трансформерної архітектури.

Предмет дослідження:
Підходи до адаптації авторегресивних моделей для двонаправленого розуміння тексту.

Мета роботи:
Дослідити ефективність стратегій двонаправленої адаптації авторегресивних мовних моделей.

Постановка задачі

1. Огляд предметної області та теоретичного апарата
2. Дослідження архітектурних особливостей трансформерів
3. Реалізація фреймворку для двонаправленої адаптації моделей
4. Оцінка ефективності модифікацій на практичних завданнях
5. Дослідження експериментальних внутрішні метрики адаптації
6. Аналіз отриманих результатів

4

Transformer (2017)

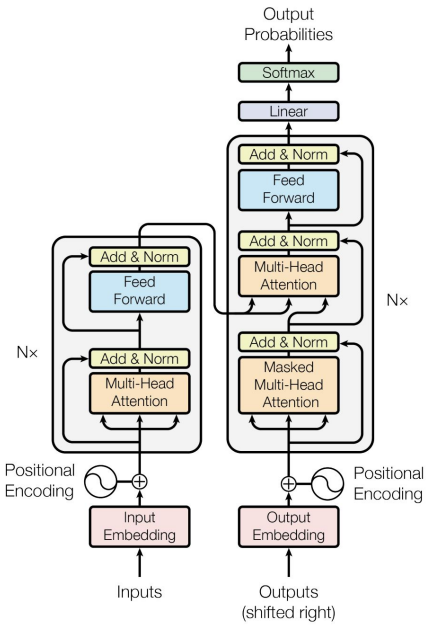
Модель була запропонована для вирішення **задачі перекладу** та повної паралелізації обчислень та вирішення інших проблем RNN.

Архітектура будувалася з двох компонент:

1. **Encoder** перетворює вхідний текст у контекстні вектори,
2. **Decoder** генерує вихід, запитуючи цей контекст.

Основний механізм: **Self-Attention**

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}.$$



5

Decoder-only

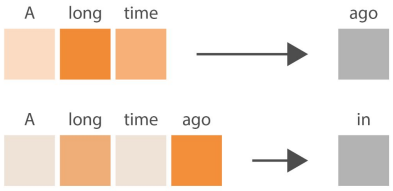
Використовують лише декодерну частину Transformer.

Навчаються за допомогою **авторегресивної задачі** (Causal Language Modeling):

$$\mathcal{L}_{\text{CLM}} = -\sum_{i=0}^{n} \log P(x_i \mid \mathbf{x}_{<i})$$

Реалізується через **Self-Attention з каузальною маскою** для запобігання «підгляданню» у майбутні токени:

$$\text{Attention}_{\text{causal}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}.$$



	On	the	river	bank
On	0.7	0	0	0
the	0.1	0.6	0	0
river	0.3	0.03	0.6	0
bank	0.05	0.1	0.05	0.8

Сучасні декодери

До сьогодні архітектура зазнала значних модифікацій на рівні окремих компонентів, серед яких:

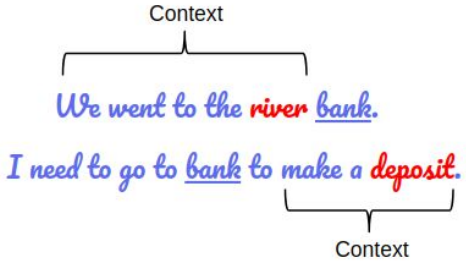
1. **Удосконалена нормалізація**: RMS LayerNorm для стабілізації процесу оптимізації.
2. **Ротаційні позиційні кодування (RoPE)**: покращення репрезентації послідовностей і точності моделі на довгих контекстах.
3. **Активація SwiGLU в MLP-блоках**: підвищення виразності моделі.

$X = S\,W_{\text{embeddings}},$	Inputs & Sampling
Repeat L times:	
$Y = \frac{X}{\sqrt{\frac{1}{d}\sum_i X_i^2 + \epsilon}}\, w_a,$	Pre RMS LayerNorm
$Q = YW_q, \quad K = YW_k, \quad V = YW_v,$	
$Q \leftarrow Q\cos + Q_R \sin,$	RoPE Embeddings
$K \leftarrow K\cos + K_R \sin,$	RoPE Embeddings
$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V,$	Multi-head Attention
$X \leftarrow X + ZW_o,$	Residual Connection
$Y = \frac{X}{\sqrt{\frac{1}{d}\sum_i X_i^2 + \epsilon}}\, w_b,$	Post RMS LayerNorm
$Z = f(YW_{\text{gate}})\,(YW_{\text{up}}),$	MLP / SwiGLU
$X \leftarrow X + ZW_{\text{down}},$	Residual Connection
end repeat.	
$X = \frac{X}{\sqrt{\frac{1}{d}\sum_i X_i^2 + \epsilon}}\, w_L,$	RMS LayerNorm
$X = XW_{\text{lm\_head}},$	Logits

Звуження контексту через каузальну маску

Проблема: Авторегресивна (каузальна) маска в декодерних архітектурах архітектурно обмежує модель — вона не має доступу до майбутнього (правого) контексту.

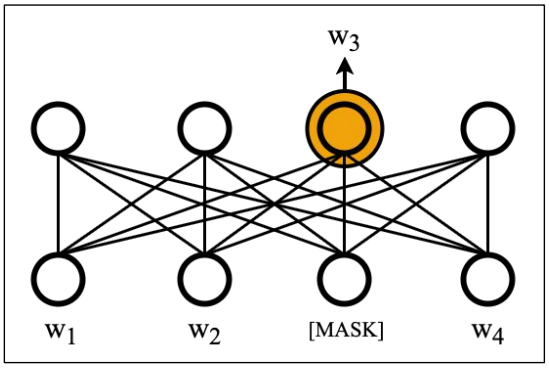
Наслідок: Це фундаментально унеможливлює повноцінне розуміння тексту, що критично для задач, де значення залежить від інформації після поточної позиції (маркування послідовностей, створення семантичних ембедингів, повнотекстова класифікація).



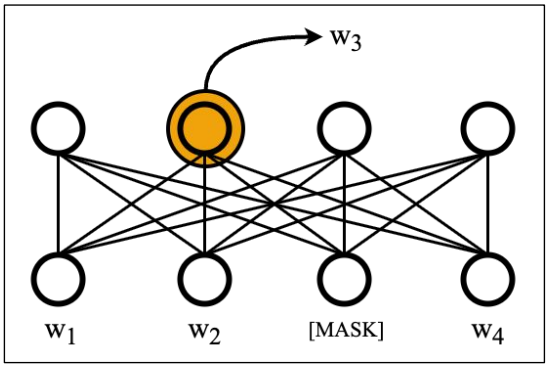
8

Методи двонаправленої адаптації

Masked Language Modeling
Пряме застосування задачі тренування енкодерних моделей.



Masked Next Token Prediction
Адаптація MLM до декодерних архітектур, зберігаючи генеративні властивості.



9

Masked Language Modeling

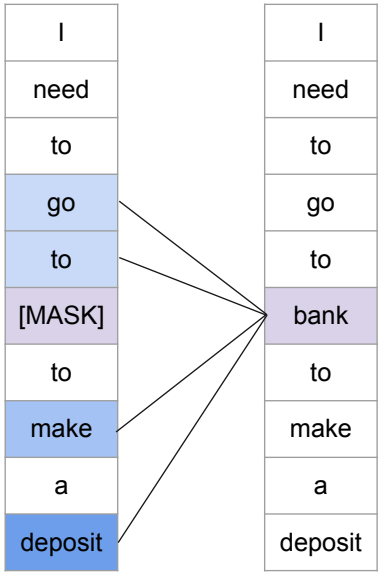
Мета: навчити модель відновлювати випадково замасковані токени у послідовності, використовуючи *весь* контекст.

Формулювання:

1. Дано послідовність токенів: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.
2. Кожен токен x_i замасковується незалежно з ймовірністю p_{mask} (зазвичай $p_{\text{mask}} = 0.15$).
3. Набір маскованих позицій: $M \sim \text{Bernoulli}(p_{\text{mask}})$.
4. Масковані токени замінюються спеціальним токеном [MASK].
5. Функція втрат:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in M} \log P(x_i \mid \mathbf{x}_{\setminus M})$$

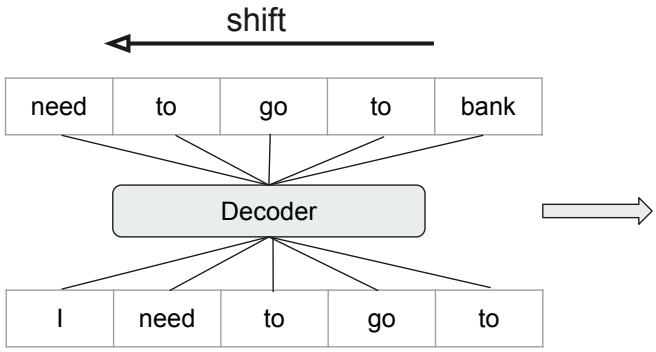
де $\mathbf{x}_{\setminus M}$ — послідовність з маскованими токенами.



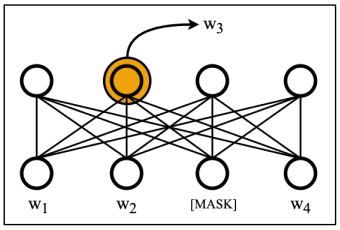
10

Masked Next Token Prediction

Ідея: адаптація задачі MLM до моделей навчених за допомогою Causal Language Modeling.
На практиці **CLM** реалізується як **класифікація** кожного кожного **логіту** трансформеру із таргетним значенням рівним вхідній послідовності **посунутій лівіше на одну позицію**. Відповідна модифікація застосовується й до задачі MLM, отримуючи в результаті задачу Masked Next Token Prediction (**MNTP**).



$$\mathcal{L}_{\text{MNTP}} = -\sum_{i \in \mathcal{M}, i>1} \log P_\theta(x_i | \tilde{\boldsymbol{x}}, \text{pos} = i-1).$$



11

Дані для переднавчання

Корпус для переднавчання з двонаправленою адаптацією було сформовано на основі Wikipedia (hf: wikimedia/wikipedia) для української та англійської мов. Такий вибір дозволяє дослідити вплив мовної належності корпусу на результати адаптації моделі для задач української мови.

Аналізується, чи необхідна мона відповідність корпусу адаптації цільовій задачі, або ж вирішальним чинником є лише сама наявність двонаправленого навчання.

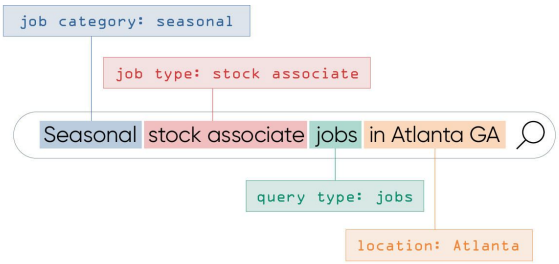
WIKIPEDIA

12

Оцінка якості адаптації

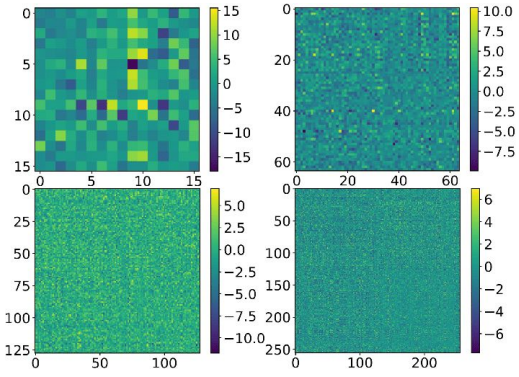
Зовнішні метрики

Якість адаптації оцінюється на задачах, де критичною є двонаправленість контексту (напр. маркування послідовності).



Внутрішні метрики

Для інтерпретації змін моделі аналізуються вагові зсуви та характеристики матриць уваги.



13

UNLP 2025 Shared Task

Критеріями відбору набору даних для оцінки зовнішних метрик:

1. Золотий стандарт розмітки
2. Стандартизоване розподілення даних
3. Актуальність та дослідницька релевантність

Спираючись на окреслені критерії, було обрано один з **UNLP 2025 Shared Task** – датасет для **виявлення маніпулятивних технік** у **цифрових медіа**.
Містить 9,500 дописів з Telegram, розмічених медіа-експертами. Особливість датасету полягає у необхідності **character-level** маркування маніпулятивних фрагментів, що пред ставляє особливий виклик для моделей та дозволяє оцінити їх здатність до точної локалізації цільових елементів тексту.

УКРАЇНЦІВ ЗМУСЯТЬ СТАТИ НА ОБЛІК ЗА КОРДОНОМ? Українців за кордоном, які не стали на облік, хочуть позбавляти банківських та консульських послуг, — депутат Вадим Івченко. «Банк запросить також нову ідентифікацію, але в цій ідентифікації має бути це посвідчення, безпосередньо цей військовий облік. Це означає, що можуть вимкнутись навіть банківські картки.

14

UNLP 2025 Shared Task Results

Табл. 4.1. Порівняльні результати на датасеті UNLP 2025 Shared Task

Модель	**Переднавчання**	**Test F_1**
biLlama-3.2-3B	MNTP (Ukrainian)	**0.624**
biLlama-3.2-3B	MLM (Ukrainian)	0.621
biLlama-3.2-3B	MNTP (English)	0.616
biLlama-3.2-3B	MLM (English)	0.612
biLlama-3.2-3B	-	0.617
Llama-3.2-3B	-	0.593
biGemma-3-4B	MNTP (Ukrainian)	**0.633**
biGemma-3-4B	MLM (Ukrainian)	**0.633**
biGemma-3-4B	MNTP (English)	**0.633**
biGemma-3-4B	MLM (English)	0.629
biGemma-3-4B	-	0.539
Gemma-3-4B	-	0.605
mDeBERTaV3	-	0.604

15

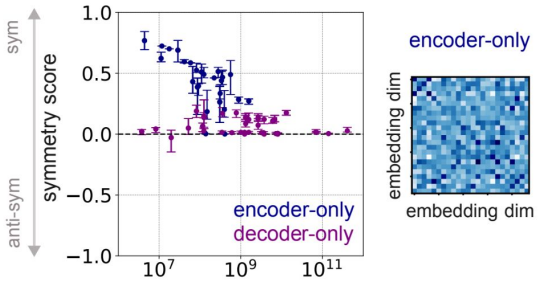
Attention Geometry

Мерика Симетричності

$$\mathcal{S}(\mathbf{W}) = \|\mathbf{W}^{(s)}\|_F^2 \; - \; \|\mathbf{W}^{(a)}\|_F^2$$

$\mathbf{W}^{(s)} = \frac{\mathbf{W} + \mathbf{W}^T}{2}$ (симетрична компонента),

$\mathbf{W}^{(a)} = \frac{\mathbf{W} - \mathbf{W}^T}{2}$ (кососиметрична компонента).

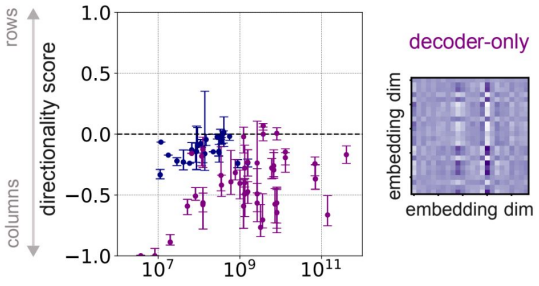


Мерика Направленості

$$\mathcal{D}(\mathbf{W}) = \frac{R - C}{R + C}$$

$C = \sum_{j:\|\mathbf{W}_{:j}\|_2 > c_\mu + \lambda c_\sigma} \|\mathbf{W}_{:j}\|_2$ (сумарна норма важких стовпців)

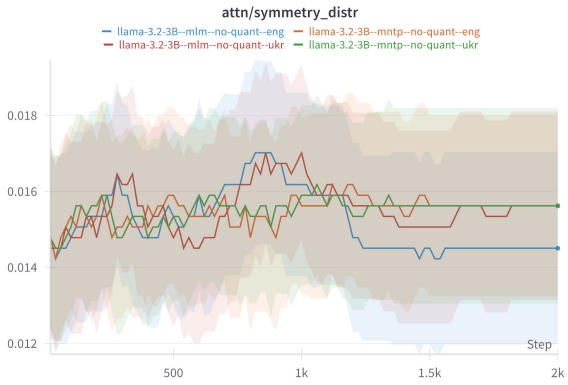
$R = \sum_{i:\|\mathbf{W}_{i:}\|_2 > r_\mu + \lambda r_\sigma} \|\mathbf{W}_{i:}\|_2$ (сумарна норма важких рядків).

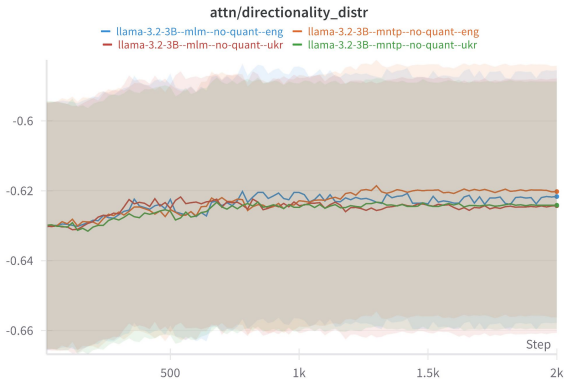


16

Attention Geometry Results

Стабільність метрик протягом всіх експериментів на відміну від очікуваної еволюції у бік збільшення симетричності та зменшення направленості.





17

Висновки

- Зняття каузального маскування в авторегресивних моделях суттєво покращує їхню ефективність на задачах розуміння тексту. Для деяких моделей навіть мінімальні архітектурні зміни без додаткового навчання забезпечують значний приріст продуктивності.
- Відповідні моделі значно перевершують справжні енкодери, що остаточно доводить доцільність двонаправлених модифікацій.
- Експериментальні метрики двонаправленості виявилися нечутливість до змін поведінки моделі та потребують подальшого розгляду.

Подальші дослідження

- Аналіз додаткових модифікацій архітектури: альтернативні позиційні енкодинги, пулінг послідовностей та механізму уваги.
- Розширення експериментального бенчмарку на задачі семантичних ембедингів.
- Продовження аналізу внутрішніх метрик процесу адаптації трансформерних моделей.
- Дослідження патернів активацій матриці уваги.

Перелік джерел посилання

1. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. **Attention is all you need.** Advances in Neural Information Processing Systems. 2017. Vol. 30.
2. Lv A., Zhang K., Xie S., Tu Q., Chen Y., Wen J.-R., Yan R. **An analysis and mitigation of the reversal curse.** arXiv preprint arXiv:2311.07468. 2023.
3. BehnamGhader P., Adlakha V., Mosbach M., Bahdanau D., Chapados N., Reddy S. **LLM2Vec: Large language models are secretly powerful text encoders.** arXiv preprint arXiv:2404.05961. 2024.
4. Suganthan P., Moiseev F., Yan L., Wu J., Ni J., Han J., Zitouni I., Alfonseca E., Wang X., Dong Z. **Adapting decoder-based language models for diverse encoder downstream tasks.** arXiv preprint arXiv:2503.02656. 2025.
5. **UNLP 2025 shared task on detecting social media manipulation.** GitHub repository. 2025. Available at: https://github.com/unlp-workshop/unlp-2025-shared-task (last accessed 09.06.2025).
6. Saponati M., Sager P., Aceituno P. V., Stadelmann T., Grewe B. **The underlying structures of self-attention: Symmetry, directionality, and emergent dynamics in Transformer training.** arXiv preprint arXiv:2502.10927. 2025.

Дякую за увагу