**ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ**
**Навчально-науковий інститут економіки, управління, права та інформаційних технологій**
**Кафедра інформаційних систем та технологій**

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Застосування локальних великих мовних моделей для автоматизованого перекладу текстів»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бз_2022
Готра Михайло Михайлович
Керівник: Слюсар Вадим Іванович
Рецензент: Муравльов Володимир
В’ячеславович

**Полтава – 2026 року**

**ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ**
**Навчально-науковий інститут економіки, управління, права та інформаційних технологій**
**Кафедра інформаційних систем та технологій**

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
_____ Юрій УТКІН
«10» квітня 2025 року

# З А В Д А Н Н Я
# НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

**Готрі Михайлу Михайловичу**

1. Тема кваліфікаційної роботи:
«Застосування локальних великих мовних моделей для автоматизованого перекладу текстів»,
Керівник роботи: д. т. н., професор, професор кафедри інформаційних систем та технологій Слюсар Вадим Іванович.
Затверджено наказом закладу вищої освіти від «19» березня 2026 року № 282-ст

2. Строк подання здобувачем вищої освіти роботи «26» травня 2026 року

3. Вихідні дані до роботи: науково-технічна література, аналітичні джерела мережі Інтернет, що містять інформацію про LLM, архітектури Gemma 2(3), техніки RAG, локальні програмні засоби для роботи з LLM, україномовні моделі MamayLM і Lapa LLM

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1 Аналіз особливостей великих мовних моделей
Розділ 2. Розробка системи автоматизованого локального перекладу
Розділ 3. Рекомендації щодо практичної реалізації системи автоматизованого локального перекладу

5. Перелік графічного матеріалу: схеми, рисунки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання отримав |
| Розрахунок економічної ефективності результатів дослідження | Шкурупій О. В., д. е. н., професор, професор кафедри економіки та публічного управління | 24.03.2026 р. | 09.04.2026 р. |

7. Дата видачі завдання «10» квітня 2025 року

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Строк виконання етапів кваліфікаційної роботи | Примітка |
|---|---|---|---|
| 1 | Вибір і затвердження теми роботи | 01.04.2025 р. | |
| 2 | Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу | 02.04.2025 р. – 10.04.2025 р. | |
| 3 | Опрацювання джерел інформації | 03.06.2025 р. – 15.06.2025 р. | |
| 4 | Збір, вивчення і обробка інформації, необхідної для виконання роботи | 16.06.2025 р. – 04.07.2025 р. | |
| 5 | Виконання теоретико-методологічного розділу роботи | 08.09.2025 р. – 15.11.2025 р. | |
| 6 | Виконання дослідницько-аналітичного розділу роботи | 16.11.2025 р. – 30.01.2026 р. | |
| 7 | Виконання проектно-рекомендаційного розділу роботи | 01.02.2026 р. – 21.03.2026 р. | |
| 8 | Розрахунок економічної ефективності результатів дослідження | 24.03.2026 р. – 09.04.2026 р. | |
| 9 | Оформлення тексту роботи | 10.04.2026 р. – 25.05.2026 р. | |
| 10 | Попередній захист роботи на кафедрі | 26.05.2026 р. | |
| 11 | Доопрацювання роботи з урахуванням зауважень і пропозицій | 27.05.2026 р. - 28.05.2026 р. | |
| 12 | Нормоконтроль | 29.05.2026 р. | |
| 13 | Захист кваліфікаційної роботи | 03.06.2026 р. | |

**Здобувач вищої освіти** **Михайло ГОТРА**

**Керівник роботи** **Вадим СЛЮСАР**

**ЗМІСТ**

# ВСТУП

*Актуальність* теми кваліфікаційної роботи підтверджується необхідністю локального застосування великих мовних моделей (LLM) для перекладу. На відміну від класичних систем машинного перекладу, вони виконують переклад на основі глибокого аналізу контексту, стилю та змісту вхідного тексту. Такі локальні рішення дають змогу зберігати повний контроль над даними, моделлю та процесом обробки інформації, а також підвищити стійкість до зовнішніх ризиків.

Водночас, використання LLM для створення системи автоматизованого локального перекладу потребує відповідного програмного забезпечення, оптимізації моделей. Все це свідчить про актуальність теми роботи.

*Метою* кваліфікаційної роботи є підвищення ефективності обробки природної мови за рахунок використання локалізації open source LLM.

*Завданнями* кваліфікаційної роботи є:

– обґрунтування вибору інструментарію для реалізації локальних LLM;

– визначення особливостей використання україномовних LLM;

– формування рекомендацій щодо практичної реалізації системи автоматизованого локального перекладу;

– техніко-економічне обґрунтування прийнятих рішень.

*Об'єктом дослідження* є процес обробки природної мови.

*Предметом дослідження* є програмні засоби локального перекладу.

*Методами* дослідження є в рамках визначення інструментарію для локального перекладу на основі мовних моделей та техніко-економічного обґрунтування прийнятих рішень використовувався аналітичний метод досліджень, а для локалізації україномовних LLM – моделювання.

*Інформаційна база* кваліфікаційної роботи сформована з Інтернет-ресурсів, що місять інформацію про LLM, архітектури Gemma 2(3), техніки

RAG, локальні програмні засоби для роботи з LLM, україномовні моделі MamayLM і Lapa LLM.

*Практична значущість* роботи полягає у розробці рекомендацій щодо практичної реалізації системи автоматизованого локального перекладу; pipeline використання MamayLM у LM Studio, в тому числі, з RAG, – можуть бути використані для подальших досліджень за даною тематикою та при проектуванні локальних сервісів NLP.

*Апробація результатів* відбувалася в рамках XXI щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету (22 квітня 2026 р., м. Полтава).

За результатами досліджень здійснено публікацію тез доповідей.

*Структура кваліфікаційної роботи* логічно пов’язана з завданнями досліджень і містить вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг пояснювальної записки кваліфікаційної роботи складає 65 сторінок формату А4. Вона містить 12 рисунків і 4 таблиці.

# РОЗДІЛ 1
# АНАЛІЗ ОСОБЛИВОСТЕЙ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

## 1.1 Особливості застосування локальних мовних моделей

У багатьох компаніях співробітники стикаються із завданнями, де потрібно зібрати, класифікувати та систематизувати великі обсяги інформації. У цьому рутина – одне з головних перешкод підвищення продуктивності. Наприклад, згідно з [1], близько 40 % робочого часу працівників можна автоматизувати за допомогою великих мовних моделей (Large Language Models, LLM) [2]. Вони ефективні у виконанні завдань, що повторюються, таких як: класифікація текстів, переклад, складання звітів, сумаризація, чищення даних. Але впровадити їх у великій компанії може бути проблематично. Це пов’язано з кількома чинниками.

1. Політика безпеки та захист даних. Великі компанії обробляють мільйони записів персональних даних користувачів: П.І.Б., телефони, адреси, платіжні дані. У деяких випадках, передача таких даних за кордон суворо заборонена без дотримання виняткових умов.

2. Комерційна таємниця. Дані про попит, пошукові запити, сезонність та географію угод – це важливі бізнес-активи. Аналіз цих даних зовнішніми інструментами ШІ може розкрити вразливість над ринком.

3. Ризики витоків. Закордонні провайдери ШІ зобов'язані передавати дані владі своїх країн, наприклад, згідно із законом FISA до США. Навіть якщо дані «анонімні», існує ризик, що користувачів можна ідентифікувати за непрямою інформацією – історії покупок або геолокації.

Через ці та інші причини компанії шукають інші способи впровадження ШІ в щоденну роботу, безпечного використання нейронних мереж та обходу обмежень. Такий підхід можна зарахувати до «In-house solutions». Це програмні, технічні чи організаційні рішення, які розробляються, впроваджуються та підтримуються всередині самої компанії, без залучення

сторонніх підрядників чи готових зовнішніх продуктів. Вони розробляються внутрішньою командою (IT-відділом), повністю адаптовані під конкретні завдання бізнесу та забезпечують високий рівень контролю та безпеки. Для великих компаній у висококонкурентних нішах повний контроль над даними та моделлю – не просто перевага, а стратегічна необхідність. In-house solutions має різні переваги: дані та розробок залишаються всередині компанії; модель кастомізується саме за необхідною специфікою; рішення незалежні від вендорів та їх політики та будуть стійкими у довгостроковій перспективі.

При цьому варто пам'ятати, що попри всі плюси такі рішення вимагають від компаній певних потужностей. Наприклад, потрібно інвестувати у власні GPU-кластери та MLOps-інфраструктуру, розвивати внутрішню експертизу в області машинного навчання (Machine Learning, ML) [3] або обробки природньої мови (Natural Language Processing, NLP) [4], створювати чи адаптувати відкриті моделі та навчати їх лише на внутрішніх даних в ізольованому контурі.

Таким чином, локальні LLM знаходять широке застосування у задачах автоматичного перекладу, обробки технічної документації, створення інтелектуальних асистентів, аналізу текстових даних та автоматизації бізнес-процесів. Зокрема, їх використання є доцільним у системах, де критичними є швидкість обробки, автономність та безпека інформації.

Локальні LLM є класом моделей ШІ, які можуть функціонувати без підключення до віддалених серверів, виконуючи обробку даних безпосередньо на локальному обчислювальному пристрої користувача. Такі моделі базуються на сучасних досягненнях у галузі обробка природної мови та використовують архітектуру трансформерів для аналізу та генерації тексту.

Основною особливістю локальних LLM є їх автономність, що забезпечує підвищений рівень конфіденційності даних, оскільки обробка інформації не потребує передачі запитів до зовнішніх сервісів. Це особливо важливо при роботі з чутливою або корпоративною інформацією. Крім того,

локальні моделі дозволяють працювати в умовах обмеженого або відсутнього доступу до мережі Інтернет, що розширює можливості їх застосування у вбудованих системах, польових умовах та на периферійних пристроях.

Сучасні локальні LLM оптимізуються для роботи на пристроях, що мають обмежені ресурс (ПК або одноплатні системи, наприклад, Raspberry Pi [5]). Це досягається завдяки використанню методів зменшення розміру моделей, зокрема квантизації та дистиляції знань, що дозволяє знизити вимоги до оперативної пам’яті та обчислювальної потужності без значної втрати якості результатів.

В цілому, до переваг локальних мовних моделей належать: незалежність від хмарних сервісів, контроль над даними, відсутність затримок, пов’язаних із мережею, а також можливість гнучкого налаштування під конкретні задачі. Водночас їх недоліками є обмежена продуктивність порівняно з великими хмарними моделями та необхідність оптимізації під конкретне апаратне забезпечення.

## 1.2 Використання трансформерних моделей для автоматизованого перекладу текстів

Переклад за допомогою LLM є сучасним підходом до автоматизованого опрацювання текстів, який принципово відрізняється від класичних методів машинного перекладу. На відміну від статистичних або правил-орієнтованих систем, LLM реалізують переклад як задачу генерації тексту на основі глибокого семантичного аналізу вхідної послідовності.

У загальному вигляді, переклад у LLM формулюється як задача типу sequence-to-sequence, де модель отримує текст мовою-джерелом і генерує текст мовою-призначення. При цьому модель не виконує послівний переклад, а створює нову послідовність, яка є семантично еквівалентною до оригіналу. Це дозволяє враховувати контекст, стиль і домен тексту, що суттєво підвищує

якість результату. Процес перекладу в трансформерних моделях [6] складається з кількох основних етапів. На першому етапі виконується токенізація вхідного тексту – розбиття його на елементарні одиниці (токени), які можуть відповідати словам, частинам слів або символам. Далі ці токени перетворюються у векторні представлення, що подаються на вхід моделі.

Ключовим етапом є побудова контекстного представлення за допомогою self-attention (механізм самоуваги). Завдяки цьому механізму модель здатна визначати взаємозв'язки між словами незалежно від їх розташування у реченні. Наприклад, при перекладі технічного тексту модель може враховувати, що слово «antenna» пов'язане з поняттями «частота» та «робота пристрою», що дозволяє обрати коректний переклад у відповідному домені. Після формування контекстного представлення відбувається етап декодування, під час якого модель генерує переклад послідовно, токен за токеном. Кожен наступний елемент перекладу залежить як від уже згенерованих токенів, так і від повного контексту вхідного тексту. Таким чином, переклад формується як імовірнісна послідовність, що максимізує умовну ймовірність правильного вихідного тексту.

Математично цей процес можна описати як оцінку умовної ймовірності $P(Y \mid X)$, де $X$ – вхідний текст, а $Y$ – переклад. Модель обирає таку послідовність $Y$, яка має найбільшу ймовірність за заданого $X$.

Суттєвою перевагою LLM є здатність враховувати контекст, стиль та предметну область тексту. Наприклад, слово «current» може перекладатися як «струм» або «поточний» залежно від контексту. У технічних дисциплінах це має критичне значення, оскільки неправильний переклад термінів може призвести до втрати змісту.

Разом із тим, використання LLM має і певні обмеження. До основних проблем належать можливість виникнення так званих «галюцинацій», коли модель генерує некоректну або вигадану інформацію, а також нестабільність термінології при перекладі великих обсягів тексту. Для вирішення цих проблем у практичних системах застосовують додаткові підходи, зокрема

використання словників термінів (glossary), розбиття тексту на частини (chunking) та постобробку результатів перекладу.

В основі LLM лежить архітектура Transformer, яка забезпечує ефективну обробку послідовностей і дозволяє моделі враховувати довгі залежності між словами. Саме ця архітектура зробила можливим суттєве підвищення якості машинного перекладу (рис. 1.1).

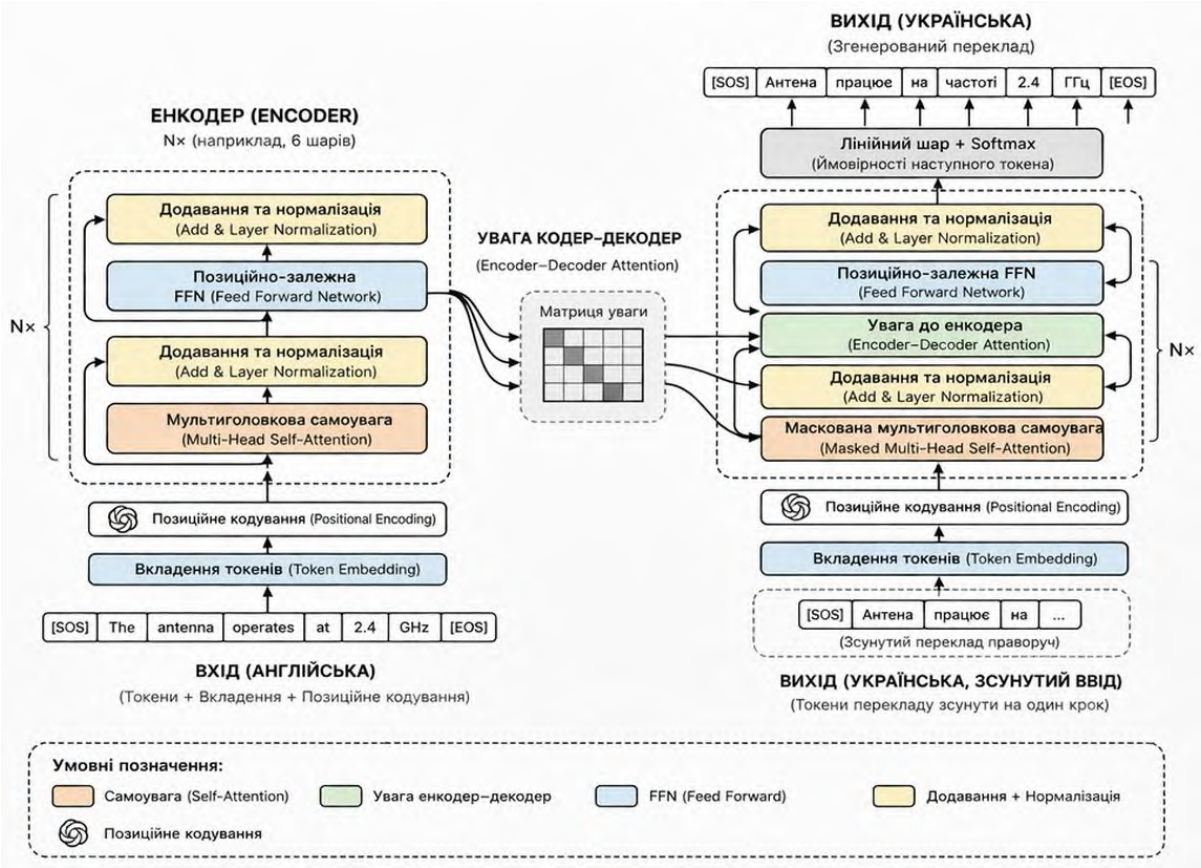


Рисунок 1.1 – Архітектура моделі Transformer для задачі перекладу

Архітектура Transformer складається з енкодера (Encoder) та декодера (Decoder), які взаємодіють між собою через механізм уваги. Енкодер призначений для обробки вхідного тексту мовою-джерелом. Він перетворює послідовність токенів у набір контекстних векторів. Кожен шар енкодера містить механізм самоуваги, який дозволяє визначити важливість інших слів у реченні для кожного конкретного слова, а також повнозв’язний нейронний шар для подальшої обробки інформації. Додатково використовується

позиційне кодування, яке забезпечує врахування порядку слів у послідовності.

Декодер виконує генерацію перекладу. Він отримує як вихід енкодера, так і вже згенеровану частину перекладу. У його структурі використовується маскована самоувага, що не дозволяє моделі враховувати майбутні токени під час генерації, а також механізм уваги до виходу енкодера (encoder-decoder attention), який забезпечує узгодженість перекладу з вхідним текстом.

Механізм уваги є ключовим елементом Transformer. Він дозволяє моделі визначати, які слова вхідного речення є найбільш релевантними для генерації кожного слова перекладу. Це забезпечує точніше відтворення змісту та збереження логічних зв’язків між частинами речення.

Загальний процес перекладу включає токенізацію тексту, побудову контекстного представлення в енкодері, послідовну генерацію перекладу в декодері та формування фінального результату. Завдяки такій архітектурі Transformer забезпечує високу якість перекладу, ефективну обробку довгих текстів і можливість використання однієї моделі для різних мовних пар.

### 1.3 Класифікація мовних моделей

Мовні моделі є одним із ключових компонентів сучасних систем NLP, які забезпечують аналіз, генерацію та трансформацію текстової інформації. З розвитком ШІ їх роль суттєво зросла, оскільки вони стали основою для реалізації таких задач, як машинний переклад, автоматична генерація тексту, аналіз документів і створення інтелектуальних асистентів.

Правильний вибір типу моделі дозволяє отримати оптимальний баланс між якістю результатів, швидкістю роботи та обчислювальними ресурсами. Залежно від архітектурних особливостей, обсягу параметрів та сфери застосування, мовні моделі поділяються на кілька основних класів (табл. 1.1). Найбільш поширеними серед них є LLM, малі мовні моделі (Small Language

Models, SLM), а також спеціалізовані та доменно-орієнтовані рішення. LLM являють собою нейронні мережі з великою кількістю параметрів, яка може досягати десятків або навіть сотень мільярдів.

Таблиця 1.1 – Порівняння популярних мовних моделей

| Модель | GPT (OpenAI) | LLaMA (Meta) | Mistral | MamayLM | Gemma (Google) |
|---|---|---|---|---|---|
| Тип | LLM | LLM / SLM | LLM | SLM | SLM / LLM |
| Кількість параметрів | ~10B – 1T+ (залежно від версії) | 7B, 13B, 34B, 70B | 7B, Mixtral (8×7B) | ~1B – 7B (залежить від версії) | 2B, 7B |
| Архітектура | Decoder-only (Transformer) | Decoder-only | Decoder-only + MoE | Transformer | Decoder-only |
| Контекст (токени) | до 128k+ | до 4k–32k | до 8k–32k | ~4k–8k | до 8k |
| Можливість локального запуску | Ні (закрита модель) | Так | Так | Так | Так |
| Основні переваги | дуже висока якість, універ-сальність | відкритість, добре підходить для локальних задач | висока ефектив-ність, швидкість | оптимізова на для UA/EN, офлайн робота | легка, оптимі-зована для локального запуску |
| Основні недоліки | потребує API, немає локального доступу | потребує оптимізації (кванти-зація) | складніша архітектура (MoE) | менша універсаль ність | слабша за великі LLM |

Такі моделі навчаються на значних обсягах текстових даних і, як правило, базуються на архітектурі Transformer. Основною їх перевагою є здатність до генерації зв’язного тексту, глибоке розуміння контексту та універсальність, що дозволяє використовувати одну модель для широкого спектра задач. Зокрема, LLM застосовуються для машинного перекладу, створення чат-ботів, генерації тексту, програмування та аналізу великих масивів документів.

Разом із тим, використання LLM супроводжується низкою обмежень. Насамперед це значні вимоги до обчислювальних ресурсів і пам’яті, що ускладнює їх локальне застосування. Ці моделі можуть генерувати некоректні або неточні відповіді, що відомо як ефект «галюцинацій». Незважаючи на це,

LLM залишаються найбільш потужним інструментом для складних задач обробки тексту.

На відміну від LLM, SLM орієнтовані на ефективну роботу в умовах обмежених ресурсів. Вони мають значно меншу кількість параметрів (від мільйонів до кількох мільярдів), що дозволяє запускати їх на локальних пристроях, включаючи вбудовані системи, такі як Raspberry Pi. Основними перевагами SLM є висока швидкість, знижені вимоги до апаратного забезпечення та можливість офлайн-використання. Це робить їх особливо актуальними для створення локальних перекладачів, мобільних застосунків та Edge-рішень.

Однак компактність SLM обмежує їх здатність до глибокого розуміння складного контексту, що може негативно впливати на якість результатів у порівнянні з великими моделями. Таким чином, вибір між LLM та SLM визначається компромісом між якістю, швидкістю та доступними ресурсами.

Окрім поділу за розміром, мовні моделі класифікуються також за архітектурою. Зокрема, виділяють encoder-only моделі, decoder-only моделі та encoder-decoder (seq2seq) моделі. Encoder-only моделі призначені переважно для аналізу тексту, наприклад класифікації або визначення тональності, і не виконують генерацію тексту. Decoder-only моделі, навпаки, орієнтовані на генерацію та працюють у режимі послідовного створення тексту (token-by-token). Encoder-decoder моделі поєднують обидва підходи і широко використовуються для задач машинного перекладу, сумаризації та перефразування тексту.

Окрему групу становлять доменно-орієнтовані моделі, які навчаються на спеціалізованих даних у конкретних галузях, таких як медицина, юриспруденція, радіотехніка або програмування. Їх перевагою є більш точне використання термінології, що особливо важливо для технічних і наукових текстів. Також важливим напрямом розвитку є інструкційно-налаштовані моделі, які оптимізовані для виконання конкретних запитів користувача, а також мультимодальні моделі, здатні працювати не лише з текстом, а й із

зображеннями, аудіо та відео. Сучасні тенденції розвитку мовних моделей спрямовані на підвищення ефективності та доступності цих технологій. Зокрема, активно досліджуються методи зменшення розміру моделей без втрати якості, розвиток локальних рішень для роботи офлайн, інтеграція з базами знань (RAG-підхід), а також підвищення точності результатів і зменшення кількості помилок. Таким чином, мовні моделі є фундаментальною складовою ІС. Різноманіття їх типів дозволяє обирати оптимальне рішення залежно від конкретної задачі, забезпечуючи баланс між якістю обробки тексту, швидкістю роботи та вимогами до обчислювальних ресурсів.

## 1.4 Методи підвищення ефективності мовних моделей

У сучасних дослідженнях у галузі ML особлива увага приділяється методам підвищення ефективності та адаптивності LLM. До найбільш поширених підходів належать дистиляція знань (Knowledge Distillation) [7], донавчання (Fine-tuning) [8] та підхід із розширенням генерації за рахунок зовнішніх джерел (Retrieval-Augmented Generation, RAG) [9]. Fine-tuning передбачає додаткове навчання попередньо натренованої моделі на спеціалізованих або доменних даних. Основною метою цього підходу є адаптація моделі до конкретної предметної області або задачі, наприклад, автоматичного перекладу, аналізу технічних текстів або роботи з вузькоспеціалізованими термінами. На відміну від дистиляції, при донавчанні не змінюється архітектура моделі, а відбувається уточнення її вагових коефіцієнтів відповідно до нових даних [10]. Дистиляція знань (рис. 1.2) є методом навчання, при якому знання передаються від великої, обчислювально складної моделі (teacher-моделі) до меншої за розміром моделі (student-моделі). У процесі дистиляції компактна модель навчається не лише на вихідних навчальних даних, але й на відповідях, згенерованих більш

потужною моделлю, що дозволяє відтворювати її поведінку з мінімальними втратами якості. Застосування даного підходу, зокрема у мовних моделях середнього розміру (наприклад, із кількістю параметрів близько 9 млрд), забезпечує суттєве зменшення вимог до обчислювальних ресурсів при збереженні високої точності обробки природної мови [11].

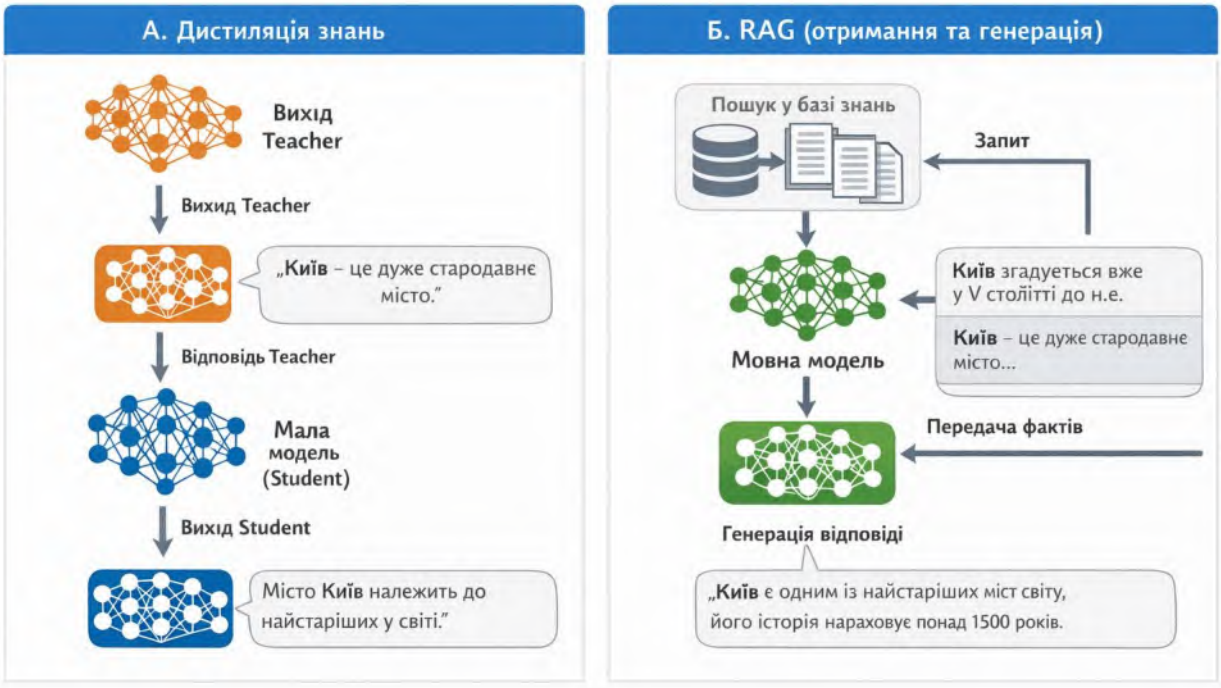


Рисунок 1.2 – Методи навчання мовних моделей

У сучасних ІС, що базуються на LLM, однією з ключових проблем є обмеженість внутрішніх знань моделі та можливість генерації недостовірної інформації. Для подолання цих обмежень активно застосовується підхід RAG, який поєднує механізми пошуку інформації та генерації тексту [12]. Технологія RAG являє собою гібридну архітектуру, у якій мовна модель доповнюється зовнішнім джерелом знань, таким як база документів, корпоративна база даних або спеціалізоване сховище текстів. Основна ідея полягає в тому, що перед генерацією відповіді модель отримує релевантну інформацію із зовнішнього джерела, яка використовується як контекст для формування результату. Процес роботи RAG можна описати як послідовність кількох етапів (рис. 1.3). На першому етапі виконується формування запиту

користувача. Далі цей запит перетворюється у векторне представлення за допомогою embedding-моделі. Отриманий вектор використовується для пошуку найбільш релевантних фрагментів тексту у векторній базі даних. Після цього знайдені документи або їх частини передаються разом із початковим запитом у мовну модель, яка генерує відповідь з урахуванням отриманого контексту.

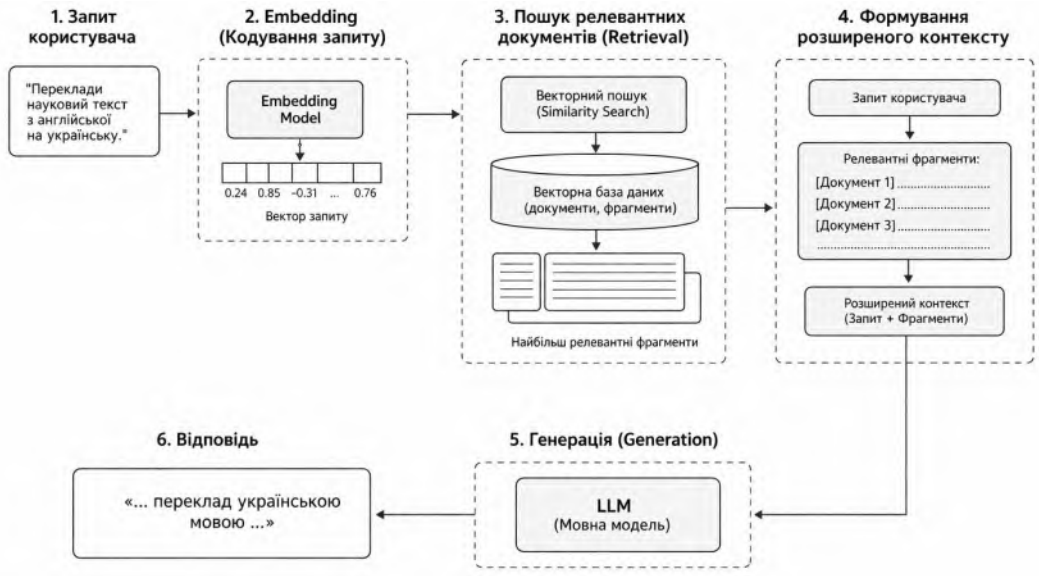


Рисунок 1.3 – Етапи роботи RAG

Тобто, на відміну від класичних LLM, які покладаються лише на знання, отримані під час навчання, RAG дозволяє використовувати актуальну, специфічну та контрольовану інформацію. Це особливо важливо для задач, пов’язаних з технічною документацією, науковими дослідженнями або корпоративними знаннями.

Однією з головних переваг RAG є підвищення достовірності відповідей. Оскільки модель опирається на реальні документи, зменшується ймовірність виникнення «галюцинацій». Крім того, RAG забезпечує актуальність інформації, адже база знань може регулярно оновлюватися без необхідності повторного навчання моделі. Ще перевагою є адаптація системи до конкретної предметної області шляхом додавання спеціалізованих документів.

Важливим аспектом є також ефективність використання ресурсів. Замість навчання великої моделі на вузькоспеціалізованих даних, можна використовувати вже готову LLM і доповнювати її необхідною інформацією через механізм пошуку. Це значно спрощує впровадження системи та знижує витрати на її розробку.

Незважаючи на значні переваги, технологія RAG має низку обмежень, що треба враховувати при проєктуванні ІС на її основі. Перш за все, якість роботи RAG-системи безпосередньо залежить від ефективності механізму пошуку. Якщо на етапі retrieval не будуть знайдені релевантні документи або знайдені лише частково релевантні фрагменти, мовна модель отримає некоректний або недостатній контекст. У такому випадку навіть потужна LLM не зможе забезпечити якісний результат, оскільки генерація відповіді ґрунтується на отриманих даних. Другим суттєвим недоліком є залежність від якості та повноти бази знань. Якщо база містить застарілу, суперечливу або неповну інформацію, це безпосередньо вплине на кінцевий результат. Крім того, створення та підтримка такої бази знань потребує додаткових ресурсів, включаючи підготовку даних, їх індексацію та регулярне оновлення. Ще одним важливим аспектом є збільшення складності системи. На відміну від класичних мовних моделей, RAG передбачає інтеграцію кількох компонентів: embedding-моделі, векторної бази даних, механізму пошуку та генеративної моделі. Це ускладнює розробку, налаштування та підтримку системи, а також підвищує вимоги до її архітектури.

Додатковою проблемою є затримка під час формування відповіді. Оскільки перед генерацією необхідно виконати пошук у базі даних, загальний час обробки запиту зростає. Це може бути критичним для систем реального часу або інтерактивних застосунків. Також слід враховувати обмеження контекстного вікна мовної моделі. Кількість інформації, яку можна передати до LLM, є обмеженою, тому не всі знайдені документи можуть бути використані одночасно. Це вимагає додаткових стратегій відбору та стиснення інформації. Окремою проблемою є так зване

«змішування контексту» (context mixing), коли модель отримує кілька джерел інформації з різними або суперечливими даними. У таких випадках модель може сформувати відповідь, яка частково некоректно поєднує ці джерела.

Крім того, існує ризик перенасичення контексту, коли надмірна кількість додаткової інформації погіршує якість генерації. У таких ситуаціях модель може втрачати фокус на основному запиті або генерувати менш точні відповіді.

Незважаючи на високу ефективність, використання RAG вимагає ретельного проєктування системи, оптимізації процесу пошуку та контролю якості бази знань.

Попри зазначені обмеження, RAG є одним з перспективних напрямів розвитку систем на основі мовних моделей. Його застосування дозволяє поєднати гнучкість генеративних моделей з точністю інформаційно-пошукових систем.

У контексті задач машинного перекладу та локальних LLM, зокрема на пристроях типу Raspberry Pi, технологія RAG може використовуватися для забезпечення узгодженості термінології. Наприклад, система може використовувати локальний словник технічних термінів або попередньо перекладені документи, що дозволяє підвищити якість перекладу без збільшення розміру моделі.

RAG є ефективним інструментом розширення можливостей мовних моделей, який забезпечує підвищення точності, актуальності та керованості результатів генерації тексту. Його використання є доцільним у системах, де критичною є достовірність інформації та можливість роботи з доменно-специфічними даними. Технологія RAG може реалізовуватися у різних варіантах залежно від архітектури системи, способу пошуку інформації та рівня інтеграції з мовною моделлю. Розглянемо основні підходи, які застосовуються на практиці. Одним із базових варіантів є класичний (naive) RAG. У цьому підході після отримання запиту виконується пошук релевантних документів у векторній базі, і знайдені фрагменти

безпосередньо додаються до промпта мовної моделі. Такий підхід є простим у реалізації, проте має обмежену ефективність у складних задачах. Більш досконалим варіантом є RAG із повторним ранжуванням (Re-ranking) [13]. У цьому випадку після первинного пошуку отримується набір кандидатів, які додатково оцінюються за допомогою спеціальної моделі або алгоритму. Це дозволяє відібрати найбільш релевантні документи та покращити якість контексту. Іншим підходом є багатокроковий (Multi-step) RAG, у якому пошук виконується ітеративно [14]. На кожному кроці уточнюється запит або доповнюється контекст, що дозволяє отримати більш точну інформацію. Такий підхід є ефективним для складних запитів, але потребує більше обчислювальних ресурсів. Окремо виділяють гібридний RAG, який поєднує різні методи пошуку, наприклад, векторний пошук і класичний лексичний (Keyword-based) пошук [15]. Це дозволяє враховувати як семантичну подібність, так і точні збіги ключових слів, що підвищує повноту результатів. Також поширеним є підхід із використанням знань у вигляді графів (Graph RAG) [16]. У цьому випадку інформація зберігається не лише у вигляді текстів, а й у вигляді зв’язків між сутностями. Це дозволяє краще відображати складні залежності між поняттями та підвищує якість відповідей у складних предметних областях. Ще одним варіантом є RAG із використанням пам’яті (Memory-Augmented RAG), де система зберігає попередні взаємодії з користувачем. Це дозволяє забезпечити більш контекстно-залежну поведінку та персоналізацію відповідей. У задачах перекладу особливе значення має так званий Glossary-based RAG, у якому база знань містить термінологічні словники та стандартизовані переклади. Такий підхід дозволяє забезпечити узгодженість термінів у великих документах і є особливо ефективним для технічних текстів. Крім того, застосовується Chunk-aware RAG [17], у якому особлива увага приділяється правильному розбиттю тексту на фрагменти. Це дозволяє зберегти семантичну цілісність інформації та підвищити точність пошуку. Існує широкий спектр технік реалізації RAG. Вибір конкретної з них залежить від

задачі, наявних ресурсів або вимог до якості результату. Для задач машинного перекладу найбільш доцільним є поєднання кількох підходів, зокрема використання глосаріїв, гібридного пошуку та механізмів повторного ранжування.

Таким чином, дистиляція знань спрямована на оптимізацію та зменшення розміру моделей, донавчання – на їх спеціалізацію під конкретні задачі, а RAG – на інтеграцію зовнішніх знань у процес генерації відповідей. Комплексне використання зазначених підходів дозволяє створювати ефективні мовні системи, здатні працювати в умовах обмежених обчислювальних ресурсів та забезпечувати високу якість NLP.

# РОЗДІЛ 2
# РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ЛОКАЛЬНОГО ПЕРЕКЛАДУ

## 2.1 Обґрунтування вибору архітектури трансформерної моделі для автоматизованого перекладу текстів

Вибір конкретної архітектури локальної LLM є важливим етапом під час побудови системи автоматизованого перекладу, оскільки саме базова модель визначає якість розуміння контексту, здатність працювати з багатомовними даними та вимоги до обчислювальних ресурсів. У цьому контексті особливий інтерес становить сімейство моделей Gemma, розроблене Google DeepMind [18] як відкрита лінійка легких сучасних мовних моделей, побудованих на дослідженнях і технологічних підходах, використаних у моделях Gemini [19].

Доцільність розгляду Gemma у межах роботи, присвяченої локальним LLM для автоматизованого перекладу текстів, підтверджується її використанням як базової платформи для україномовних моделей MamayLM [20] та Lapa LLM [21]. Зокрема, MamayLM створювалася на основі Google Gemma 2 9B, а Lapa LLM позиціонується як відкрита велика мовна модель на основі Gemma-3-12B з фокусом на обробку української мови. Таким чином, використання Gemma як основи для українських мовних моделей свідчить про її практичну придатність до адаптації під україномовні задачі, зокрема генерацію, розуміння та переклад текстів. Саме тому подальший розгляд LLM Gemma є логічним продовженням аналізу локальних великих мовних моделей, які можуть застосовуватися для створення автономних систем машинного перекладу.

Моделі Gemma від Google завжди були дуже сильними, але традиційно недооціненими порівняно з більш гучними моделями, такими як серія LLaMA [22]. Однією з відмінних ознак Gemma є великий словник. Це

зроблено для кращої підтримки мультимовності. Крім того, у моделей акцент саме на розмір 27B (на відміну від 8B або 70B). Gemma 2 також доступна у менших конфігураціях: 1B, 4B та 12B. В даному випадку, 27B – це дійсно вдалий компроміс: модель помітно потужніша за 8B, але при цьому не вимагає таких ресурсів, як 70B. Основна властивість полягає у застосуванні архітектури Mixture-of-Experts (MoE) [23] для зниження вимог до пам'яті при інференсі при фіксованому розмірі моделі. Як відомо, MoE – це архітектурний підхід у нейронних мережах, зокрема у LLM, за якого модель складається з кількох окремих підмодулів – «експертів». Під час обробки запиту активуються не всі експерти одночасно, а лише частина з них, яку обирає спеціальний механізм маршрутизації. У контексті LLM це дозволяє збільшити загальну кількість параметрів моделі без пропорційного зростання обчислювальних витрат. Наприклад, модель може мати багато мільярдів параметрів, але для кожного окремого токена використовувати лише кілька «експертів». Завдяки цьому MoE-моделі можуть бути потужними, але відносно ефективними під час виконання.

## 2.2 Дослідження мовних моделей сімейства Gemma 2

В даному контексті, Gemma 2 можна розглядати як ефективну текстову LLM, орієнтовану на практичне використання. Її архітектура поєднує класичну decoder-only Transformer-основу з сучасними оптимізаціями: GQA, RoPE, RMSNorm, GeGLU, logit soft-capping і чергуванням локальної та глобальної уваги. Завдяки цьому Gemma 2 забезпечує добрий баланс між якістю, швидкістю та апаратними вимогами. Для локального запуску найбільш практичними є моделі 2B і 9B, тоді як 27B доцільніше використовувати на потужних GPU або у хмарній інфраструктурі. Надалі більш детально розглянемо її архітектуру. Gemma 2 – це друге покоління відкритих мовних моделей сімейства Gemma, розроблених Google. Моделі

Gemma належать до класу open-weight language models, тобто їхні ваги доступні для дослідників і розробників, що дає змогу запускати, тестувати, донавчати та інтегрувати їх у власні програмні системи. Google позиціонує Gemma 2 як сімейство моделей, орієнтованих на високу продуктивність при відносно помірних апаратних вимогах. Позначення 2B, 9B і 27B є округленими маркетинговими назвами, оскільки фактична кількість параметрів дещо відрізняється через великий словник токенізатора та embedding-шари (табл. 2.1).

Таблиця 2.1 – Основні параметри моделей Gemma 2

| Характеристика | Gemma 2 2B | Gemma 2 9B | Gemma 2 27B |
|---|---|---|---|
| Тип архітектури | Decoder-only Transformer | Decoder-only Transformer | Decoder-only Transformer |
| Розмір прихованого стану d_model | 2304 | 3584 | 4608 |
| Кількість шарів | 26 | 42 | 46 |
| Тип attention | GQA | GQA | GQA |
| Кількість attention heads | 8 | 16 | 32 |
| Кількість KV heads | 4 | 8 | 16 |
| Розмір head | 256 | 256 | 128 |
| Контекст | 8192 токени | 8192 токени | 8192 токени |
| Sliding window | 4096 токенів | 4096 токенів | 4096 токенів |
| Розмір словника | 256 128 токенів | 256 128 токенів | 256 128 токенів |
| Активація | GeGLU | GeGLU | GeGLU |
| Нормалізація | RMSNorm | RMSNorm | RMSNorm |

Ці параметри показують, що зі збільшенням моделі зростають не лише кількість параметрів, а й глибина мережі, розмір прихованого простору, кількість attention heads та feed-forward розмір. Модель Gemma 2 розміром 2B має приблизно 590 млн embedding-параметрів і 2,02 млрд non-embedding-параметрів, модель 9B – приблизно 918 млн embedding-параметрів і 8,32 млрд non-embedding-параметрів, а модель 27B – приблизно 1,18 млрд embedding-параметрів і 26,05 млрд non-embedding-параметрів. За призначенням Gemma 2 є текстовою мовною моделлю, тобто вона працює з текстом на вході та генерує текст на виході. Вона не є мультимодальною моделлю і не призначена безпосередньо для аналізу зображень, аудіо чи

відео. У технічному звіті Google зазначено, що Gemma 2 не є multimodal-моделлю та не навчалася спеціально для досягнення найвищих багатомовних можливостей. Gemma 2 є оптимальною для локального запуску, оскільки вона поєднує декілька властивостей, корисних для практичних застосувань: відносно компактні розміри, оптимізовану Transformer-архітектуру, підтримку instruction-tuned версій і добру сумісність із популярними інструментами запуску. Google вказує, що Gemma 2 підтримується в екосистемах Hugging Face Transformers, JAX, PyTorch, TensorFlow/Keras, vLLM, gemma.cpp, llama.cpp та Ollama. Для локального використання найбільш практичними є моделі Gemma 2 2B і Gemma 2 9B, особливо у квантизованих форматах. Модель 27B має значно вищу якість генерації, але потребує більше оперативної або відеопам’яті. Gemma 2 27B розрахована на ефективний запуск у full precision на одному Google Cloud TPU host, NVIDIA A100 80GB [24] або NVIDIA H100 [25], що робить її потужною, але менш доступною для слабких локальних пристроїв.

Gemma 2 побудована на основі архітектури decoder-only Transformer. Це означає, що модель використовує лише декодерну частину Transformer-архітектури та працює за авторегресійним принципом: кожен наступний токен генерується на основі попередніх токенів контексту. Такий підхід є типовим для сучасних великих мовних моделей, призначених для генерації тексту, діалогових систем, узагальнення, перекладу, пояснення коду та інших NLP-задач. На відміну від класичних encoder-decoder моделей, decoder-only LLM не має окремого енкодера для аналізу вхідної послідовності. Уся обробка виконується всередині послідовності токенів, де кожен токен може враховувати тільки попередні токени або дозволену частину контексту. Саме тому такі моделі добре підходять для генерації продовження тексту, чат-ботів і систем автоматичного завершення фрагментів.

Однією з ключових архітектурних особливостей Gemma 2 є поєднання локальної sliding-window уваги та глобальної уваги. У моделі шари локальної та глобальної уваги чергуються через один шар. Локальні шари мають sliding

window розміром 4096 токенів, тоді як глобальні шари працюють із контекстом до 8192 токенів. Це рішення дозволяє моделі поєднувати два типи обробки контексту. Локальна увага ефективно обробляє близькі фрагменти тексту, що важливо для граматики, локальної семантики та зв'язності речень. Глобальна увага дає змогу враховувати ширший контекст, наприклад попередні абзаци, інструкції користувача або віддалені смислові зв'язки в документі. У класичній глобальній увазі кожен токен взаємодіє з усіма попередніми токенами, що потребує значних обчислювальних ресурсів. Sliding-window attention зменшує навантаження, оскільки кожен токен «бачить» лише обмежене вікно сусідніх токенів. Тому чергування локальної та глобальної уваги є компромісом між якістю розуміння довгого контексту та ефективністю обчислень.

У Gemma 2 використовується різновид attention-механізму, який зменшує кількість ключів і значень у порівнянні з класичною Multi-Head Attention (Grouped-Query Attention, GQA). Хоча Gemma 2 теж використовує GQA, але для підвищення швидкості інференсу при збереженні якості роботи моделі. У стандартній Multi-Head Attention кожна attention-head має власні матриці запитів, ключів і значень. У GQA кілька query-heads можуть спільно використовувати меншу кількість key/value-heads. Це зменшує обсяг обчислень і пам'яті, необхідної для зберігання KV-cache під час генерації тексту. Практично це означає, що Gemma 2 краще підходить для швидкої генерації тексту, ніж модель із повною Multi-Head Attention тієї самої розмірності. Особливо це важливо для локального запуску, де обсяг оперативної пам'яті або відеопам'яті є обмеженим.

Усі основні моделі Gemma 2 мають контекстне вікно 8192 токени. Це означає, що модель може враховувати до 8192 токенів вхідного тексту під час формування відповіді. Hugging Face також описує Gemma 2 як сімейство моделей з чергуванням з локальною увагою на 4096 токенів і глобальною увагою на 8192 токени. Контекст у 8192 токени є достатнім для багатьох стандартних задач: діалогів, коротких документів, фрагментів коду,

інструкцій, невеликих статей або окремих розділів тексту. Однак для великих PDF-документів, дисертацій, великих технічних інструкцій або повноцінного RAG-аналізу великої бази знань цього контексту може бути недостатньо. У таких випадках доцільно використовувати розбиття тексту на фрагменти, retrieval-механізми або моделі з довшим контекстним вікном.

Gemma 2 використовує токенізатор типу SentencePiece з розміром словника 256128 токенів. Він також застосовується у старших моделях (Gemma 1 та Gemini): з поділом цифр, збереженням пробілів і byte-level encoding. Великий словник токенізатора має важливе значення для багатомовної обробки тексту. Хоча Gemma 2 не позиціонується як спеціалізована багатомовна модель, великий словник дає їй змогу працювати з різними мовами краще, ніж моделям з вузьким англомовним словником. Водночас якість роботи українською мовою може поступатися англійській, оскільки початково тренувальні дані були переважно англомовними.

У Transformer-блоці після шару уваги використовується мережа прямого поширення (Feed-Forward Network, FFN), яка окремо обробляє кожне представлення токена та виконує нелінійне перетворення ознак. У Gemma 2 для цього застосовується нелінійність GeGLU. Всі три варіанти Gemma 2 (2B, 9B і 27B) икористовують GeGLU як нелінійну функцію активації. GeGLU є різновидом функції активації з вентильним механізмом (gated activation function). Вона дозволяє моделі гнучкіше керувати проходженням інформації крізь feed-forward блок. У порівнянні з простішими функціями активації, такими як ReLU, gated-активації часто забезпечують кращу якість у LLM, оскільки дають змогу моделі ефективно фільтрувати та комбінувати ознаки.

Для стабілізації навчання Gemma 2 використовує RMSNorm як до, так і після підшарів Transformer-блоку. RMSNorm – це спрощений варіант нормалізації, який масштабує вхідні значення на основі їхнього середньоквадратичного рівня. На відміну від LayerNorm, RMSNorm не центрує дані відносно середнього значення, а лише нормалізує їхню амплітуду. Завдяки цьому зменшуються обчислювальні витрати, зберігається

стабільність навчання та підвищується ефективність роботи LLM. Таким чином, RMSNorm застосовується для нормалізації входу та виходу шару уваги та шару прямого поширення. Це важлива архітектурна деталь, оскільки великі нейронні мережі можуть бути нестабільними під час навчання. Нормалізація допомагає контролювати масштаби активацій, полегшує оптимізацію та зменшує ризик деградації якості під час глибокого навчання. Також, додаткові RMSNorm-компоненти в Gemma 2 покращують стабільність процесу навчання.

Gemma 2 використовує позиційне кодування Rotary Position Embeddings, або RoPE. Це механізм позиційного кодування, який дає моделі змогу враховувати порядок токенів у послідовності. RoPE є одним з елементів, успадкованих від попередніх моделей Gemma. У мовних моделях позиційна інформація є критично важливою. Без неї модель не могла б коректно розрізняти порядок слів у реченні. Наприклад, фрази «модель обробляє текст» і «текст обробляє модель» містять ті самі слова, але мають різний зміст. RoPE дозволяє механізму уваги враховувати відносне розташування токенів, що покращує роботу з довшими послідовностями. Ще однією особливістю Gemma 2 є використання механізму м’якого обмеження логітів (logit soft-capping). Логіти обмежуються в шарах уваги і фінальному шарі моделі, щоб значення логітів залишалися у контрольованому діапазоні. Логіти – це числові значення, які модель формує перед перетворенням у ймовірності токенів. Якщо логіти стають надто великими, навчання або генерація можуть бути менш стабільними. Soft-capping допомагає згладити надмірно великі значення, що позитивно впливає на стабільність моделі.

Gemma 2 навчалася на великому корпусі текстових даних. Наприклад, модель Gemma 2 27B навчалася на 13 трлн токенів, Gemma 2 9B – на 8 трлн токенів, а Gemma 2 2B – на 2 трлн токенів. Джерела даних включали вебдокументи, код і наукові статті. Важливо, що моделі 2B і 9B навчалися з використанням дистиляції знань. Тобто менші моделі навчалися на основі ймовірнісних передбачень більшої моделі-вчителя.

Дистиляція знань дозволяє меншій моделі перейняти частину поведінки більшої моделі. Замість того щоб навчатися лише на «правильному наступному токені», модель-студент навчається наближати розподіл ймовірностей моделі-вчителя. Це може покращити якість малої моделі, оскільки вона отримує не лише жорстку відповідь, а й інформацію про те, які альтернативні токени модель-вчитель вважала ймовірними.

Gemma 2 доступна у двох основних типах: pre-trained і instruction-tuned. Pre-trained модель є базовою мовною моделлю, навченою передбачати наступний токен або наслідувати розподіл моделі-вчителя. Вона добре підходить для подальшого fine-tuning, досліджень або інтеграції в спеціалізовані NLP-системи.

Instruction-tuned версія додатково адаптована до виконання інструкцій користувача. Hugging Face зазначає, що instruction-tuned варіанти Gemma 2 проходили етап післянавчання, який включав кероване тонке налаштування та навчання з підкріпленням.

Для практичного використання в чат-ботах, асистентах, навчальних застосунках, перекладі, поясненні тексту або генерації відповідей, зазвичай, доцільно використовувати саме instruction-tuned версії. Базові pre-trained моделі краще підходять для дослідницьких задач або подальшого спеціалізованого донавчання.

Основною перевагою Gemma 2 є поєднання продуктивності та ефективності (табл. 2.2). Модель Gemma 2 27B демонструє конкурентну якість порівняно з моделями значно більшого розміру, а Gemma 2 9B показує сильні результати у своєму класі.

Попри сильні властивості, Gemma 2 має низку обмежень. По-перше, це текстова, а не мультимодальна модель. Вона не може безпосередньо аналізувати зображення або аудіо без додаткових компонентів. По-друге, контекстне вікно 8192 токени є достатнім для багатьох задач, але може бути обмеженням для великих документів. Наприклад, для аналізу довгих PDF-файлів, дисертацій або великих баз знань потрібно використовувати RAG,

chunking або інші механізми розбиття та пошуку релевантних фрагментів. По-третє, модель навчалася переважно на англомовних даних. Тобто якість роботи з українською може бути нижчою, ніж з англійською, особливо у складних спеціалізованих або стилістично чутливих задачах. По-четверте, як і інші LLM, Gemma 2 може генерувати неточні або вигадані твердження. Це може бути пов’язано з навчальними даними, складністю завдання та формулюванням запиту.

Таблиця 2.2 – Властивості Gemma 2

| Властивість | Сутність |
|---|---|
| Відкрита доступність ваг | Модель можна завантажувати, запускати локально та інтегрувати у власні системи |
| Кілька розмірів | 2B підходить для легших систем, 9B – компроміс якості й ресурсів, 27B – для високої якості |
| GQA | Зменшує витрати пам’яті та прискорює inference |
| Sliding-window + global attention | Поєднує локальну ефективність і ширше контекстне бачення |
| Instruction-tuned версії | Краще підходять для діалогових систем і виконання інструкцій |
| Сумісність з інструментами | Підтримується Hugging Face, Ollama, llama.cpp, Keras, PyTorch, TensorFlow та іншими фреймворками |

Gemma 2 може використовуватися для широкого кола задач NLP. Серед основних напрямів застосування є чат-боти та асистенти, переклад, узагальнення текстів, RAG-системи, програмування, освітні системи, локальні AI-рішення. Для задач перекладу або роботи з документами Gemma 2 доцільно комбінувати з RAG-механізмом, оскільки контекст у 8192 токени не завжди дозволяє передати моделі повний документ. RAG дає змогу спочатку знайти релевантні фрагменти, а потім передати їх у модель для генерації відповіді.

З погляду локального запуску Gemma 2 має різні сценарії використання залежно від розміру моделі. Gemma 2 2B є найкращим варіантом для пристроїв з обмеженими ресурсами. Вона може бути цікавою для запуску на CPU, малопотужних GPU або одноплатних комп’ютерах, хоча якість відповідей буде нижчою, ніж у більших моделей. Gemma 2 9B є

компромісним варіантом. Вона помітно якісніша за 2B, але ще може бути запущена локально на сучасному ПК або ноутбуці за умови квантизації. Локальний запуск Gemma 2 27B потребує значно більше пам’яті. Google вказує, що повноточний запуск 27B орієнтований на потужне апаратне забезпечення, зокрема A100 80GB або H100. Наприклад, для Raspberry Pi 5 практично доцільно розглядати лише квантизовані версії малих моделей, насамперед 2B. Навіть у такому випадку швидкість генерації буде обмеженою. Тому Gemma 2 на Raspberry Pi більше підходить для експериментів, демонстрацій і простих локальних асистентів, ніж для високопродуктивних чат-ботів.

## 2.3 Порівняння архітектури Gemma 2 та Gemma 3

На даний час, Gemma 2 позиціонується як текстова decoder-only Transformer-модель з гібридною увагою: локальна (sliding-window attention) чергується з глобальною увагою у співвідношенні 1:1. Вона орієнтована на ефективну роботу з контекстом до 8192 токенів.

Gemma 3 зберігає загальну основу Gemma 2, але суттєво змінює архітектуру для довгого контексту, мультимодальності та меншого споживання KV-cache. KV-cache – це кеш проміжних векторів Key і Value, які використовуються механізмом уваги під час генерації тексту. Він дає змогу не обчислювати заново інформацію про попередні токени, що прискорює роботу мовної моделі. Водночас зі збільшенням довжини контексту обсяг KV-cache суттєво зростає, тому в Gemma 3 були використані архітектурні зміни, спрямовані на зменшення споживання пам’яті під час роботи з довгими послідовностями. Коли модель генерує текст по одному токену, їй потрібно «пам’ятати», на які попередні слова вона вже звертала увагу. Щоб не перераховувати це щоразу заново, вона зберігає ці дані в KV-cache. У Gemma 3 головна зміна – це співвідношення локальних і глобальних шарів уваги 5:1,

тобто 5 локальних шарів уваги на один глобальний. Це дозволяє підтримувати контекст до 128K токенів у моделях 4B/12B/27B.

Таблиця 2.3 – Порівняння Gemma 2 і Gemma 3

| Ознака | Gemma 2 | Gemma 3 |
|---|---|---|
| Базова архітектура | Decoder-only Transformer | Decoder-only Transformer, спадкоємна щодо Gemma 2 |
| Тип моделі | Текстова LLM | Мультимодальна VLM/LLM: текст + зображення на вході, текст на виході |
| Основні розміри | 2B, 9B, 27B | 1B, 4B, 12B, 27B |
| Контекстне вікно | 8192 токени | 32K для 1B; до 128K для 4B/12B/27B |
| Увага | Чергування локальної та глобальної уваги 1:1 | Інтерлівінг 5 локальних шарів : 1 глобальний шар |
| Sliding window | 4096 токенів | 1024 токени для локальних шарів |
| Глобальна увага | До 8192 токенів | Глобальні шари відповідають за довгий контекст |
| GQA | Так, Grouped-Query Attention | Так, Grouped-Query Attention |
| Нормалізація | RMSNorm: pre-norm і post-norm | RMSNorm: pre-norm і post-norm |
| Активація FFN | GeGLU | Переважно зберігає спадкоємність архітектури Gemma |
| Позиційне кодування | RoPE | RoPE з модифікаціями: для глобальної уваги збільшено base frequency до 1M |
| Soft-capping | Є logit soft-capping | Soft-capping замінено на QK-norm |
| Vision encoder | Відсутній | Є SigLIP-based vision encoder для моделей 4B/12B/27B |
| Основна мета архітектурних змін | Баланс якості й ефективності для текстових задач | Довгий контекст, мультимодальність, менший KV-cache, краща багатомовність |

У Gemma 2 локальна та глобальна увага чергуються через один шар: один локальний шар, один глобальний шар. Локальна увага працює зі sliding window 4096 токенів, а глобальна – з контекстом до 8192 токенів. У Gemma 3 архітектуру змінили так, щоб зменшити витрати пам’яті під час роботи з довгим контекстом. Замість чергування 1:1 використано блоки, де на п’ять локальних шарів уваги припадає один глобальний шар. Sliding window у локальних шарах зменшено до 1024 токенів. Тобто Gemma 2 частіше використовує глобальну увагу, а Gemma 3 робить глобальну увагу рідшою,

але стратегічно важливою. Це зменшує навантаження на пам’ять, особливо при довгих промптах.

Gemma 2 має контекстне вікно 8192 токени. Для багатьох чат-ботів і коротких документів цього достатньо, але для великих файлів PDF, довгих діалогів, RAG-сценаріїв або аналізу великих текстів цього вже мало. Gemma 3 підтримує до 128K токенів для моделей 4B, 12B і 27B, а модель 1B має контекст 32K токени. Для цього в Gemma 3 використано змінену схему уваги, RoPE rescaling і підвищену base frequency RoPE для глобальних шарів уваги. У transformer-моделях під час генерації зберігається KV-cache – кеш ключів і значень механізму уваги. Чим довший контекст, тим більше пам’яті потрібно. Gemma 3 краще пристосована до довгого контексту. Gemma 2 уже була ефективною за класичну повністю глобальну увагу, але її схема 1:1 усе ще створювала значне навантаження при масштабуванні контексту. Gemma 3 спеціально змінює співвідношення локальних і глобальних шарів, щоб KV-cache не зростав настільки різко. В цілому, архітектура Gemma 3 з L:G = 5:1 і sliding window = 1024 зменшує пам’ять KV-cache порівняно з глобальною увагою. Практично це означає, що Gemma 3 краще підходить для локального запуску з довгими документами, бо при великому контексті витрачає пам’ять біль раціонально.

На відміну від Gemma 2 (не є мультимодальною моделлю) Gemma 3 отримала vision-компонент. Для моделей 4B, 12B і 27B використовується vision encoder на основі SigLIP. Зображення перетворюються на послідовність «м’яких» токенів, які потім обробляє мовна модель. Vision encoder працює з роздільною здатністю 896×896, а візуальні embeddings стискаються до фіксованих 256 векторів, що зменшує обчислювальні витрати. Тому Gemma 3 може аналізувати зображення, документи з скриншотами, інтерфейси, графіки, таблиці у вигляді зображень тощо. Gemma 2 використовує logit soft-capping в шарах уваги та фінальному шарі. Це механізм стабілізації, який обмежує надто великі значення логітів. У Gemma 3 цей підхід замінено на нормалізацію query/key у механізмі уваги

(QK-norm). Хоча Gemma 3 зберігає GQA, pre-norm/post-norm і RMSNorm, але замість soft-capping Gemma 2 використовує QK-norm [26]. У спрощеному вигляді: Gemma 2 стабілізує увагу через обмеження логітів, а Gemma 3 – через нормалізацію компонентів уваги.

## 2.4 Реалізація генерації з пошуковим доповненням

Реалізація технології генерації з пошуковим доповненням (RAG) для перекладу документів ґрунтується на поєднанні механізмів пошуку релевантної інформації, формування розширеного контексту та генерації перекладу за допомогою мовної моделі. Такий підхід є особливо доцільним у випадках, коли необхідно забезпечити не лише загальну якість перекладу, а й термінологічну узгодженість, точність передачі змісту та адаптацію до конкретної предметної області. У класичному підході переклад виконується безпосередньо мовною моделлю на основі вхідного тексту. Однак при роботі з великими документами, технічними матеріалами, науковими статтями або інструкціями виникає проблема нестабільності термінів і втрати контексту між окремими частинами тексту. Саме тому використання RAG у перекладі документів дозволяє підвищити якість результату за рахунок звернення до зовнішньої бази знань, у якій зберігаються словники термінів, попередньо перекладені фрагменти, стандартизовані формулювання або спеціалізовані документи. Практична реалізація такої системи починається з підготовки корпусу документів, який буде використовуватися як джерело знань. До цього корпусу можуть входити глосарії технічних термінів, шаблони перекладу, попередні версії перекладених текстів, нормативна документація, наукові статті та інші матеріали, релевантні до конкретної предметної області. Наприклад, для задачі перекладу технічної документації з англійської на українську до бази знань доцільно включити переклади термінів на кшталт «wave port», «radiation boundary», «feed line», «impedance

matching», а також типові конструкції, які часто повторюються, наприклад, в текстах радіотехнічного спрямування. Після формування бази знань виконується етап попередньої обробки документів. Він передбачає очищення тексту, нормалізацію структури, розбиття документів на окремі фрагменти та перетворення цих фрагментів у векторні представлення. Розбиття на фрагменти (chunking), є важливим етапом, оскільки надто великі блоки тексту можуть знижувати точність пошуку, а надто малі – втрачати цілісність змісту. Тому на практиці обирається такий розмір фрагмента, який одночасно забезпечує збереження локального контексту і точний пошук релевантної інформації. Далі для кожного текстового фрагмента обчислюється embedding – числове векторне представлення для відображення семантичного змісту. Ці embedding-вектори зберігаються у векторній БД. У результаті формується індексована колекція знань, до якої система може звертатися під час перекладу. Коли користувач подає документ на переклад, система не передає весь текст безпосередньо до мовної моделі. Спочатку документ проходить етап сегментації на логічні частини. Це можуть бути абзаци, розділи або змістові блоки. Для кожного такого фрагмента формується окремий запит до системи пошуку. Запит також перетворюється у векторне представлення, після чого виконується пошук найбільш релевантних елементів у векторній базі. На цьому етапі система визначає ті документи або фрагменти, які мають найбільшу семантичну подібність до поточного перекладного сегмента. Наприклад, якщо у фрагменті зустрічаються терміни з галузі антен, система може знайти відповідні записи у технічному глосарії або попередніх перекладах. Якщо документ належить до галузі ML, можуть бути знайдені стандартизовані переклади термінів, пов’язаних із нейронними мережами, трансформерами, embeddings або fine-tuning. Після завершення пошуку виконується формування розширеного контексту. Тобто, до вихідного фрагмента тексту додаються знайдені релевантні матеріали: термінологічні відповідники, приклади попередніх перекладів, фрагменти з довідкових документів або короткі пояснення. У такому вигляді мовна

модель отримує не лише сам текст для перекладу, а й допоміжні знання, які спрямовують генерацію відповіді в потрібному напрямі.

Особливу роль у практичній реалізації RAG відіграє побудова правильного промпта для мовної моделі. Промпт повинен містити чітку інструкцію щодо мови перекладу, стилю, вимог до збереження термінології, одиниць вимірювання, форматування та способу використання знайденого контексту. Наприклад, у промпті може бути задано, що модель повинна перекладати текст українською мовою, зберігати технічні терміни відповідно до наданого глосарію, не змінювати числові значення та одиниці вимірювання, а також дотримуватися наукового стилю викладу. Такий підхід суттєво зменшує ймовірність появи термінологічних помилок і підвищує узгодженість перекладу.

Наступним етапом є безпосередня генерація перекладу. Мовна модель аналізує початковий текстовий фрагмент разом із доданим контекстом і формує переклад цільовою мовою. У цьому випадку генерація вже не є повністю автономною: вона спрямовується інформацією, отриманою із зовнішньої бази знань. Саме тому переклад стає більш контрольованим і придатним для спеціалізованих задач.

Після генерації перекладу доцільно виконати етап постобробки. Для великих документів це особливо важливо, оскільки переклад виконується частинами, а отже можуть виникати відмінності у вживанні термінів, скорочень або стилістичних конструкцій між окремими фрагментами. Постобробка може включати перевірку термінологічної узгодженості, нормалізацію форматування, вирівнювання назв, позначень і одиниць вимірювання, а також об’єднання всіх фрагментів у цілісний документ.

При застосуванні RAG для перекладу документів можна виділити кілька ключових переваг. По-перше, система забезпечує значно кращу узгодженість термінології, що є критично важливим для технічних, наукових і нормативних текстів. По-друге, зменшується ризик галюцинацій, оскільки модель опирається на реальні документи та словники, а не лише на внутрішні

статистичні закономірності. По-третє, з’являється можливість адаптувати перекладацьку систему до конкретної галузі без повторного навчання мовної моделі: достатньо оновити або розширити базу знань.

Разом із тим, практична реалізація RAG потребує врахування низки технічних обмежень. Насамперед результат залежить від якості сформованої бази знань. Якщо векторна база містить неповні, застарілі або неякісні документи, це може негативно вплинути на переклад. Крім того, необхідно правильно підібрати параметри розбиття тексту на фрагменти, алгоритми пошуку та кількість релевантних документів, що передаються до моделі. Надмірна кількість контексту може перевантажувати модель, тоді як недостатня – не дасть потрібного ефекту.

Для локального використання, зокрема на пристроях з обмеженими ресурсами, практична реалізація RAG має ще перевагу. Замість запуску LLM можна використовувати компактну локальну модель (SLM), а якість перекладу підвищувати саме за рахунок зовнішньої бази знань. Це робить підхід особливо актуальним для офлайн-систем, локальних перекладачів і edge-пристроїв. У такому випадку навіть відносно невелика мовна модель може демонструвати якісний результат у вузькоспеціалізованій задачі завдяки добре підготовленому контексту. В цілому, реалізація RAG для перекладу документів є ефективним способом поєднання можливостей мовних моделей із контрольованим доступом до зовнішніх знань. Такий підхід дозволяє підвищити точність перекладу, забезпечити стабільність термінології, адаптувати систему до конкретної предметної області та зменшити вимоги до розміру самої моделі. Саме тому технологія RAG є перспективним напрямом у створенні сучасних інтелектуальних систем машинного перекладу.

# РОЗДІЛ 3
# РЕКОМЕНДАЦІЇ ЩОДО ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ АВТОМАТИЗОВАНОГО ЛОКАЛЬНОГО ПЕРЕКЛАДУ

## 3.1 Україномовні моделі для локального перекладу

Для автоматизованого перекладу текстів особливий інтерес становлять LLM, що адаптовані не лише до загальних багатомовних задач, а й до специфіки української мови. Більшість універсальних LLM демонструють високі результати для англійської мови, однак для мов з меншою представленістю у навчальних корпусах якість перекладу, стилістична природність і коректність термінології можуть бути нижчими. Саме тому поява моделей, орієнтованих на українську мову, є важливою для розроблення локальних перекладацьких систем, які можуть працювати без передавання даних на зовнішні сервери.

Однією з таких моделей є MamayLM [20, 27] – відкрита мовна модель, створена з фокусом на українську мову. Перша версія MamayLM була побудована на основі Google Gemma 2 9B і позиціонувалася як ефективна модель для української та англійської мов, здатна працювати на одній графічній карті, що робить її придатною для локального розгортання в умовах обмежених обчислювальних ресурсів. Важливою особливістю MamayLM є її практична спрямованість: модель може використовуватися як основа для прикладних систем, зокрема чат-ботів, інтелектуальних помічників, систем аналізу документів і автоматизованого перекладу (рис. А.1). Це має особливе значення для задач перекладу. Локальний запуск дозволяє обробляти конфіденційні тексти без передавання їх до комерційних API. Подальший розвиток MamayLM відбувся у версії MamayLM v1.0, яка була представлена як мультимодальна українсько-орієнтована LLM (рис. А.2). На відміну від першої версії, MamayLM v1.0 базується на серії моделей Gemma 3 і доступна у варіантах на 4 та 12 млрд параметрів. Розробники

зазначають, що нова версія отримала розширений контекст, кращі мовні можливості та підтримку візуальних вхідних даних. Для завдань автоматизованого перекладу особливо важливим є збільшений контекст, оскільки переклад великих документів потребує збереження логіки викладу, узгодженості термінів і врахування попередніх фрагментів тексту. Це дає змогу використовувати MamayLM не лише для перекладу окремих речень, а й для роботи з більшими текстовими блоками, наприклад розділами документів, інструкціями або технічною документацією.

Іншим важливим прикладом є Lapa LLM [21] – українська відкрита мовна модель, створена на основі Gemma-3-12B. Метою Lapa LLM є створення відкритої моделі для опрацювання української мови разом із наборами даних для попереднього навчання та інструкційного донавчання. Такий підхід є важливим для академічних і прикладних досліджень, оскільки відкритість моделі, навчальних матеріалів і документації спрощує перевірку результатів, повторне використання та адаптацію під конкретні задачі. Однією з ключових технічних особливостей Lapa LLM є адаптація токенізатора до української мови. За даними розробників, у моделі було замінено 80 тис. токенів з 250 тис. на українські, що дало змогу ефективніше обробляти українськомовний текст (рис. А.3). Для систем машинного перекладу це має безпосереднє значення: чим природніше модель розбиває українські слова, словоформи та сталі конструкції на токени, тим менше обчислень потрібно для аналізу тексту та тим вищою може бути якість генерації перекладу. Особливо це важливо для української мови, яка має розвинену систему відмінювання, велику кількість словоформ і складну морфологічну структуру. Lapa LLM демонструє прикладну цінність завдяки спеціалізації на українській мові. У відкритих матеріалах зазначено, що версія Lapa LLM v0.1.2 показала результат 33 BLEU на тесті FLORES для перекладу з англійської на українську, а також має сильні результати в узагальненні текстів, відповідях на запитання та роботі з документами. Це робить модель перспективною не лише для прямого машинного перекладу, а

й для складніших сценаріїв, наприклад, перекладу з попереднім пошуком термінів у базі знань, перекладу документів із контекстним уточненням або використання у складі RAG-систем.

В цілому, MamayLM і Lapa LLM можна розглядати як приклади переходу від універсальних багатомовних моделей до спеціалізованих україномовних рішень. Їхня перевага полягає у поєднанні кількох факторів: відкритості, можливості локального запуску, адаптації до української мови та придатності для задач обробки документів. У порівнянні з хмарними API локальні моделі можуть забезпечити кращий контроль над даними, нижчу залежність від зовнішніх сервісів і можливість налаштування під конкретну предметну галузь. Водночас їх використання потребує врахування апаратних вимог, якості квантування, обмежень контекстного вікна та необхідності додаткового тестування на реальних перекладацьких наборах. Таким чином, MamayLM і Lapa LLM є актуальними прикладами українсько-орієнтованих LLM, що можуть бути використані для побудови локальних сервісів NLP. MamayLM демонструє переваги компактної та продуктивної моделі, орієнтованої на українську та англійську мови, тоді як Lapa LLM акцентує увагу на відкритості, оптимізації токенізатора та розвитку українських датасетів. Для практичної реалізації системи перекладу ці моделі можуть застосовуватися як базові генеративні компоненти, що виконують переклад, стилістичне редагування, термінологічне узгодження та адаптацію тексту до вимог конкретної предметної області.

## 3.2 Оцінка локальних програмних засобів для роботи з великими мовними моделями

Локальний програмний засіб для роботи з LLM – це UI-клієнт, через який користувач може запускати LLM на власному комп’ютері та спілкуватися з моделлю. Таких UI-клієнтів досить багато. Функціонал у них

може дуже відрізнятися. У найпростішому вигляді є лише чат. У найбільш просунутих є вбудовані бази знань, робота із зображеннями та багато інших функцій. Тому доцільно проаналізувати їх властивості.

1. Open WebUI [28] – вебклієнт, який розгортається у Docker-середовищі [29]. Він підтримує як запуск локальних моделей з власними вагами, так і підключення до моделей через API, Це є гнучким рішенням для різних сценаріїв використання (рис. 3.1).

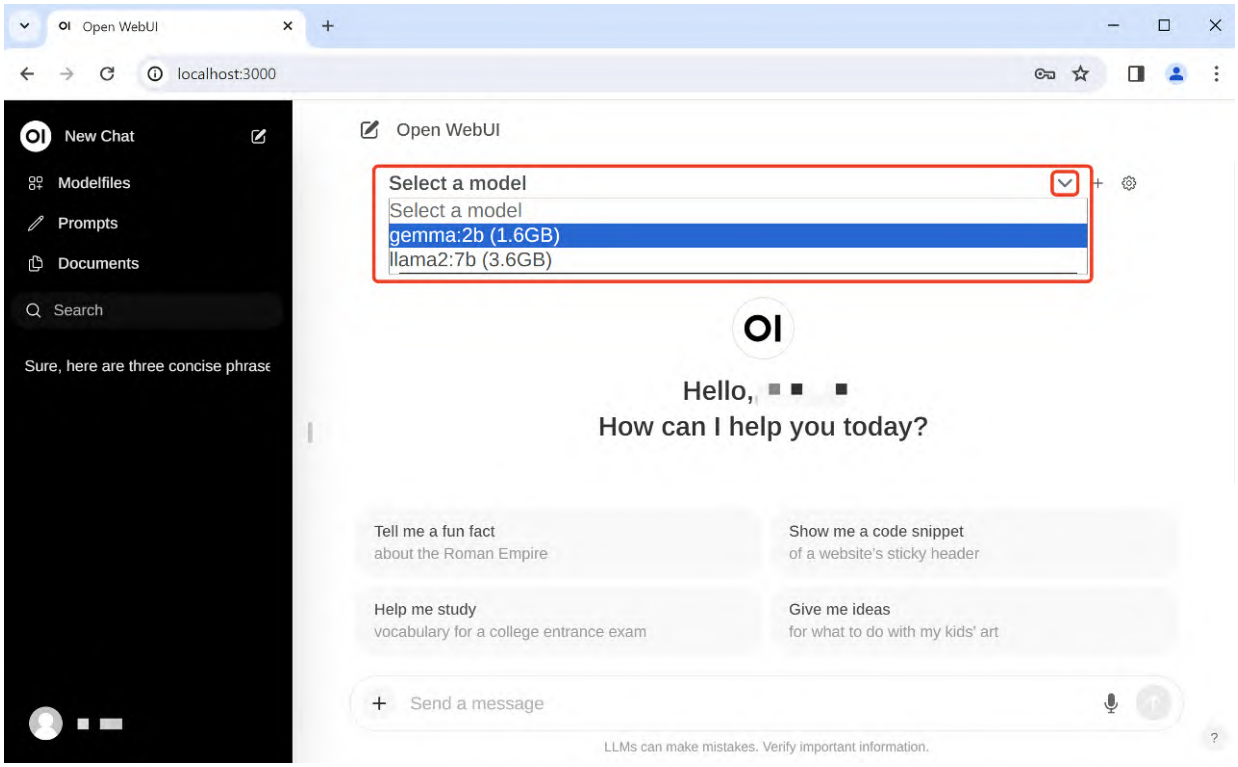


Рисунок 3.1 – Варіант застосування Open WebUI

Однією з головних його переваг є широкий набір функціональних можливостей. Система підтримує роботу зі звуком, зокрема запис голосу та синтез мови за допомогою TTS. Також передбачено роботу з зображеннями, вебпошук, обробку файлів, використання RAG, створення вбудованої бази знань, а також підтримку function/tool calling. Крім того, Open WebUI може працювати в багатокористувацькому режимі, що передбачає створення облікових записів, розподіл ролей та керування доступом користувачів. Open WebUI поширюється за дещо модифікованою ліцензією BSD-3.

2. LM Studio – десктопний клієнт для ОС Windows, Linux і macOS (рис. 3.2) [30]. На відміну від інших рішень, LM Studio передбачає роботу з моделями, які вже завантажені на локальний диск комп'ютера. Водночас програма має вбудований каталог моделей з Hugging Face, тому користувачу не потрібно самостійно шукати та завантажувати моделі вручну.

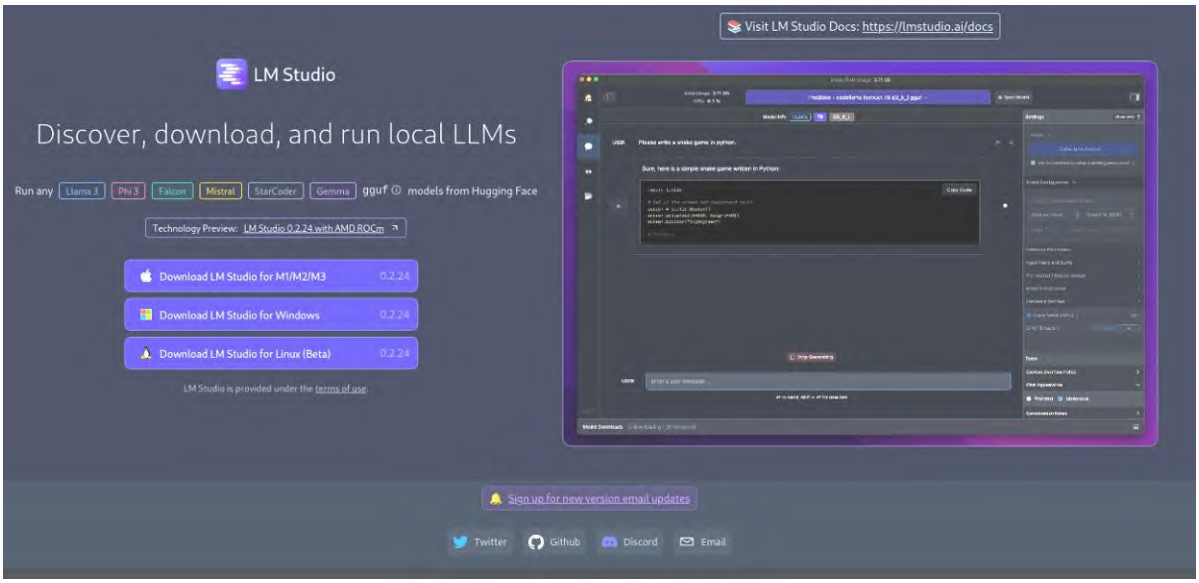


Рисунок 3.2 – LM Studio

Також підтримується використання GPU, що дає змогу суттєво прискорити процес інференсу за наявності відповідного апаратного забезпечення. Функціональні можливості LM Studio є досить широкими. Програма підтримує RAG, створення бази знань, роботу з файлами, мультимодальні моделі, а також налаштування параметрів генерації відповідей. При цьому LM Studio може працювати як локальний API-сервер: інші застосунки, скрипти або сервіси можуть звертатися до запущеної в LM Studio локальної моделі через REST API або OpenAI-compatible API. Таким чином, LM Studio доцільно розглядати не як клієнт для різних зовнішніх API, а як середовище для локального запуску моделей із можливістю надання до них API-доступу. Це дозволяє використовувати локальні моделі в інших застосунках або власних програмних рішеннях. LM Studio має пропрієтарну ліцензію, однак є безкоштовною як для особистого, так і для комерційного

використання. Загалом це зручний та функціонально насичений інструмент для локальної роботи з мовними моделями. Його перевагою є підтримка різних форматів моделей і рушіїв інференсу, що робить програму гнучкою для експериментів та практичного застосування. Водночас LM Studio має суттєве обмеження: програма не призначена для роботи з моделями через зовнішні API, зокрема з хмарними провайдерами або сторонніми сервісами. Тобто її основне призначення – локальне використання на конкретному комп'ютері з попередньо завантаженими моделями. Саме тому LM Studio є дуже вдалим рішенням для індивідуального користувача, дослідника або розробника, але менш універсальним варіантом для сценаріїв, де потрібна інтеграція з хмарними сервісами чи централізована робота через API.

3. Msty Studio – десктопний клієнт для типових ОС (рис. 3.3) [31]. Він підтримує два основні варіанти використання моделей. Перший передбачає роботу з локальними моделями через Ollama [32], що дає змогу запускати мовні моделі безпосередньо на комп'ютері користувача.

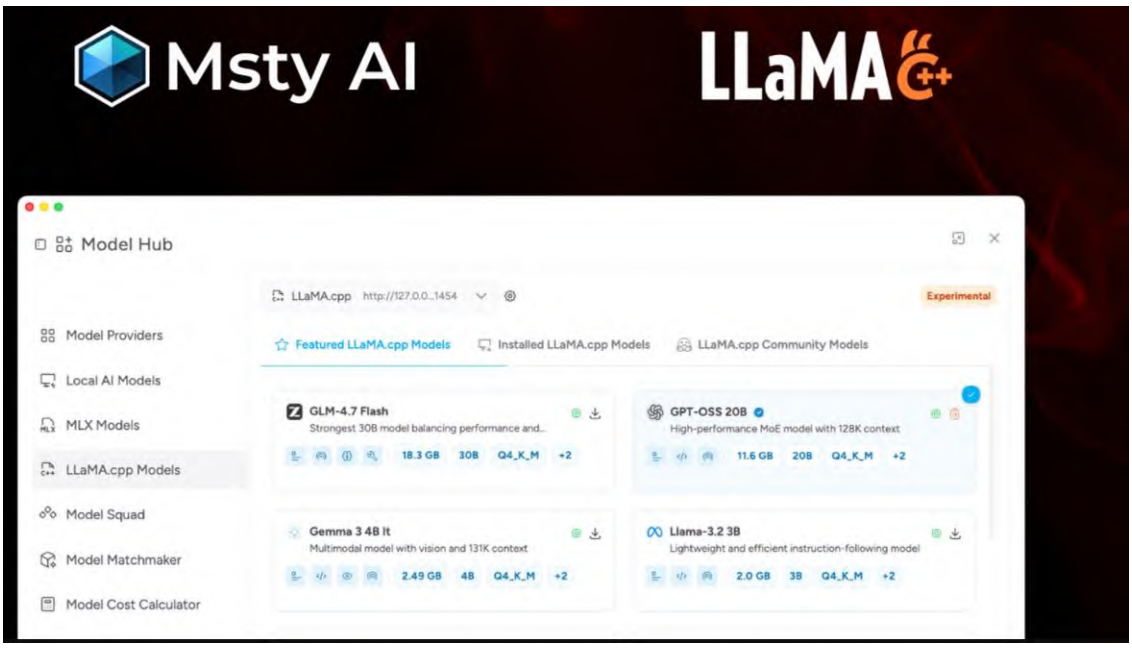


Рисунок 3.3 – Msty Studio

Другий варіант – підключення до моделей через API. Msty має широкий набір провайдерів, що робить його гнучким інструментом для локального

(хмарного) використання. Програма підтримує використання інструментів, RAG, створення бази знань, роботу з файлами та зображеннями, вебпошук та налаштування параметрів генерації відповідей. Тобто Msty може використовуватися не лише як звичайний чат-клієнт, а й як повноцінне середовище для роботи з документами, знаннями та додатковими джерелами інформації. Msty Studio має модель ліцензування Freemium. Базовий функціонал доступний безкоштовно, однак частина можливостей є платною.

AnythingLLM (рис. 3.4) – клієнт для роботи з мовними моделями, який доступний у двох варіантах: як десктопний застосунок для Windows, Linux та macOS, а також як вебклієнт, що може розгортатися через Docker [33]. ПЗ підтримує роботу як з локальними моделями, так й з моделями через API.


Рисунок 3.4 – Інтерфейс AnythingLLM

При цьому система має велику кількість доступних провайдерів, що робить її досить гнучкою для різних сценаріїв використання. Функціональні можливості AnythingLLM є досить широкими. Платформа підтримує RAG, роботу з документами, створення окремих робочих просторів, використання векторної бази даних, інструменти та агентів, мультимодальні моделі, а також синтез мовлення за допомогою TTS. Наявність робочих просторів дає змогу

організовувати документи, бази знань і взаємодію з моделями за окремими темами або проєктами, що є корисним для командної чи дослідницької роботи. ПЗ поширюється за ліцензією MIT, що робить його відкритим і придатним для використання, модифікації та розгортання в різних середовищах. До недоліків можна віднести те, що інтерфейс програми може справляти враження дещо застарілого або менш зручного порівняно з деякими сучасними альтернативами.

Cherry Studio (рис. 3.5) – десктопний клієнт для ОС Windows, Linux та macOS [34]. ПЗ підтримує роботу як з локальними моделями через Ollama, так й з моделями, що підключені через API. Програма має велику кількість підтримуваних провайдерів, що робить її гнучким інструментом для різних сценаріїв використання: від повністю локальної роботи до взаємодії з хмарними сервісами та сторонніми API.

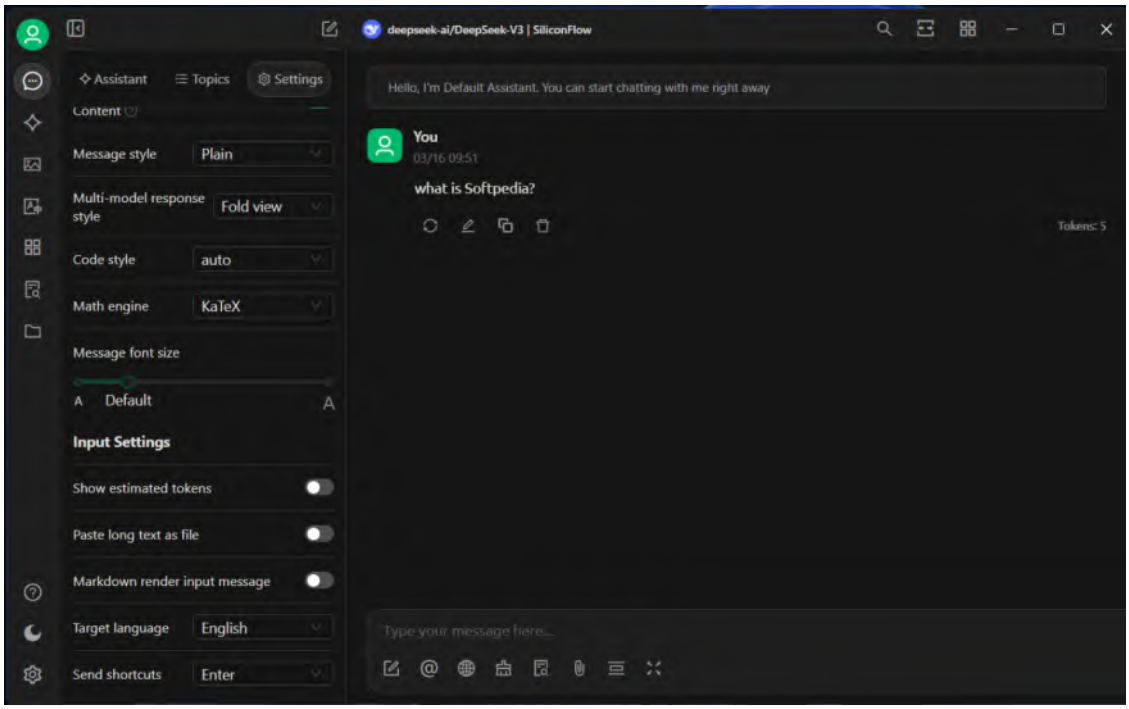


Рисунок 3.5 – Початкові налаштування Cherry Studio

Функціональні можливості Cherry Studio є дуже широкими. Клієнт підтримує RAG, роботу з файлами, створення бази знань, використання інструментів, вебпошук, налаштування параметрів генерації відповідей,

керування режимами reasoning, функції пам’яті, а також мультимодальні можливості, зокрема роботу з зображеннями та аудіо. Завдяки цьому Cherry Studio можна використовувати не лише як звичайний чат-клієнт, а і як повноцінне середовище для роботи з документами, знаннями, різними моделями та додатковими інструментами. Програмне забезпечення поширюється за ліцензією GNU. Однією з помітних переваг Cherry Studio є дуже приємний, сучасний і продуманий інтерфейс, який робить роботу з моделями зручною навіть для користувачів без глибокої технічної підготовки. Крім того, програма має багатий функціонал і підтримує багато корисних можливостей для практичного використання мовних моделей. До незначних недоліків можна віднести те, що в інтерфейсі іноді трапляються китайські ієрогліфи або неповністю перекладені елементи. Це не є критичною проблемою для роботи, однак може створювати певні незручності під час використання програми.

Якщо потрібний веб, то спочатку варто орієнтуватись на Open WebUI, потім Librechat (там дещо складніше налаштування). Якщо потрібен робочий стіл, то лідера два: LM Studio і Msty Studio. LM Studio виглядає потужніше, але не може підключатися через API до зовнішніх провайдерів. Якщо не бентежать дрібні огріхи в інтерфейсі, обов'язково спробуйте Cherry Studio. Крім розглянутих, є й інші рішення, наприклад: private-gpt [35], KoboldCpp [36], SillyTavern [37], Text Generation WebUI [38] та ін.

### 3.3 Формування Pipeline використання MamayLM у LM Studio

Локальне використання LLM передбачає побудову послідовного pipeline, у якому всі основні етапи обробки виконуються на комп’ютері користувача без обов’язкового звернення до хмарних сервісів [39]. В якості середовища запуску моделі розглядається LM Studio, що дає змогу завантажувати, налаштовувати та використовувати локальні LLM через

графічний інтерфейс або локальний API. Для перекладу використовується модель mamaylm-gemma-3-12b-it-v1.0 [40], яка обробляє сформований користувацький запит і генерує переклад на основі заданих інструкцій. Загальна схема локального перекладу складається з кількох етапів (рис. 3.6). З початку користувач подає вхідний текст, який необхідно перекласти. Це може бути окремий фрагмент, абзац, стаття, інструкція або документ, попередньо підготовлений для обробки моделлю. Якість вхідного тексту безпосередньо впливає на результат перекладу, тому перед поданням до моделі бажано усунути технічні помилки, зайві символи, некоректні переноси рядків або фрагменти, які не мають бути перекладені.

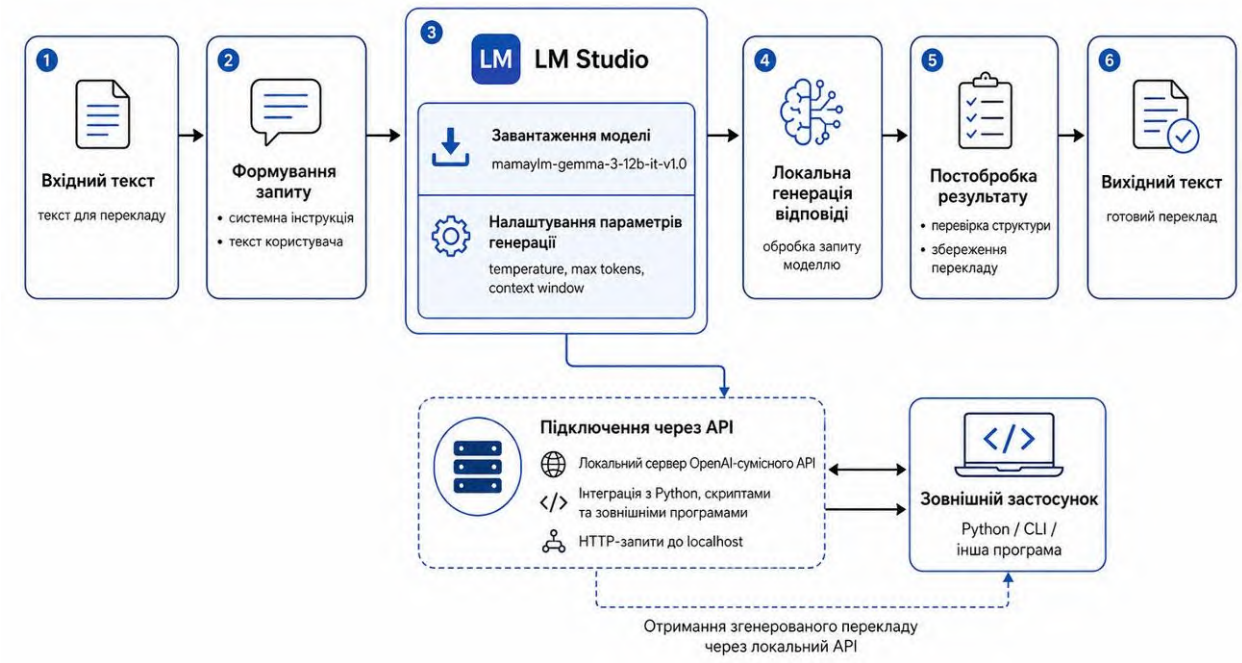


Рисунок 3.6 – Pipeline локального перекладу тексту з використанням LM Studio та моделі mamaylm-gemma-3-12b-it-v1.0

Наступним етапом є формування запиту до моделі. У ньому поєднуються системна інструкція та текст користувача. Системна інструкція визначає роль моделі, мову перекладу, бажаний стиль, вимоги до збереження термінології, структури, форматування та особливостей вихідного документа. Наприклад, модель може отримати інструкцію перекладати текст

українською мовою, зберігати нумерацію, не змінювати назви змінних, технічних параметрів або фрагментів коду. Саме на цьому етапі формується контекст, у межах якого модель буде інтерпретувати поставлене завдання.

Після формування запиту він передається до LM Studio, де попередньо завантажується модель mamaylm-gemma-3-12b-it-v1.0 [40]. У середовищі LM Studio користувач може встановити параметри генерації, зокрема температуру, максимальну кількість токенів, розмір контекстного вікна та інші налаштування, які впливають на стабільність і варіативність відповіді. Для задач автоматизованого перекладу доцільно використовувати помірну або низьку температуру, оскільки переклад потребує не творчої варіативності, а точності, послідовності та збереження змісту оригінального тексту.

Після отримання запиту модель виконує локальну генерацію відповіді. Вона аналізує вхідний текст, враховує системну інструкцію та створює переклад. Оскільки обробка відбувається локально, текст не передається на сторонні сервери. Таке локальне використання також дає змогу краще контролювати параметри генерації та повторюваність результатів.

Після генерації перекладений текст проходить етап постобробки результату. При цьому перевіряється збереження структури документа, коректність абзаців, списків, таблиць, заголовків, позначень, числових значень і спеціальної термінології. Постобробка може виконуватися вручну або частково автоматизовано за допомогою додаткових скриптів. У результаті формується вихідний текст, тобто готовий переклад, який можна зберегти у файл або використати в подальшій роботі.

Окремою можливістю LM Studio є підключення до моделі через локальний OpenAI-сумісний API. У такому випадку LM Studio працює як локальний сервер, до якого можуть звертатися зовнішні застосунки, Python-скрипти, CLI-інструменти та ін. Це дозволяє інтегрувати модель у власне ПЗ, автоматизувати переклад великих обсягів тексту та організувати пакетну обробку документів. Наприклад, зовнішній застосунок може надсилати HTTP-запити до локального сервера LM Studio, отримувати згенерований

переклад і зберігати результат у потрібному форматі. Це робить pipeline більш гнучким і придатним як для ручного перекладу окремих фрагментів, так і для створення системи автоматизованого перекладу. Базовий pipeline локального перекладу може бути розширений за допомогою RAG (рис. 3.7). У такому випадку модель отримує не лише текст для перекладу та системну інструкцію, а й додатковий релевантний контекст із підготовленої бази знань. Такий підхід є особливо корисним для перекладу спеціалізованих текстів, у яких важливо зберігати сталі терміни, назви понять, галузеву лексику або внутрішні правила перекладу. У такому pipeline спочатку формується база знань, до якої можуть входити документи, статті, глосарії, інструкції, приклади перекладів або термінологічні словники. Далі ці матеріали проходять етап індексації: документи розбиваються на менші фрагменти, для них створюються embedding-представлення, після чого вони зберігаються у векторному сховищі.

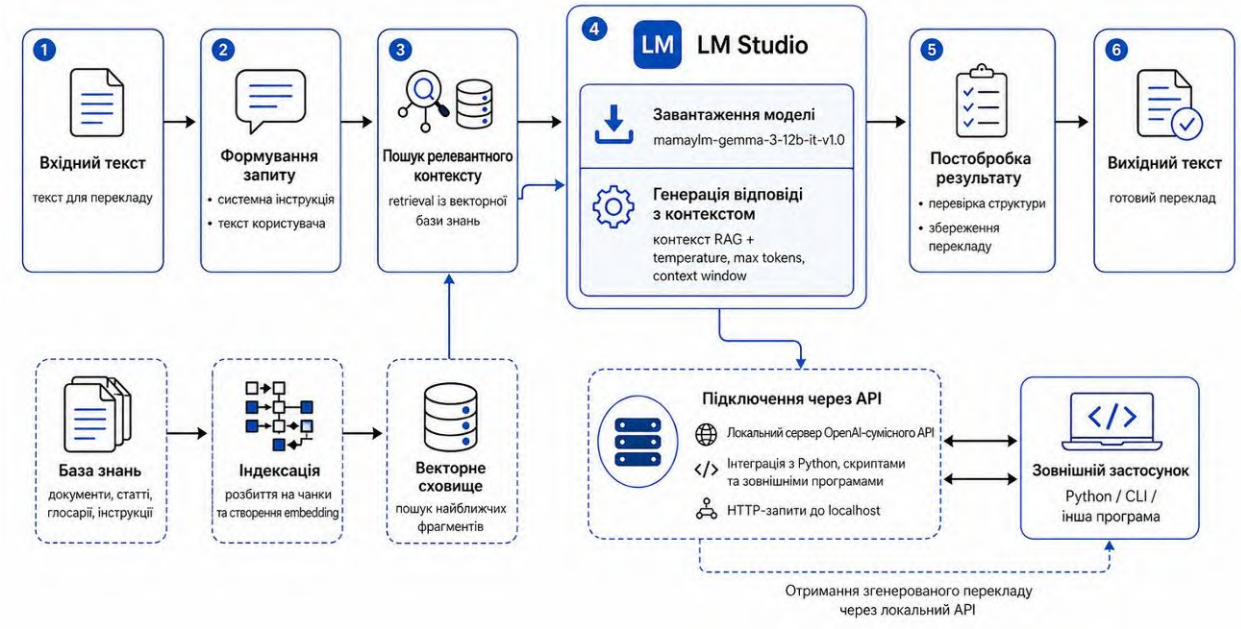


Рисунок 3.7 – Pipeline локального перекладу тексту з використанням RAG, LM Studio та моделі mamaylm-gemma-3-12b-it-v1.0

Векторне сховище дає змогу виконувати пошук найближчих за змістом фрагментів до поточного запиту користувача. Коли користувач надсилає текст

для перекладу, система спочатку виконує пошук релевантного контексту у векторній базі знань. Наприклад, якщо в тексті зустрічається спеціальний технічний термін, система може знайти відповідний фрагмент із глосарія або попереднього документа, де вже визначено бажаний варіант перекладу. Після цього знайдений контекст додається до основного запиту і передається в LM Studio разом із текстом користувача. Таким чином, модель генерує переклад не ізольовано, а з урахуванням додаткових довідкових матеріалів. Використання RAG підвищує якість перекладу в тих випадках, коли звичайних знань моделі недостатньо або коли необхідно дотримуватися конкретної термінології. Для бакалаврської роботи це є важливим аспектом, оскільки автоматизований переклад часто потребує не лише граматично правильного результату, а й відповідності предметній області. Наприклад, один і той самий термін у загальному тексті та в технічній документації може перекладатися по-різному. Додавання релевантного контексту дає змогу зменшити кількість таких помилок і забезпечити більшу узгодженість перекладу в межах усього документа.

Отже, схема локального використання mamaylm-gemma-3-12b-it-v1.0 у LM Studio демонструє два основні варіанти побудови системи автоматизованого перекладу: базовий pipeline без зовнішнього контексту та розширений pipeline із використанням RAG. У першому випадку модель перекладає текст на основі сформованої інструкції та власних мовних можливостей. У другому випадку до процесу додається база знань, векторне сховище та механізм пошуку релевантних фрагментів, що дозволяє підвищити точність, термінологічну сталість і контрольованість перекладу. Завдяки підтримці локального API система може інтегруватись у зовнішні застосунки, що відкриває можливість створення повністю локального програмного комплексу для автоматизованого перекладу текстів.

## 3.4 Практичне підтвердження працездатності системи автоматизованого локального перекладу

Для практичного підтвердження можливості локального використання мультимодальної MamayLM в задачах автоматизованого опрацювання та перекладу текстів було проведено тестування у LM Studio моделі mamaylm-gemma-3-12b-it-v1.0. Основною метою цього етапу було перевірити, чи здатна локально розгорнута мовна модель без звернення до зовнішніх хмарних сервісів коректно сприймати інструкції українською мовою, формувати зв’язні відповіді, працювати з різними типами вхідних даних і зберігати змістову цілісність під час опрацювання тексту.

Тестування виконувалося у LM Studio 0.3.30. У верхній частині інтерфейсу було обрано модель mamaylm-gemma-3-12b-it-v1.0, після чого користувач вводив інструкції українською мовою. Важливою особливістю такого підходу є те, що обробка запитів виконується локально. Це має значення для задач перекладу документів, оскільки текстові дані не передаються до сторонніх онлайн-сервісів, що підвищує рівень конфіденційності та дає змогу використовувати систему навіть у випадках, коли доступ до мережі Інтернет є обмеженим або небажаним.

На першому етапі було перевірено здатність моделі розуміти просту інструкцію українською мовою та формувати природний опис вхідного зображення (рис. 3.8). Модель отримала запит «Опиши зображення» і сформувала відповідь українською мовою, у якій коректно визначила основні об’єкти сцени: білого ведмедя, крижину, арктичний пейзаж, пляшку газованого напою та загальний позитивний настрій ілюстрації. Отриманий результат демонструє, що локально запущена модель здатна не лише генерувати текст українською мовою, а й підтримувати логічну структуру опису: спочатку називається головний об’єкт, далі уточнюються деталі сцени, а наприкінці подається узагальнена характеристика зображення. Цей результат є важливим для задач автоматизованого перекладу, оскільки

переклад не обмежується механічною заміною слів однієї мови словами іншої. Якісний переклад передбачає розуміння контексту, визначення головних і другорядних смислових елементів, а також побудову граматично правильного й стилістично узгодженого тексту цільовою мовою. Здатність моделі формувати зв’язний україномовний опис підтверджує її придатність до роботи з інструкціями природною мовою та до генерації тексту, який може бути використаний у подальших перекладацьких сценаріях.

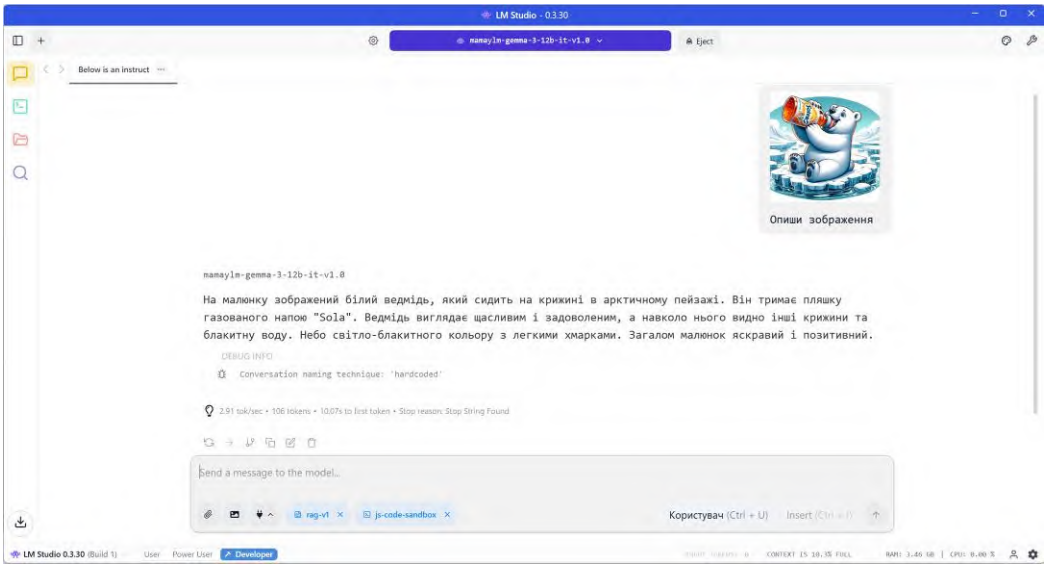


Рисунок 3.8 – Генерація україномовного опису зображення локальною моделлю mamaylm-gemma-3-12b-it-v1.0

На другому етапі було перевірено роботу моделі зі складнішим текстовим матеріалом, що містить науково-технічну термінологію, математичні формули, позначення та структуровані смислові блоки. У відповідь на запит модель сформувала опис схеми, пов’язаної з динамікою відкритих систем (рис. 3.9). У відповіді були згадані такі поняття, як рівняння Ліндблада, гамільтоніан, дисипатор, векторний простір, embedding-compression, метод Nyström, SVD, матриця спостережень, оцінка похибки, RMSE та інтегральна похибка. Попри наявність спеціалізованих термінів і формул, модель змогла подати матеріал у вигляді послідовного пояснення з поділом на ключові аспекти.

Проведений експеримент показує, що mamaylm-gemma-3-12b-it-v1.0 у локальному середовищі здатна працювати з подібним типом інформації та формувати осмислену відповідь українською мовою.

Практичне тестування також підтвердило, що модель може використовуватись для побудови локального інструменту автоматизованого перекладу. У такій системі користувач може передавати до моделі фрагменти тексту, абзаци, технічні пояснення або документи, після чого модель генерує переклад або адаптований варіант тексту цільовою мовою. При цьому LM Studio може виконувати роль середовища для локального запуску моделі, а також для перевірки окремих запитів, налаштування параметрів генерації та оцінювання якості відповідей.

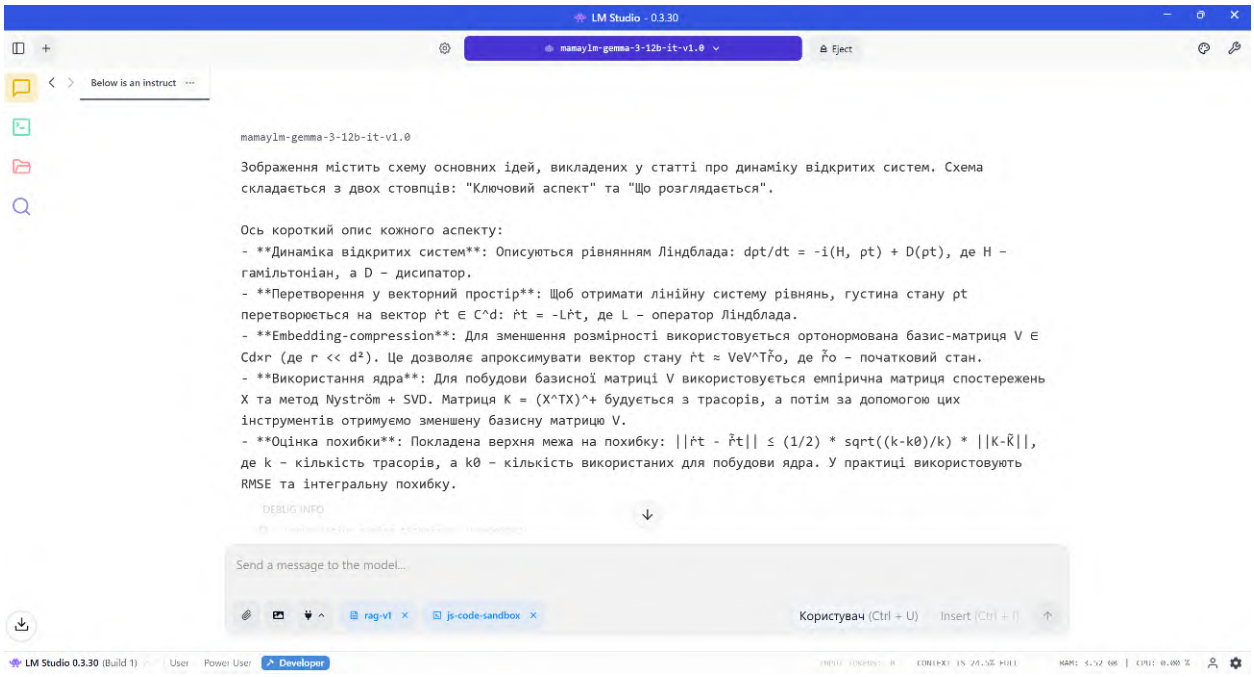


Рисунок 3.9 – Обробка науково-технічного тексту локальною моделлю mamaylm-gemma-3-12b-it-v1.0 у LM Studio

Окремою перевагою локального підходу є можливість контролювати процес роботи моделі. На відміну від хмарних сервісів, де користувач передає текст на зовнішній сервер, локальна модель працює в межах власного пристрою. Це особливо важливо під час перекладу документів, що можуть

містити персональні дані, службову інформацію, навчальні матеріали або внутрішню документацію організації. Крім того, локальне розгортання дає змогу експериментувати з різними промптами, форматами запитів і параметрами генерації без додаткових витрат за кожен запит до API. Разом з тим, результати тестування показують, що якість роботи локальної мовної моделі залежить від характеру вхідного запиту. Для отримання кращого результату доцільно формулювати інструкцію чітко: вказувати мову перекладу, бажаний стиль, необхідність збереження термінів, формат вихідного тексту та обмеження щодо зміни структури документа. Для перекладу технічного тексту доцільно додавати уточнення про збереження формул, одиниць вимірювання, назв методів і спеціалізованих термінів. Це дозволяє зменшити ризик неточностей і підвищити стабільність результату.

### 3.5 Техніко-економічне обґрунтування прийнятих рішень

Для кількісного підтвердження економічної доцільності використання локальної мовної моделі виконаний орієнтовний розрахунок витрат і потенційної економії часу під час автоматизованого перекладу текстів. Розрахунок є наближеним, оскільки фактичні показники можуть залежати від продуктивності комп’ютера, складності тексту, обсягу документа, швидкості генерації відповіді моделлю та часу, необхідного користувачу для редагування перекладу. Для прикладу приймемо, що протягом місяця необхідно перекласти $V = 100$ сторінок тексту. За умов ручного перекладу однієї сторінки середньої складності користувач витрачає приблизно $t_1 = 20$ хв. У разі використання локальної LLM значна частина роботи виконується автоматично, а користувач витрачає час переважно на перевірку, редагування та виправлення неточностей. Приймемо, що в такому випадку на одну сторінку потрібно приблизно $t_2 = 7$ хв. Тоді загальні витрати часу для ручного перекладу становитимуть:

$$T_{ручн} = \frac{V \cdot t_1}{60} = 33{,}3 \text{ год.} \quad (3.1)$$

Витрати часу під час використання локальної мовної моделі:

$$T_{LLM} = \frac{V \cdot t_2}{60} = 11{,}7 \text{ год.} \quad (3.2)$$

Як наслідок, економія часу становитиме:

$$\Delta T = T_{ручн} - T_{LLM} = 21{,}6 \text{ год.} \quad (3.3)$$

Отже, використання локальної мовної моделі дає змогу скоротити час виконання перекладацької роботи приблизно на:

$$E_T = \frac{\Delta T}{T_{ручн}} \cdot 100\% = 64{,}9\%. \quad (3.4)$$

Таким чином, за наведених умов автоматизований переклад із використанням локальної LLM дозволяє зменшити витрати часу майже на 65 % порівняно з повністю ручним перекладом. Для оцінки економічного ефекту приймемо умовну вартість однієї години роботи користувача або перекладача на рівні $S = 100$ грн/год. Тоді вартість ручного перекладу становитиме:

$$C_{ручн} = T_{ручн} \cdot S = 3330 \text{ грн.} \quad (3.5)$$

Вартість роботи користувача під час використання локальної LLM:

$$C_{ред} = T_{LLM} \cdot S = 1170 \text{ грн.} \quad (3.6)$$

Оскільки локальна модель працює на комп'ютері користувача, необхідно також врахувати витрати на електроенергію. Приймемо середню потужність комп'ютера під час роботи моделі $P = 0{,}15$ кВт, тривалість роботи $T_{LLM} = 11{,}7$ год, а умовний тариф на електроенергію – $C_{кВт} = 5$ грн/кВт·год. Тоді витрати на електроенергію визначаються за формулою:

$$C_{ел} = T_{LLM} \cdot P \cdot C_{кВт} = 8{,}78 \text{ грн.} \quad (3.7)$$

Отже, витрати на електроенергію під час виконання місячного обсягу перекладу є незначними та становлять $\approx 9$ грн. Можна врахувати умовні витрати на модернізацію або амортизацію обладнання. Наприклад, якщо для

роботи з локальною моделлю було виконано оновлення апаратної частини комп'ютера на суму $S_2 = 3000$ грн, а строк використання такого оновлення прийнято 36 місяців, то щомісячна амортизаційна складова становитиме:

$$C_{ам} = \frac{S_2}{36} = 83,3 \text{ грн/міс.} \quad (3.8)$$

Визначимо загальні витрати на використання локальної моделі протягом місяця.

Їх можна описати як суму витрат на редагування результату, електроенергію та амортизацію обладнання:

$$C_{LLM} = C_{ред} + C_{ел} + C_{ам} = 1262,08 \text{ грн.} \quad (3.9)$$

Порівняння отриманих результатів наведено у табл. 3.1. Економічний ефект від використання локальної LLM становитиме:

$$E = C_{ручн} - C_{LLM} = 2067,92 \text{ грн/міс.} \quad (3.10)$$

Таблиця 3.1 – Порівняння отриманих результатів

| Показник | Ручний переклад | Переклад із використанням локальної LLM |
|---|---|---|
| Обсяг тексту | 100 сторінок | 100 сторінок |
| Витрати часу | 33,3 год | 11,7 год |
| Вартість роботи | 3330 грн | 1170 грн |
| Витрати на електроенергію | – | 8,78 грн |
| Амортизація обладнання | – | 83,3 грн |
| Загальні витрати | 3330 грн | 1262,08 грн |

Отже, за умов перекладу 100 сторінок тексту на місяць орієнтовна економія становить ≈ 2068 грн на місяць. Якщо враховувати витрати на умовну модернізацію обладнання в розмірі 3000 грн, можна визначити строк окупності такого рішення:

$$T_{окуп} = \frac{S_2}{E} = 1,45 \text{ міс.} \quad (3.11)$$

Таким чином, за наведених умов витрати на оновлення обладнання можуть окупитися приблизно за 1,5 місяця активного використання системи. Локальне використання MamayLM у LM Studio є економічно доцільним за

умови регулярної роботи з текстами. Найбільший ефект досягається за рахунок скорочення часу на переклад і зменшення залежності від ручної праці. Основними факторами витрат залишаються початкове налаштування системи, можливе оновлення обладнання та час на перевірку результату. Використання локальної LLM для автоматизованого перекладу текстів дозволяє зменшити витрати, підняти продуктивність роботи та забезпечити автономність обробки текстових даних.

# ВИСНОВКИ

Основою сучасних мовних моделей, що застосовуються для перекладу є архітектура Transformer, яка використовує механізм самоуваги для встановлення зв’язків між словами в реченні. Трансформерні моделі здатні забезпечувати високу якість перекладу, особливо у випадках, коли необхідно враховувати предметну область і спеціалізовану термінологію. Для підвищення стабільності перекладу великих текстів доцільно застосовувати додаткові підходи, такі як глосарії, розбиття тексту на фрагменти та постобробка.

На основі класифікації мовних моделей за розміром, архітектурою та сферою застосування Встановлено, що вибір мовної моделі залежить від конкретної задачі, вимог до якості результату, швидкості роботи та доступних обчислювальних ресурсів. В роботі досліджено основні методи підвищення ефективності мовних моделей. Розглянуто дистиляцію знань, донавчання та RAG, які дозволяють адаптувати моделі до конкретних задач і зменшити вимоги до ресурсів. Технологія RAG дозволяє доповнювати генерацію зовнішніми знаннями, що підвищує достовірність відповідей і зменшує ризик галюцинацій. Окремо показано, що для задач перекладу найбільш ефективним є поєднання глосаріїв, гібридного пошуку, повторного ранжування та правильного розбиття тексту на фрагменти. Обґрунтовано доцільність використання трансформерної архітектури для побудови системи автоматизованого локального перекладу. Вибір базової мовної моделі має безпосередній вплив на якість розуміння контексту, підтримку багатомовності та вимоги до апаратних ресурсів. Особливу увагу приділено моделям сімейства Gemma 2 та 3.

На основі розгляду архітектурних компонентів моделі запропонована практична реалізація RAG, яка поєднує можливості LLM з пошуком релевантної інформації у зовнішній базі знань. Вона поєднує можливості LLM з пошуком релевантної інформації у зовнішній базі знань. Такий підхід

дозволяє покращити термінологічну узгодженість, зменшити ризик галюцинацій і підвищити якість перекладу спеціалізованих текстів.

Для автоматизованого перекладу особливо перспективними є LLM, що орієнтовані не тільки на виконання загальних багатомовних завдань, а й на врахування особливостей української мови, наприклад, MamayLM та Lapa LLM. Для запуску локальних моделей виконана оцінка властивостей UI-клієнтів. В ході досліджень в якості бази розглядається платформа LM Studio. Практичне тестування підтвердило працездатність локального використання моделі mamaylm-gemma-3-12b-it-v1.0 у LM Studio. Модель успішно обробляє україномовні інструкції, генерує зв'язні відповіді, працює як із простими описовими запитами, так і зі складнішими науково-технічними текстами. Це дає підстави розглядати її як придатний інструмент для автоматизованого перекладу текстів у локальному середовищі. Використання такої моделі може бути корисним у випадках, коли важливими є автономність, конфіденційність, можливість роботи без хмарних сервісів і гнучке налаштування процесу перекладу під потреби користувача. Локальне використання LLM передбачає побудову послідовного pipeline, у якому всі основні етапи обробки виконуються на комп'ютері користувача без обов'язкового звернення до хмарних сервісів, в тому числі з використанням технік RAG. В рамках техніко-економічного обґрунтування прийнятих рішень визначення окупність запропонованих рішень протягом 1,5 місяця за умов імовірної модернізацію обладнання.

Таким чином, результатами роботи є рекомендації щодо практичної реалізації системи автоматизованого локального перекладу; pipeline використання MamayLM у LM Studio, в тому числі, з RAG. Вони можуть бути використані для подальших досліджень за даною тематикою та при проектуванні локальних сервісів NLP.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Why Large Language Models are the future of manufacturing. *Weforum*. URL: https://www.weforum.org/stories/2024/04/why-large-language-models-are-so-important-for-the-future-of-the-manufacturing-industry/ (дата звернення: 05.05.2026)

2. Brown T. et al. Language Models are Few-Shot Learners. arXiv:2005.14165, 2020. URL: https://arxiv.org/abs/2005.14165 (дата звернення: 05.05.2026)

3. Загальні відомості про концепції машинного навчання. *Microsoft Build 2026*. URL: https://learn.microsoft.com/uk-ua/training/modules/fundamentals-machine-learning/ (дата звернення: 05.05.2026)

4. Jurafsky D., Martin J. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3rd ed. *Online manuscript*, 2026. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 05.05.2026)

5. There’s a computer for that. Raspberry Pi. URL: https://www.raspberrypi.com (дата звернення: 05.05.2026)

6. Slyusar V.I. Applications of Large Language Models in the Military Sphere. *Artificial Intelligence: Achievements and Recent Developments*. River Publishers Series in Computing and Information Science and Technology, 2025. P. 53-82. DOI: 10.1201/9788743800989-3.

7. Se K., Vert A. Everything You Need to Know about Knowledge Distillation. Hugging Face. URL: https://huggingface.co/blog/Kseniase/kd (дата звернення: 05.05.2026)

8. Rao S. The Comprehensive Guide to Fine-tuning LLM. *Medium.com*. https://medium.com/data-science-collective/comprehensive-guide-to-fine-tuning-llm-4a8fd4d0e0af (дата звернення: 05.05.2026)

9. Maheshus M. Retrieval-Augmented Generation (RAG): Real Advances in 2025. Medium.com. URL: https://medium.com/@maheshus007/retrieval-augmented-generation-rag-real-advances-in-2025-8a0933ce20c2 (дата звернення: 05.05.2026)

10. Godoy D. A Hands-On Guide to Fine-Tuning Large Language Models with PyTorch and Hugging Face. *Leanpub, Revised Edition*, 2025. URL: https://leanpub.com/finetuning (дата звернення: 05.05.2026)

11. Pedrycz W., Chen S.-M. (eds.) Advancements in Knowledge Distillation: Towards New Horizons of Intelligent Systems. *Cham: Springer*, 2023. 232 p. DOI: 10.1007/978-3-031-32095-8.

12. Lewi P. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401v4, 2021. URL: https://arxiv.org/pdf/2005.11401 (дата звернення: 05.05.2026)

13. Hu Z., Xu Y., Yu W. et al. *DynamicRAG: Leveraging LLM Outputs for Dynamic Re-ranking*. arXiv:2505.07233, 2025. URL: https://arxiv.org/abs/2505.07233 (дата звернення: 05.05.2026)

14. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997, 2023. URL: https://arxiv.org/abs/2312.10997 (дата звернення: 05.05.2026)

15. Ribeiro L. Graph-Augmented Hybrid Retrieval and Multi-Stage Re-ranking Framework. *DEV*, 2025. URL: https://dev.to/lucash_ribeiro_dev/graph-augmented-hybrid-retrieval-and-multi-stage-re-ranking-a-framework-for-high-fidelity-chunk-50ca (дата звернення: 05.05.2026)

16. Knollmeyer S. et al. Document Graph RAG: Knowledge Graph Enhanced Retrieval-Augmented Generation. *Electronics*, 2025, 14(11), 2102. URL: https://doi.org/10.3390/electronics14112102 (дата звернення: 05.05.2026)

17. Merola, C. et al. Reconstructing Context: Evaluating Advanced Chunking Strategies for Retrieval-Augmented Generation. arXiv:2504.19754, 2025. URL: https://arxiv.org/abs/2504.19754 (дата звернення: 05.05.2026)

18. Gemma. Google. URL: https://deepmind.google/models/gemma/ (дата звернення: 05.05.2026)

19. Gemini Google. URL: https://gemini.google.com/app?hl=uk (дата звернення: 05.05.2026)

20. MamayLM. URL: https://huggingface.co/blog/INSAIT-Institute/mamaylm-ukr (дата звернення: 05.05.2026)

21. Lapa LLM. URL: https://huggingface.co/lapa-llm (дата звернення: 05.05.2026)

22. Llama – build on your own terms. URL: https://www.llama.com (дата звернення: 05.05.2026)

23. Mu S., Lin S. A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory and Applications. *arXiv:2503.07137*. URL: https://arxiv.org/abs/2503.07137 (дата звернення: 05.05.2026)

24. NVIDIA A100 | Tensor Core GPU. *NVIDIA*. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf (дата звернення: 05.05.2026)

25. NVIDIA H100. *NVIDIA*. URL: https://www.nvidia.com/en-us/data-center/h100/ (дата звернення: 05.05.2026)

26. Henry A., Dachapally P., Pawar S., Chen Y. Query-Key Normalization for Transformers. arXiv:2010.04245. URL: https://arxiv.org/abs/2010.04245 (дата звернення: 05.05.2026)

27. Онопрієнко О. Open Ukrainian language model Lapa LLM has received a public release. URL: https://dev.ua/en/news/open-ukrainian-language-model-lapa-llm-has-received-a-public-release (дата звернення: 05.05.2026)

28. Open WebUI. URL: https://github.com/open-webui/open-webui (дата звернення: 05.05.2026)

29. Docker. URL: https://www.docker.com (дата звернення: 05.05.2026)

30. LM Studio. URL: https://lmstudio.ai (дата звернення: 05.05.2026)

31. Msty Studio.URL: https://msty.ai (дата звернення: 05.05.2026)

32. Ollama. URL: https://ollama.com (дата звернення: 05.05.2026)

33. AnythingLLM. URL: https://anythingllm.com (дата звернення: 05.05.2026)

34. Cherry Studio. URL: https://www.cherry-ai.com (дата звернення: 05.05.2026)

35. Private-gpt. URL: https://privategpt.io (дата звернення: 05.05.2026)

36. Run LLMs anywhere, with KoboldCpp. URL: https://koboldcpp.com (дата звернення: 05.05.2026)

37. SillyTavern. LLM Frontend for Power Users. URL: https://sillytavern.app (дата звернення: 05.05.2026)

38. TextGen. URL: https://github.com/oobabooga/textgen (дата звернення: 05.05.2026)

39. Готра М. Система автоматизованого локального перекладу на основі MamayLM і LM Studio. *Збірник XXI щорічної студентської наукової конференції «Сучасні інформаційні технології та інноваційні методики в економіці, менеджменті та бізнесі» Полтавського державного аграрного університету* (квітень 2026 р., м. Полтава). Полтава: ПДАУ, 2026 р. С. 62, 63.

40. MamayLM-Gemma-3-12B-IT-v1.0. URL: https://huggingface.co/INSAIT-Institute/MamayLM-Gemma-3-12B-IT-v1.0 (дата звернення: 05.05.2026)