

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКР АІНІ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО

В.о. завідувача кафедри

Олександр КОВАЛЬ

«__» _____ 2026 р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Параметричне керування великими мовними моделями»**

Виконав:

Студент IV курсу, групи ТВ-23

Гайдук Максим Олександрович

(прізвище, ім'я, по батькові)

(підпис)

Керівник:

доцент, к.н.т., доцент, Стативка. Ю.І.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент:

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

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

Студент

(підпис)

Київ – 2026

**Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення
інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Олександр КОВАЛЬ

«__» _____ 2026р.

ЗАВДАННЯ
на дипломну роботу студенту

Гайдуку Максиму Олександровичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Параметричне керування великими мовними моделями

керівник роботи Стативка Юрій Іванович, к.н.т., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «28» травня 2026 р. № 1992-с

2. Строк подання студентом роботи «10» червня 2026 р.

3. Вихідні дані до роботи: мова програмування Python, середовище розробки Codex

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): Відсортувати датасети і натренувати LoRA адаптери щоб реалізувати параметричне керування великою мовною моделлю. Верифікувати результати і створити зручну платформу для перегляду результатів генерацій моделі з використанням створених LoRA адаптерів

5. Перелік ілюстративного матеріалу: Графіки, інтерфейс програми

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	31.10.2025	Виконано
2	Дослідження предметної області	01.11.2025 – 31.11.2025	Виконано
3	Дослідження існуючих рішень	01.12.2025 – 31.12.2025	Виконано
4	Постановка вимог до проєктування системи	01.01.2026 – 31.01.2026	Виконано
5	Розробка програмного продукту	01.02.2026 – 03.05.2026	Виконано
6	Тестування	04.05.2026 – 10.05.2026	Виконано
7	Захист програмного продукту	11.05.2026 – 15.05.2026	Виконано
8	Оформлення дипломної роботи	18.05.2026 – 31.05.2026	Виконано
9	Передзахист	01.06.2026 – 05.06.2026	Виконано
10	Захист	15.06.2026 – 19.06.2026	Виконано

Студент



(підпис)

Максим ГАЙДУК

(ім'я, ПРІЗВИЩЕ)

Керівник роботи

(підпис)

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Дипломна робота складається зі вступу, шести розділів, висновків, списку використаних джерел і додатків. Робота містить 57 сторінок, 5 рисунків, 5 таблиць, 2 додатки та 13 джерел. Точні значення необхідно оновити після остаточного форматування пояснювальної записки.

Метою роботи є реалізація та експериментальна перевірка системи параметричного керування стилем відповідей великої мовної моделі українською та англійською мовами. У роботі досліджується підхід, за якого стиль відповіді змінюється не лише через текстові інструкції у промпті, а через LoRA-адаптери та коефіцієнт сили їхнього впливу.

У процесі виконання роботи підготовлено двомовний набір промптів, сформовано нейтральні та стилізовані цільові відповіді, очищено дані, навчено чотири LoRA-адаптери для стилів *formal*, *informal*, *concise* і *verbose*, реалізовано інференс для *baseline*-режиму, одиночних адаптерів, різних значень *alpha* та композицій адаптерів. Для оцінювання результатів використано метрики мовної коректності, семантичної подібності, співвідношення довжини, формальності, наявності некоректних відповідей і ознак обрізання генерації.

Практичне значення одержаних результатів полягає у можливості змінювати стиль відповіді мовної моделі без повного донавчання базової моделі. Розроблений *pipeline* дозволяє навчати окремі адаптери, запускати керовану генерацію, оцінювати результати та переглядати їх у вебінтерфейсі *Results Explorer*.

Ключові слова: ВЕЛИКА МОВНА МОДЕЛЬ, LM STEERING, LORA, PEFT, АДАПТЕР, ПАРАМЕТРИЧНЕ КЕРУВАННЯ, ГЕНЕРАЦІЯ ТЕКСТУ, СТИЛЬ ВІДПОВІДІ, GEMMA, RESULTS EXPLORER.

ABSTRACT

Structure and scope of the thesis. The thesis consists of an introduction, six chapters, conclusions, a list of references and appendices. The work contains 57 pages, 5 figures, 5 tables, 2 appendices and 13 references. The exact numbers should be updated after the final formatting of the document.

The purpose of the work is to implement and experimentally evaluate a system for parametric steering of large language model response style in Ukrainian and English. The thesis investigates an approach in which response style is controlled not only through textual prompt instructions, but also through LoRA adapters and a numeric coefficient that defines the strength of adapter influence.

During the work, a bilingual prompt dataset was prepared, neutral and styled target answers were generated, the data were cleaned, and four LoRA adapters were trained for the formal, informal, concise and verbose styles. The system supports inference in the baseline mode, with a single active adapter, with different alpha values and with adapter compositions. The generated outputs were evaluated using automatic metrics, including language correctness, semantic similarity, length ratio, formality score, invalid output rate and truncation proxy.

The practical value of the obtained results lies in the ability to change the response style of a language model without full fine-tuning of the base model. The developed pipeline makes it possible to train separate adapters, run controlled generation, evaluate results and inspect them through the Results Explorer web interface.

Keywords: LARGE LANGUAGE MODEL, LM STEERING, LORA, PEFT, ADAPTER, PARAMETRIC STEERING, TEXT GENERATION, RESPONSE STYLE, GEMMA, RESULTS EXPLORER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	9
1 ПОСТАНОВКА ЗАДАЧІ ТА ВИМОГИ	14
1.1 Предметна область і поняття LM Steering.....	14
1.2 Керовані властивості відповіді.....	15
1.3 Функціональні вимоги	16
1.4 Нефункціональні вимоги та обмеження.....	18
1.5 Критерії успішності експерименту	19
Висновки до розділу.....	20
2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ	22
2.1 Prompt engineering.....	22
2.2 Повне донавчання моделі	23
2.3 RLHF та інструкційне вирівнювання.....	24
2.4 LoRA та PEFT	24
2.5 Activation steering і parameter steering.....	26
2.6 Обґрунтування вибору підходу	26
Висновки до розділу.....	27
3 ТЕХНОЛОГІЧНИЙ СТЕК І ЗАСОБИ РОЗРОБКИ.....	28
3.1 Мова програмування та середовище виконання	28
3.2 Бібліотеки для роботи з мовними моделями.....	28
3.3 Підготовка та зберігання даних.....	29
3.4 Генерація еталонних відповідей.....	29
3.5 Обчислювальна інфраструктура	31
3.6 Вебзастосунок для демонстрації.....	31
Висновки до розділу.....	32
4 АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ СИСТЕМИ.....	33
4.1 Загальна архітектура pipeline.....	33
4.2 Підготовка двомовного набору даних	34
4.3 Формування стилізованих відповідей	35
4.4 Навчання LoRA-адаптерів	36
4.5 Інференс і умови керування.....	38
4.6 Автоматичне оцінювання результатів	39
4.7 Експорт артефактів для вебінтерфейсу	41
Висновки до розділу.....	42
5 РОБОТА СИСТЕМИ ТА ІНТЕРФЕЙС	44
5.1 Сценарій використання системи	44
5.2 Головна сторінка вебзастосунку Results Explorer	45
5.3 Перегляд промптів і умов керування.....	46
5.4 Візуалізація метрик	47
5.5 Ручна оцінка відповідей.....	50
Висновки до розділу.....	51

6 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОЇ ПЕРЕВІРКИ.....	52
6.1 Підсумкові результати експериментів.....	52
6.2 Інтерпретація впливу коефіцієнтів керування.....	53
6.3 Обмеження реалізованого підходу	55
6.4 Напрями подальшого розвитку	56
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТОК А Лістинг розробленої системи	62
ДОДАТОК Б Презентація	99

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Термін	Пояснення
LLM	Large Language Model, велика мовна модель, нейромережева модель, навчена на великих обсягах тексту і здатна генерувати відповіді на основі вхідного запиту
BMM	Велика мовна модель; український відповідник скорочення LLM
LM Steering	Керування поведінкою мовної моделі під час генерації (зміна стилю, довжини та формальності відповіді без повного донавчання)
Prompt	Текстовий запит користувача, який подається на вхід мовної моделі
Prompt engineering	Формулювання спеціальних текстових інструкцій у промпті для керування відповіддю мовної моделі
LoRA	Low-Rank Adaptation, метод параметрично ефективного донавчання через додавання низькорангових матриць адаптера з замороженими основними параметрами
PEFT	Parameter-Efficient Fine-Tuning, група методів параметрично ефективного донавчання великих моделей
Adapter	Додатковий набір параметрів, що підключається до базової моделі та змінює її поведінку без зміни всіх основних ваг
Alpha	Числовий коефіцієнт, що визначає силу впливу адаптера під час генерації
Baseline	Базовий режим генерації без активного адаптера
Discrete steering	Дискретне керування, коли активується один конкретний адаптер із фіксованою вагою
Continuous steering	Неперервне або масштабоване керування, коли сила впливу адаптера змінюється числовим коефіцієнтом alpha

ВСТУП

Сучасні великі мовні моделі здатні генерувати тексти високої якості, однак у прикладних задачах важливо не лише отримати правильний зміст, а й керувати формою відповіді. [1, 5] У багатьох сценаріях використання мовна модель виступає не просто генератором тексту, а компонентом інформаційної системи, який має працювати відповідно до контексту, призначення повідомлення та очікувань користувача. Через це одна й та сама фактична відповідь може бути непридатною для практичного використання, якщо вона має неправильний тон, надмірну довжину, недостатню деталізацію або невідповідний рівень офіційності.

Один і той самий зміст може бути поданий офіційно або неформально, стисло або розгорнуто, українською або англійською мовою. Для корпоративної комунікації, клієнтських консультацій, описів товарів, службових листів і внутрішніх інформаційних матеріалів така керованість є практично важливою. Наприклад, відповідь для технічної документації має бути структурованою, точною та нейтральною, тоді як відповідь для короткого повідомлення користувачу може бути простішою, менш формальною і значно коротшою. У цьому випадку проблема полягає не в тому, що модель не знає потрібної інформації, а в тому, що вона не завжди подає цю інформацію у потрібній формі.

Питання керування стилем відповіді є особливо актуальним для багатомовних систем. Якщо система підтримує українську та англійську мови, вона має не тільки генерувати текст правильною мовою, а й зберігати керовану властивість у кожній із мов. Це ускладнює задачу, оскільки модель може по-різному реагувати на однакові стильові умови залежно від мови промпта, структури запиту та характеру навчальних прикладів. Тому в межах роботи важливо було оцінювати не лише загальну якість генерації, а й стабільність поведінки окремо для українських та англійських промптів.

Типовий спосіб керування стилем полягає у додаванні інструкцій до промпта. Наприклад, користувач може попросити модель відповісти формально або коротко.

Проте такий підхід не завжди стабільний: інструкція займає частину контексту, може конфліктувати з основним запитом і не дає прямого контролю над параметрами моделі. У дипломному проєкті досліджується інший підхід: керування поведінкою моделі через окремі параметричні зміни, реалізовані у вигляді LoRA-адаптерів. [2, 8] Основна проблема полягає в тому, що мовна модель після навчання має фіксовану поведінку, а користувачеві часто потрібні різні режими генерації. Для одного запиту може бути потрібна відповідь:

- формальна, придатна для ділового листування;
- неформальна, придатна для дружньої консультації;
- стисла, коли важлива швидка відповідь;
- розгорнута, коли потрібне пояснення або деталізація.
- При цьому зміна стилю не повинна руйнувати зміст.

Система має змінювати тон або довжину відповіді, але зберігати мову, основну семантику та коректність відповіді. Саме баланс між керованістю та збереженням змісту є центральною експериментальною задачею роботи.

Ця задача має два взаємопов'язані аспекти. Перший аспект полягає у створенні самих стильових напрямів керування. Для цього потрібно сформулювати навчальні приклади, у яких одна й та сама відповідь подана у кількох варіантах: формальному, неформальному, стислому та розгорнутому. Другий аспект полягає у перевірці того, що навчені адаптери справді змінюють потрібну властивість, а не просто створюють інший випадковий текст. Тому важливо мати не лише генерацію, а й систему оцінювання, яка дозволяє порівнювати відповіді між собою.

З практичної точки зору така система може бути корисною як проміжний шар між базовою мовною моделлю та прикладним застосунком. Замість того щоб щоразу змінювати системний промпт або навчати окрему модель під кожен стиль, можна мати одну базову модель і набір адаптерів. У такому випадку прикладний сервіс може вмикати потрібний адаптер залежно від сценарію: наприклад, *formal* для службових повідомлень, *concise* для коротких відповідей у чаті підтримки,

verbose для навчальних пояснень, informal для менш офіційного спілкування з користувачем.

Окремою перевагою є те, що параметричний підхід дає змогу досліджувати силу впливу адаптера. Якщо керування відбувається лише промптом, то інтенсивність стилю задається словами й залежить від інтерпретації моделі. Якщо ж керування здійснюється через вагу адаптера, з'являється числовий параметр α , який можна змінювати, порівнювати та оцінювати експериментально. Це робить підхід зручнішим для дослідницького аналізу та для подальшої інтеграції у програмні системи.

Метою дипломного проєкту є реалізація та експериментальна перевірка системи LM-Steer-подібного керування параметрами мовної моделі для зміни стилістичних властивостей відповідей українською та англійською мовами.

Досягнення цієї мети передбачає не лише створення програмного коду для запуску моделі, а й побудову повного експериментального процесу. Такий процес має охоплювати підготовку двомовних даних, генерацію еталонних відповідей, очищення прикладів, навчання адаптерів, запуск інференсу з різними умовами керування, автоматичне оцінювання відповідей і візуальне представлення результатів. Тому робота має характер не лише реалізації окремого алгоритму, а й розробки цілісного pipeline для дослідження параметричного керування мовною моделлю.

Об'єктом дослідження є процес генерації тексту великою мовною моделлю. У межах цього процесу розглядається ситуація, коли модель отримує текстовий запит користувача та формує відповідь природною мовою. Основна увага приділяється не зміні фактичних знань моделі, а зміні форми подання відповіді.

Предметом дослідження є параметрично ефективне керування стилем відповіді за допомогою LoRA-адаптерів і коефіцієнтів їхнього впливу. Предмет охоплює спосіб навчання адаптерів, спосіб їх активації під час інференсу, можливість масштабування сили адаптера через α , а також можливість композиції кількох адаптерів.

У роботі використовується модель **google/gemma-3-4b-it**. [5 - 7] Вона обрана як основна експериментальна модель, оскільки підтримує багатомовну генерацію, має достатню якість для задачі та може запускатися на одному GPU класу A40.

Для досягнення мети потрібно виконати такі завдання:

1. Підготувати двомовний набір промптів українською та англійською мовами.
2. Згенерувати нейтральні та стилізовані цільові відповіді для навчання адаптерів.
3. Очистити дані від прикладів із неправильною мовою, відмовами, зміною змісту або невідповідною довжиною.
4. Навчити окремі LoRA-адаптери для стилів **formal**, **informal**, **concise** і **verbose**.
5. Реалізувати інференс із базовим режимом, дискретним керуванням, безперервним масштабуванням **alpha** і композицією кількох адаптерів.
6. Оцінити результати за мовною коректністю, семантичною подібністю, довжиною відповіді, формальністю і часткою некоректних генерацій.
7. Підготувати вебзастосунок для перегляду результатів, графіків і прикладів відповідей.

Розроблена система демонструє, що поведінку мовної моделі можна змінювати не лише через текстові інструкції, а й через керовані параметричні модифікації. Це корисно для задач, де одна базова модель має працювати в кількох режимах без повного донавчання під кожен сценарій.

Практичне значення роботи полягає в можливості будувати системи генерації тексту, які швидко перемикаються між стилями відповіді. Наприклад, одна система може створювати формальні службові повідомлення, короткі відповіді для підтримки користувачів, розгорнуті пояснення або менш офіційні консультаційні тексти. Додаткова цінність полягає в тому, що експеримент проводиться для двох мов, що дозволяє оцінити стабільність керування не лише для англійської, а й для української генерації.

Актуальність роботи визначається кількома чинниками. По-перше, великі мовні моделі дедалі частіше використовуються як частина прикладних програмних систем, а не лише як окремі чат-боти. Це означає, що їхня поведінка має бути контрольованою, передбачуваною та придатною до інтеграції у більші програмні комплекси. По-друге, у реальних задачах користувачу потрібна не просто відповідь, а відповідь у конкретному форматі. Вона може мати вимоги до довжини, тону, рівня деталізації, мови та стилю. По-третє, повне донавчання великих моделей є дорогим і не завжди виправданим, особливо якщо потрібно змінити лише стиль відповіді, а не фактичні знання моделі.

Науково-практичний інтерес роботи полягає у перевірці того, чи можна використовувати LoRA-адаптери як керовані напрями зміни поведінки моделі. Якщо такий підхід працює, він дозволяє відокремити базову здатність моделі генерувати змістовні відповіді від окремих стильових налаштувань. У такому випадку одна базова модель може підтримувати кілька режимів роботи, а зміна режиму відбувається не через повне перенавчання, а через підключення або масштабування відповідного адаптера.

У межах цієї роботи актуальність також пов'язана з підтримкою української мови. Більшість досліджень і прикладів використання великих мовних моделей орієнтовані насамперед на англійську. Проте для прикладних систем в українському освітньому, корпоративному та технічному середовищі важливо оцінювати якість керування саме для українських текстів. Тому двомовний характер експерименту є суттєвою частиною практичного значення роботи.

1 ПОСТАНОВКА ЗАДАЧІ ТА ВИМОГИ

1.1 Предметна область і поняття LM Steering

LM Steering означає керування поведінкою мовної моделі під час генерації. У широкому розумінні це сукупність підходів, які дозволяють змінювати спосіб відповіді моделі без обов'язкового повного перенавчання її основних параметрів. Поведінка мовної моделі може включати тон відповіді, стиль викладу, довжину, рівень деталізації, формальність, безпечність, схильність до пояснення або до стислого формулювання. У цій роботі увага зосереджена саме на стилістичних властивостях, які можна оцінювати через зміну формальності та довжини відповіді.

На відміну від звичайного prompt engineering, де бажану поведінку описують словами у вхідному запиті, steering-підхід намагається впливати на внутрішні параметри або представлення моделі. Це не означає, що текстові інструкції стають непотрібними. Навпаки, prompt engineering залишається простим і корисним інструментом. Однак він має обмеження: інструкція може бути проігнорована, може конфліктувати з основним запитом або займати значну частину контексту. Крім того, текстова інструкція не завжди дозволяє плавно змінювати силу стилю. Наприклад, складно формально визначити, наскільки саме відповідь має бути «трохи формальнішою» або «значно коротшою».

У цій роботі використано параметричний варіант такого керування: для кожної властивості відповіді навчається окремий LoRA-адаптер, а під час інференсу адаптер активується з певною вагою. [2, 8] Такий підхід дозволяє розглядати адаптер як окремий напрям у просторі параметрів моделі. Якщо адаптер навчений на формальних відповідях, його активація має зміщувати генерацію у бік більш формального стилю. Якщо адаптер навчений на стислих відповідях, його активація має сприяти скороченню тексту.

Базова модель залишається незмінною. Для адаптера навчаються лише додаткові матриці низького рангу. [2] У спрощеному вигляді модифікацію ваг можна записати так:

$$\mathbf{W}' = \mathbf{W} + \alpha * \mathbf{\Delta W}$$

$$\mathbf{\Delta W} = \mathbf{A} * \mathbf{B}$$

Тут $\mathbf{W}$ є початковою матрицею ваг базової моделі, $\mathbf{\Delta W}$ є зміною, яку додає адаптер, а α є коефіцієнтом сили керування. Якщо $\alpha = 0$, модель працює як базова. Якщо $\alpha > 0$, вплив адаптера посилюється. Негативні значення α використовуються для перевірки того, як модель поводить себе при протилежному напрямі зміни.

Важливо, що в цій роботі α розглядається не як довільний параметр генерації на кшталт temperature або top_p, а як коефіцієнт, який впливає саме на підключений адаптер. Тобто temperature змінює випадковість вибору наступних токенів, а α змінює внесок навченого стильового напрямку. Це робить експеримент більш інтерпретованим: можна порівнювати, як різні значення α впливають на довжину, формальність, мову та семантичну подібність відповіді.

1.2 Керовані властивості відповіді

У межах дипломного проєкту досліджуються дві групи властивостей:

- формальність: formal і informal;
- довжина відповіді: concise і verbose.

Вибір саме цих властивостей пояснюється тим, що вони є практично зрозумілими та відносно незалежними. Формальність переважно впливає на тон, лексику та спосіб звертання, тоді як довжина відповіді впливає на обсяг тексту та рівень деталізації. Такий поділ дозволяє окремо оцінювати, чи справді адаптер змінює ту властивість, для якої він навчався.

Формальність відповідає за тон повідомлення. Формальна відповідь має бути придатною для офіційної або корпоративної комунікації, тоді як неформальна

може використовувати простіші конструкції та менш офіційний стиль. Формальний стиль зазвичай характеризується нейтральною лексикою, відсутністю розмовних скорочень, більш обережними формулюваннями та структурованістю. Неформальний стиль, навпаки, може бути ближчим до звичайного пояснення у діалозі, використовувати коротші речення і менш офіційну подачу.

Довжина відповіді відповідає за деталізацію. Стислий режим має скорочувати текст без втрати основного змісту, а розгорнутий режим має додавати пояснення або деталі, не змінюючи початкову ідею відповіді. У цій задачі важливо не просто зробити текст коротшим або довшим. Стисла відповідь має залишатися корисною, а розгорнута відповідь не повинна додавати вигадані або зайві факти. Тому довжина розглядається разом із семантичною подібністю до нейтрального референсу.

Для кожної з чотирьох властивостей було обрано окремий адаптер. Такий дизайн має кілька переваг. По-перше, він спрощує інтерпретацію: якщо активовано formal-адаптер, очікується саме зміна формальності. По-друге, він дозволяє порівнювати адаптери між собою. По-третє, він створює можливість композиції, коли дві властивості комбінуються одночасно, наприклад formal + concise або informal + verbose.

Окремою вимогою є збереження мови. Керування стилем не повинно призводити до переходу з української на англійську або навпаки. Через це в системі використовується метрика language_ok, яка перевіряє, чи відповідає мова генерації мові початкового промпта. Для двомовного експерименту це є важливим критерієм, оскільки стильове керування не можна вважати успішним, якщо відповідь змінює мову.

1.3 Функціональні вимоги

Функціональні вимоги визначають, які дії має виконувати система для досягнення мети дипломної роботи. Оскільки система є не лише демонстраційним

застосунком, а й експериментальним pipeline, вимоги охоплюють як підготовку даних, так і навчання, генерацію, оцінювання та перегляд результатів. Система повинна підтримувати такі функції:

1. Робота з промптами українською та англійською мовами.
2. Підготовка набору даних із нейтральними та стилізованими відповідями.
3. Навчання чотирьох окремих LoRA-адаптерів: formal, informal, concise, verbose.
4. Генерація відповіді без адаптера, тобто базовий режим.
5. Генерація з одним активним адаптером, тобто дискретне керування.
6. Генерація з масштабованим адаптером для значень $\alpha \in \{-2, -1, 0, 1, 2\}$.
7. Композиція двох адаптерів у режимах formal + concise і informal + verbose.
8. Збереження генерацій у табличному форматі для подальшого аналізу.
9. Автоматичне оцінювання результатів за якісними та кількісними метриками.
10. Експорт результатів у вебінтерфейс для демонстрації та ручної перевірки.

Крім основних функцій, система має забезпечувати логічне розділення етапів. Підготовка даних не повинна бути жорстко пов'язана з навчанням, навчання — з інференсом, а інференс — з вебінтерфейсом. Таке розділення потрібне для того, щоб у разі зміни одного етапу не запускати весь процес повторно. Наприклад, якщо потрібно змінити лише метрику оцінювання, немає потреби заново навчати адаптери або генерувати всі відповіді.

Окремою функціональною вимогою є збереження проміжних артефактів. Для цієї роботи важливо мати можливість перевірити, які саме промпти потрапили в clean subset, які відповіді були згенеровані, які умови керування

використовувалися та які метрики отримані для кожної генерації. Тому система має зберігати не лише фінальні підсумки, а й детальні таблиці результатів.

Також важливо, щоб вебінтерфейс не був залежним від наявності GPU або запущеної моделі. Він має працювати як шар перегляду над готовими файлами. Це дозволяє демонструвати результати на звичайному комп'ютері або у браузері без повторного запуску важких обчислень.

1.4 Нефункціональні вимоги та обмеження

Нефункціональні вимоги визначають не те, які саме дії виконує система, а те, якою вона має бути з погляду якості реалізації, зручності відтворення та обмежень середовища. Для експериментальної роботи з великими мовними моделями ці вимоги є особливо важливими, оскільки навчання та інференс потребують значних обчислювальних ресурсів, а результати можуть залежати від версій моделей, параметрів запуску та якості підготовлених даних.

Система повинна бути відтворюваною: усі основні етапи виконуються окремими Python-скриптами, а проміжні результати зберігаються у файлах parquet, json і csv. Це дає змогу повторно запускати окремі частини pipeline без повного перерахунку всього експерименту. Наприклад, після завершення інференсу можна не генерувати відповіді заново, а лише змінити код оцінювання і повторно порахувати метрики. Так само після підготовки clean subset можна навчати різні адаптери на однаковому наборі даних.

Обчислювальне обмеження полягає у використанні одного GPU RunPod A40 з 48 GB відеопам'яті. Тому повне донавчання моделі не є раціональним для цієї роботи. Замість цього використано LoRA, яка навчає тільки невелику кількість додаткових параметрів. [2, 3, 8] Такий підхід відповідає обмеженням дипломного проекту, оскільки дозволяє провести експеримент з моделлю класу Gemma 3 4B без необхідності у багатокартковій інфраструктурі.

Мовне обмеження полягає в тому, що система повинна зберігати мову промпта. Якщо запит українською, відповідь має бути українською; якщо запит англійською, відповідь має бути англійською. Це обмеження впливає на підготовку даних, очищення відповідей і автоматичне оцінювання. Приклад, який змінює мову відповіді, не може вважатися якісним навіть тоді, коли його зміст загалом подібний до нейтрального референсу.

До нефункціональних вимог також належить модульність. Кожен етап pipeline повинен мати зрозумілий вхід і вихід. Підготовка даних створює очищені таблиці, навчання створює адаптери, інференс створює таблицю генерацій, оцінювання створює таблицю метрик, а експорт створює файли для вебінтерфейсу. Така структура зменшує ризик помилок і полегшує налагодження.

Ще однією вимогою є прозорість результатів. Система не повинна обмежуватися одним середнім числом якості. Для аналізу потрібно бачити окремі відповіді, умови керування, значення alpha, мову промпта, текст генерації, нейтральний референс і всі доступні метрики. Саме тому результати експортуються у вебінтерфейс Results Explorer.

1.5 Критерії успішності експерименту

Критерії успішності потрібні для того, щоб оцінити роботу системи не лише суб'єктивно, а й через вимірювані ознаки. У задачі стильового керування неможливо обмежитися перевіркою того, що модель просто згенерувала текст. Важливо визначити, чи цей текст відповідає мові промпта, чи зберігає основний зміст, чи змінює потрібну властивість і чи не стає некоректним або порожнім.

Експеримент вважається успішним, якщо система:

- генерує відповіді правильною мовою у більшості випадків;
- зберігає зміст нейтральної відповіді при зміні стилю;
- демонструє вимірювану зміну формальності або довжини залежно від активного адаптера;

- дозволяє порівняти базовий режим, одиночні адаптери, різні значення `alpha` і композиції адаптерів;
- надає результати у вигляді таблиць, графіків і вебінтерфейсу для аналізу.

Кожен із цих критеріїв відповідає окремій частині `pipeline`. Мовна коректність перевіряється на етапі автоматичного оцінювання. Збереження змісту оцінюється через `semantic_similarity`. Зміна довжини перевіряється через `length_ratio`, а зміна формальності — через `formality_score`. Наявність графіків і вебінтерфейсу дозволяє не лише отримати підсумкові значення, а й побачити поведінку системи на конкретних прикладах.

У фінальному експерименті отримано 3075 генерацій для 123 `eval`-промптів і 25 умов керування. Це дає достатню кількість прикладів для порівняння базового режиму, неперервного керування та композицій. Важливо, що однакові `eval`-промпти використовуються для різних умов. Завдяки цьому можна порівнювати не просто різні випадкові відповіді, а реакцію моделі на один і той самий запит при різних налаштуваннях керування.

Висновки до розділу

У першому розділі було сформульовано предметну область дипломної роботи та визначено основну задачу — параметричне керування стилем відповідей великої мовної моделі. Було розглянуто поняття `LM Steering` і показано відмінність між керуванням через текстові інструкції та керуванням через параметри або адаптери моделі.

Також було визначено основні керовані властивості відповіді: формальність і довжину. Для формальності розглядаються стилі `formal` та `informal`, а для довжини — `concise` та `verbose`. Такий вибір дозволяє оцінювати як зміну тону відповіді, так і зміну рівня деталізації.

У розділі сформульовано функціональні та нефункціональні вимоги до системи. Функціональні вимоги охоплюють підготовку двомовних даних, навчання LoRA-адаптерів, генерацію відповідей у різних умовах, автоматичне оцінювання та експорт результатів у вебінтерфейс. Нефункціональні вимоги стосуються відтворюваності, модульності, обмежень обчислювальної інфраструктури та підтримки української й англійської мов.

Крім того, було визначено критерії успішності експерименту. Успішною система вважається тоді, коли вона зберігає мову і зміст відповіді, але при цьому демонструє вимірювану зміну стилістичних властивостей залежно від активного адаптера та значення коефіцієнта α .

2 АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ

2.1 Prompt engineering

Найпростішим способом керування поведінкою мовної моделі є prompt engineering. У цьому підході користувач додає до запиту інструкцію на кшталт "відповідай формально", "пиши коротко" або "поясни детально". Перевага такого підходу полягає у простоті: він не потребує навчання моделі, додаткових ваг або спеціальної інфраструктури.

Prompt engineering є природним першим кроком у роботі з великими мовними моделями. Він дозволяє швидко перевірити, чи здатна модель змінювати стиль відповіді за текстовою інструкцією. Наприклад, можна подати один і той самий запит із різними інструкціями: "відповідай офіційно", "відповідай коротко", "поясни простими словами". Такий підхід добре підходить для інтерактивного використання, коли користувач вручну уточнює бажану форму відповіді.

Недолік prompt engineering полягає в нестабільності. Модель може частково ігнорувати інструкцію, змішувати кілька вимог або віддавати перевагу змісту промпта над стильовою умовою. Крім того, інструкції займають частину контексту і мають повторюватися для кожного запиту. Для дипломної задачі це не дає достатньо чіткого параметричного контролю.

Ще одним обмеженням є складність кількісного аналізу. Якщо стиль задається словами, то важко визначити, наскільки сильним є це керування. Дві різні інструкції можуть мати схожий зміст, але модель реагуватиме на них по-різному. Наприклад, інструкції "зроби відповідь формальною" і "відповідай у діловому стилі" можуть давати різні результати, хоча для людини вони близькі за змістом. Це ускладнює побудову відтворюваного експерименту.

У контексті цієї роботи prompt engineering розглядається як базовий підхід і як причина для пошуку альтернативи. Він не відкидається повністю, але не є основним механізмом керування. Мета роботи полягає в тому, щоб перенести

частину керування зі словесної інструкції у параметричний простір адаптера, де силу впливу можна задавати числово.

2.2 Повне донавчання моделі

Повне донавчання змінює всі або більшість параметрів мовної моделі на новому наборі даних. [2, 3] Такий підхід може дати сильний ефект, але має значні обчислювальні вимоги. Для моделей із мільярдами параметрів повне донавчання потребує багато відеопам'яті, часу та дискового простору.

Повне донавчання доцільне тоді, коли необхідно суттєво змінити поведінку моделі, додати нові знання або адаптувати модель до великого спеціалізованого корпусу. Наприклад, якщо модель має працювати у вузькій галузі та відповідати на питання з великою кількістю специфічних термінів, повне або часткове донавчання може бути виправданим. Однак у задачі цієї дипломної роботи основна мета інша: потрібно не навчити модель новій предметній області, а змінювати форму вже можливої відповіді.

Для цієї роботи повне донавчання є надмірним. Завдання полягає не в тому, щоб навчити модель новим фактам, а в тому, щоб керовано змінювати форму відповіді. Тому ефективнішим є метод, який додає невелику кількість параметрів і залишає базову модель замороженою.

Крім обчислювальної вартості, повне донавчання має ще одну проблему: воно створює одну нову версію моделі. Якщо потрібно підтримувати чотири стилі, можна було б навчати чотири окремі моделі, але це збільшило б витрати на зберігання, запуск і підтримку. LoRA-адаптери дозволяють уникнути цього, оскільки одна базова модель може мати кілька невеликих додаткових адаптерів.

2.3 RLHF та інструкційне вирівнювання

RLHF і подібні методи вирівнювання використовують людський або модельний зворотний зв'язок для формування бажаної поведінки моделі. Вони широко застосовуються для підвищення безпечності, корисності та відповідності інструкціям.

Інструкційне вирівнювання допомагає моделі краще виконувати запити користувача та відповідати у форматі діалогу. Такі підходи є важливими для створення універсальних асистентів, оскільки вони формують загальну поведінку моделі після базового попереднього навчання. Проте вони зазвичай спрямовані на глобальне покращення моделі, а не на швидке перемикання між кількома окремими режимами.

Однак для дипломного проєкту цей клас методів не є основним. RLHF потребує окремої процедури збору оцінок, моделі винагороди та складнішого циклу навчання. Крім того, він більше підходить для глобального налаштування поведінки, ніж для швидкого перемикання між кількома стилями відповіді.

Для задачі цієї роботи важливо мати легкий і відтворюваний механізм керування. У випадку RLHF зміна одного стилю може вимагати переробки значної частини навчального процесу. У випадку адаптерів можна навчити окремий адаптер для окремої властивості, зберегти його як самостійний артефакт і підключати лише тоді, коли він потрібен.

2.4 LoRA та PEFT

LoRA належить до методів параметрично ефективного донавчання. [2, 8] Ідея полягає в тому, що замість зміни повної матриці ваг модель отримує низькорангову поправку. [2] Базові параметри залишаються замороженими, а навчаються лише невеликі матриці адаптера.

Основна ідея LoRA базується на припущенні, що для багатьох задач не потрібно змінювати всі параметри великої моделі. Достатньо додати обмежену за рангом зміну до окремих матриць у шарах моделі. Це зменшує кількість параметрів, які потрібно навчати, і робить процес донавчання значно дешевшим. Для практичного дипломного проєкту це особливо важливо, оскільки експеримент має виконуватися в обмеженому обчислювальному середовищі.

Переваги LoRA для цієї роботи:

- значно менші вимоги до відеопам'яті порівняно з повним донавчанням;
- можливість зберігати окремі адаптери для окремих властивостей;
- швидке перемикання між режимами поведінки;
- можливість комбінувати кілька адаптерів під час інференсу;
- сумісність із бібліотеками transformers і peft. [8, 9]

Окремі адаптери зручні для організації експерименту. Наприклад, formal-адаптер можна навчати тільки на формальних відповідях, concise-адаптер — тільки на стислих відповідях. Після цього кожен адаптер може розглядатися як окремий модуль, який підключається до базової моделі. Така модульність добре відповідає архітектурі програмної системи, де окремі компоненти мають зрозумілі входи, виходи та призначення.

Ще однією перевагою є можливість композиції. Якщо адаптери зберігають певні напрями зміни поведінки, можна спробувати одночасно підключити два напрями. У роботі перевіряються композиції formal + concise та informal + verbose. Це дозволяє дослідити, чи може модель одночасно змінювати тон і довжину відповіді.

Саме ці властивості роблять LoRA придатною для LM-Steer-подібного керування параметрами у межах дипломної роботи. Метод забезпечує компроміс між практичною доступністю, гнучкістю керування і можливістю експериментального аналізу.

2.5 Activation steering i parameter steering

Activation steering впливає на внутрішні активації моделі під час генерації. Зазвичай він потребує доступу до прихованих станів і додавання певного вектора керування у вибраних шарах. Такий підхід може бути гнучким, але його складніше інтегрувати у стандартний pipeline навчання та інференсу.

Parameter steering, навпаки, змінює або доповнює параметри моделі. У цій роботі він реалізований через LoRA-адаптери. Керування відбувається не через текстову інструкцію, а через вагу адаптера. Це дозволяє досліджувати залежність між числовим коефіцієнтом **alpha** і властивостями згенерованого тексту.

2.6 Обґрунтування вибору підходу

Для дипломного проєкту обрано LoRA-адаптери як компроміс між якістю, відтворюваністю та обчислювальною доступністю. Prompt engineering занадто залежить від формулювання запиту. Повне донавчання є дорогим і погано підходить для кількох окремих стилів. RLHF потребує складнішого циклу навчання. Activation steering є перспективним, але вимагає більш глибокої інтеграції в механіку моделі.

LoRA дає змогу навчити чотири окремі адаптери для чотирьох режимів, а потім використовувати їх як керовані параметричні напрями. [2, 8] Це відповідає меті роботи: реалізувати практичну систему, у якій стиль відповіді змінюється через параметри моделі, а не лише через промпт.

Вибір LoRA також обґрунтовується тим, що цей підхід добре поєднується з експериментальною природою роботи. Для кожного адаптера можна окремо підготувати навчальні приклади, провести навчання, зберегти чекпойнти, запустити інференс і проаналізувати результати. Якщо певний адаптер працює гірше, його можна перевчити або покращити дані саме для цього стилю, не змінюючи інші адаптери.

З погляду програмної реалізації LoRA є зручною, оскільки підтримується бібліотекою PEFT і може інтегруватися з моделями, завантаженими через transformers. Це зменшує обсяг низькорівневого коду і дозволяє зосередитися на експериментальній частині: підготовці даних, умовах керування, оцінюванні та інтерфейсі перегляду результатів.

Таким чином, обраний підхід відповідає кільком вимогам одночасно: він є достатньо легким для доступного GPU, достатньо гнучким для кількох стилів, достатньо модульним для окремого збереження адаптерів і достатньо формалізованим для аналізу впливу коефіцієнта α .

Висновки до розділу

У другому розділі було проаналізовано основні підходи до керування поведінкою великих мовних моделей. Розглянуто prompt engineering, повне донавчання, RLHF та інструкційне вирівнювання, LoRA/PEFT, activation steering і parameter steering.

Prompt engineering є найпростішим способом керування, однак його недоліками є нестабільність і залежність від формулювання інструкції. Повне донавчання моделі може суттєво змінювати її поведінку, але потребує значних обчислювальних ресурсів і не є оптимальним для задачі швидкого перемикавання між кількома стилями. RLHF та інструкційне вирівнювання більше підходять для глобального налаштування поведінки моделі, ніж для окремого параметричного керування конкретними властивостями відповіді.

На основі аналізу обрано LoRA-адаптери як основний підхід дипломної роботи. Вони дозволяють залишати базову модель замороженою, навчати невеликі додаткові параметри, зберігати окремі адаптери для окремих стилів і комбінувати їх під час інференсу. Саме ці властивості відповідають меті роботи та обмеженням доступної обчислювальної інфраструктури.

3 ТЕХНОЛОГІЧНИЙ СТЕК І ЗАСОБИ РОЗРОБКИ

3.1 Мова програмування та середовище виконання

Основна частина системи реалізована мовою Python. Вона використовується для підготовки даних, запуску OpenAI Batch pipeline, навчання LoRA-адаптерів, інференсу, автоматичного оцінювання та експорту результатів.

Проект оформлено як Python-пакет **lm-steer-thesis**. Основні залежності описані у **pyproject.toml**, а повторно використовувана логіка винесена в модуль **src/lm_steer_thesis**. Така структура зменшує дублювання між скриптами та дозволяє запускати окремі етапи pipeline незалежно.

3.2 Бібліотеки для роботи з мовними моделями

Для роботи з моделлю та адаптерами використано такі бібліотеки:

- **torch** для виконання обчислень на GPU;
- **transformers** для завантаження моделі **google/gemma-3-4b-it**, токенизатора та генерації тексту; [9]
- **peft** для створення, навчання, завантаження та композиції LoRA-адаптерів; [8]
- **accelerate** для зручнішого розміщення моделі на доступних пристроях;
- **bitsandbytes** як інструмент для квантизації у конфігураціях, де це потрібно. [3]

У фінальному експерименті для Gemma 3 4B використано bfloat16 без 4-bit квантизації. [5 - 7] Для майбутнього масштабування у проєкті також підготовлено конфігурацію Gemma 3 12B з 4-bit режимом, але ці результати не включаються як реалізована частина звіту.

3.3 Підготовка та зберігання даних

Для обробки даних використано **pandas**, **pyarrow**, **datasets**, **langdetect**, **numpy** і допоміжні Python-скрипти. Основні табличні артефакти зберігаються у форматі **parquet**, оскільки він добре підходить для проміжних наборів даних із великою кількістю текстових рядків і метаданих.

Початковий банк промптів містить 1200 запитів: 600 англійською мовою та 600 українською мовою. Початковий розподіл даних становив 1000 train-промптів і 200 eval-промптів. Після очищення сформовано актуальний **clean subset**, який містить 836 **clean prompts** і 3344 **styled rows**. Мовний склад **clean subset** становить 520 англійських і 316 українських промптів. Для кожного з чотирьох адаптерів використовується 713 train-прикладів і 123 eval-приклади.

Таке очищення є необхідним, оскільки не кожен промпт однаково придатний для стильового керування. Наприклад, дуже короткі фактичні питання, задачі на переклад, програмування або математичні обчислення мають низьку стильову гнучкість. У таких випадках стильова зміна або майже непомітна, або може пошкодити правильність відповіді. Тому для навчання адаптерів доцільніше використовувати відкриті запити, які допускають кілька природних варіантів відповіді.

Формат **parquet** використовується для основних проміжних таблиць, тому що він дозволяє ефективно зберігати текстові дані разом із метаданими: мовою, **split**, атрибутом, текстом промпта, нейтральною відповіддю та стилізованою відповіддю. Формати **json** і **csv** використовуються там, де потрібна зручність експорту, перегляду або інтеграції з вебінтерфейсом.

3.4 Генерація еталонних відповідей

Еталонні відповіді для навчання адаптерів створено через **OpenAI Batch API**. [10] Pipeline складається з двох основних етапів:

1. Генерація нейтральної відповіді для кожного очищеного промпта.
2. Переписування нейтральної відповіді у стилі **formal, informal, concise і verbose**.

Batch API обрано тому, що він зручний для асинхронної обробки великої кількості запитів. [10] Результати зберігаються у .jsonl, після чого парсяться в parquet. Для підвищення якості передбачено гераір pipeline, який повторно генерує відсутні або проблемні приклади.

Використання двоетапної схеми має важливе методичне значення. Якщо одразу генерувати formal, informal, concise і verbose відповіді без спільного нейтрального референсу, то різні стилі можуть відрізнятися не лише формою, а й змістом. У такому випадку складно визначити, чи адаптер навчився стилю, чи просто отримав різні за змістом приклади. Нейтральна відповідь виконує роль змістової основи, від якої будуються всі стильові варіанти.

На етапі переписування нейтральної відповіді кожен стиль має свою ціль. Formal має зробити відповідь більш офіційною та стриманою. Informal має зменшити офіційність і зробити текст ближчим до неформального пояснення. Concise має скоротити відповідь, залишивши головну думку. Verbose має розширити пояснення, але не додати новий несумісний зміст. Саме така постановка дозволяє навчати адаптери як напрями зміни форми відповіді.

Після генерації еталонних відповідей важливо не використовувати всі приклади без перевірки. Навіть якісна модель-генератор може створити відповідь неправильною мовою, дати відмову, надто сильно змінити зміст або не виконати вимогу до довжини. Тому cleaning target-відповідей є не менш важливим, ніж cleaning початкових промптів. У фінальному наборі залишаються лише ті промпти, для яких усі чотири стильові варіанти пройшли перевірку.

3.5 Обчислювальна інфраструктура

Для навчання і запуску моделі використовується RunPod із GPU A40 48 GB. [12] Такий GPU достатній для експериментів із Gemma 3 4B і LoRA-адаптерами. Рекомендована конфігурація сховища для цього проєкту:

- container disk: **50 GB**;
- volume disk: мінімум **150 GB**;
- volume disk: **200 GB**, якщо потрібно кешувати модельні чекпойнти, адаптери, датасети та всі проміжні артефакти.

Таке розділення корисне, оскільки container disk використовується для середовища, залежностей і тимчасових файлів контейнера, а volume disk зберігає артефакти між перезапусками pod.

3.6 Вебзастосунок для демонстрації

Для демонстрації результатів створено статичний вебзастосунок на Next.js. [13] Він не запускає модель і не виконує Python-код під час перегляду. Застосунок читає попередньо підготовлені JSON-файли з **web/public/data**, графіки з **web/public/plots** і файли для завантаження з **web/public/downloads**.

Головне призначення вебінтерфейсу — дати змогу швидко переглядати:

- промпти українською та англійською мовами;
- базову відповідь і відповіді з керуванням;
- активну умову steering;
- метрики якості;
- графіки залежності формальності та довжини від коефіцієнта alpha;
- результати ручної оцінки.

Таким чином, вебзастосунок є демонстраційним шаром над експериментальними артефактами, а не окремою production-системою.

Такий підхід має кілька переваг. По-перше, вебзастосунок не залежить від доступності GPU, тому його можна запускати окремо від середовища навчання. По-друге, результати залишаються стабільними: під час демонстрації не відбувається нової генерації, тому користувач переглядає саме ті відповіді, які були оцінені в експерименті. По-третє, статичний формат спрощує розгортання і дозволяє використовувати вебінтерфейс як зручний інструмент для аналізу.

Вебзастосунок також виконує роль перевірки повноти pipeline. Якщо підготовлені JSON-файли, графіки й таблиці коректно відображаються у Results Explorer, це означає, що етапи генерації, оцінювання та експорту узгоджені між собою. У дипломній роботі це важливо, оскільки результати мають бути не лише обчислені, а й представлені у зрозумілій формі для аналізу та демонстрації.

Висновки до розділу

У третьому розділі було описано технологічний стек і засоби розробки системи. Основна частина pipeline реалізована мовою Python, оскільки вона має розвинену екосистему для роботи з даними, великими мовними моделями, навчанням і автоматичним оцінюванням. Проєкт оформлено як Python-пакет, що полегшує повторне використання коду та зменшує дублювання між скриптами.

Для роботи з моделлю використано бібліотеки torch, transformers, peft, accelerate та інші допоміжні інструменти. Для підготовки й зберігання даних застосовуються pandas, pyarrow, datasets, langdetect, parquet, json і csv. Такий набір технологій відповідає задачі побудови відтворюваного експериментального pipeline.

Також було описано використання OpenAI Batch API для генерації нейтральних і стилізованих відповідей, обчислювальну інфраструктуру на базі RunPod A40 48 GB та статичний вебзастосунок Results Explorer на Next.js, React і TypeScript. У результаті обраний стек забезпечує повний цикл роботи: від підготовки даних до демонстрації фінальних результатів.

4 АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Загальна архітектура pipeline

Система побудована як послідовний pipeline. Кожен етап створює артефакти, які використовуються наступним етапом. Це спрощує відтворення експерименту та дозволяє окремо перевіряти дані, навчання, інференс і оцінювання.

Загальна логіка pipeline має такий вигляд:

Промпти UA/EN → Очищення промπτів → Нейтральні відповіді → Стилізовані відповіді → Clean training subset → Навчання LoRA-адаптерів → Інференс: baseline, alpha, composition → Автоматичне оцінювання → Графіки та CSV/Parquet-артефакти → Вебзастосунок Results Explorer.

Архітектура побудована так, щоб мінімізувати залежність між етапами. Вхідні промпти не передаються безпосередньо в навчання, а спочатку проходять очищення. Згенеровані відповіді не використовуються без перевірки, а проходять target cleaning. Навчені адаптери не оцінюються безпосередньо, а спочатку застосовуються для генерації відповідей у стандартизованих умовах. Лише після цього отримані тексти проходять автоматичне оцінювання.

Кожен етап має власний результат. Підготовка даних створює clean prompts і styled targets. Навчання створює папки адаптерів formal, informal, concise і verbose. Інференс створює generations.parquet. Оцінювання створює scored.parquet і scoring_summary.json. Експорт створює viewer-prompts.json, viewer-validation.json, viewer-meta.json, графіки та файли для завантаження. Така організація дозволяє аналізувати помилки на конкретному етапі, а не в усій системі одразу.

Pipeline також відображає дослідницьку логіку роботи. Спочатку створюється контрольований набір даних, потім навчаються окремі стильові напрями, після цього перевіряється, як ці напрями поведуться при різних значеннях alpha та при композиції. На завершальному етапі результати подаються у вигляді

таблиць, графіків і вебінтерфейсу, що дозволяє поєднати автоматичну та ручну оцінку.

4.2 Підготовка двомовного набору даних

Початковий банк промптів містить **1200** рядків. Дані збалансовані за мовою: **600** промптів англійською та **600** промптів українською. Початковий розподіл за split становить **1000** train і **200** eval.

Промпти проходять очищення. Відхиляються занадто короткі або занадто довгі запити, багатоблокові приклади, задачі на переклад, програмування, математику, виправлення граматики, заповнення пропусків, yes/no питання та інші формати з низькою стильовою гнучкістю. Також перевіряється відповідність мови: англійські промпти мають бути переважно латинкою, українські — кирилицею.

Таблиця 4.1 – clean subset

Показник	Значення
Clean-промпти	836
Стилізовані приклади	3344
Train-промпти	713
Eval-промпти	123
Мовний склад	520 EN / 316 UA
Атрибути	formal, informal, concise, verbose

Очищення промптів виконує роль першого фільтра якості. На цьому етапі система відкидає приклади, які не підходять для навчання стильового керування. Занадто короткі запити часто не мають достатнього контексту для осмисленої відповіді. Занадто довгі запити можуть містити кілька задач одночасно або

вимагати спеціальної обробки. Запити на переклад, програмування, математику, виправлення граматики чи заповнення пропусків також не є оптимальними, оскільки в них стильова зміна може заважати виконанню основної задачі.

Окремо перевіряється мова промпта. Для англійських прикладів очікується переважно латинська писемність, для українських — кирилична. Це потрібно для того, щоб модель навчалась відповідати тією ж мовою, якою сформульовано запит. У двомовній системі помилка мови є суттєвою, оскільки навіть змістовно правильна відповідь може бути непридатною для користувача, якщо вона згенерована не тією мовою.

Сформований `clean subset` використовується як основа для навчання всіх чотирьох адаптерів. Це означає, що адаптери порівнюються на однаковій базі промπτів, але з різними `target_text` для кожного атрибута. Завдяки цьому різницю між адаптерами можна пов'язувати саме зі стильовим навчанням, а не з різницею у вхідних даних.

4.3 Формування стилізованих відповідей

Для кожного промпта спочатку генерується нейтральна відповідь. Далі вона переписується у чотири стилізовані варіанти:

- `formal` — офіційніша відповідь;
- `informal` — менш офіційна відповідь;
- `concise` — коротша відповідь;
- `verbose` — розгорнутіша відповідь.

Така двоетапна схема важлива, бо всі стилі походять від одного нейтрального змісту. Це полегшує контроль семантичної стабільності: стилізована відповідь має змінювати форму, а не основний зміст. Якщо для кожного стилю генерувати відповідь незалежно, то різниця між варіантами може бути зумовлена не стилем, а випадковою зміною змісту. Нейтральна відповідь виконує роль спільної основи для всіх стилізованих `target`-відповідей.

Formal-варіант має бути більш офіційним, структурованим і придатним для ділової або академічної комунікації. Informal-варіант має бути менш офіційним, але не повинен втрачати коректність або переходити у надмірно розмовний стиль. Concise-варіант має стисло передавати головну думку без другорядних деталей. Verbose-варіант має розширювати пояснення, додаючи контекст або деталізацію, але не змінюючи основного сенсу.

Згенеровані відповіді також очищуються. Відхиляються порожні відповіді, відмови, відповіді неправильною мовою, невдалі concise/verbose співвідношення довжини та приклади із семантичним зсувом. Для concise очікується приблизно 50%–75% довжини нейтрального тексту, для verbose — приблизно 130%–180%.

Target cleaning потрібен для того, щоб адаптер навчався на якісних прикладах. Якщо у навчальний набір потрапить відповідь неправильною мовою, адаптер може засвоїти небажану поведінку. Якщо verbose-відповідь буде занадто короткою або concise-відповідь буде майже такою ж довгою, як нейтральна, модель не отримає чіткого сигналу для навчання. Якщо formal або informal відповідь змінить зміст, адаптер може почати змінювати не тільки стиль, а й фактичну частину відповіді.

Таким чином, якість підготовлених target-відповідей безпосередньо впливає на якість навчених LoRA-адаптерів. У цій роботі очищення розглядається як окремий важливий етап, а не як допоміжна технічна операція.

4.4 Навчання LoRA-адаптерів

Для навчання використовується модель google/gemma-3-4b-it. [5 – 7] Базові параметри моделі залишаються замороженими, а оновлюються тільки LoRA-параметри. [2, 8] Для кожного атрибута навчається окремий адаптер.

Таблиця 4.2 – Основні параметри навчання

Параметр	Значення
Модель	google/gemma-3-4b-it
Тип обчислень	bfloat16
LoRA rank	16
LoRA alpha	32
LoRA dropout	0.05
Кількість епох	2
Max sequence length	1024
Train batch size per device	1
Gradient accumulation steps	4

LoRA застосовується до модулів:

q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj

Вибір модулів охоплює як attention-проекції, так і MLP-проекції. Це дозволяє адаптеру впливати не лише на механізм уваги, а й на внутрішні перетворення в шарах моделі. При цьому базова модель не змінюється повністю, що зменшує обсяг навчання і дозволяє зберігати адаптери як окремі артефакти.

Тренувальний приклад формується як chat-повідомлення:

user: prompt

assistant: target_text

Такий формат відповідає інструкційному способу використання моделі. Модель отримує запит користувача і повинна сформулювати відповідь у потрібному стилі. Для formal-адаптера target_text містить формальну відповідь, для informal — неформальну, для concise — стисло, для verbose — розгорнуто. Таким чином, кожен адаптер навчається на власному підмножині рядків, але структура прикладу залишається однаковою.

Після навчання зберігаються чотири папки адаптерів:

- formal/
- informal/
- concise/
- verbose/

Окреме збереження адаптерів дає змогу підключати їх незалежно. Це важливо для подальшого інференсу, оскільки система має перевіряти різні режими: baseline без адаптера, одиночний адаптер, масштабування адаптера та композицію двох адаптерів. Якщо адаптери зберігаються окремо, їх можна комбінувати без повторного навчання.

4.5 Інференс і умови керування

Під час інференсу базова модель завантажується один раз, після чого до неї підключаються навчені адаптери. Для кожного промпта система генерує відповіді за кількома типами умов.

Таблиця 4.3 – Підтримувані типи умов:

Тип умови	Опис
baseline	модель без активного адаптера
discrete	один адаптер із вагою 1
continuous	один адаптер із коефіцієнтом alpha
composition	лінійна комбінація двох адаптерів

Baseline потрібен як контрольна умова. Він показує, як модель відповідає без параметричного втручання. Без baseline неможливо визначити, чи справді адаптер змінив відповідь, оскільки не було б точки порівняння.

Discrete-режим відповідає стандартному використанню одного адаптера. У цьому режимі активується один стиль, наприклад formal або concise, і модель генерує відповідь із фіксованою силою адаптера. Цей режим дозволяє перевірити базову працездатність кожного окремого адаптера.

Continuous-режим дозволяє змінювати силу адаптера через alpha. Для неперервного керування використовуються значення:

$$\alpha \in \{-2, -1, 0, 1, 2\}$$

Значення $\alpha = 0$ відповідає базовому режиму без адаптера. Додатні значення α мають посилювати напрям, якому навчався адаптер. Негативні

значення використовуються для дослідницької перевірки того, як модель поводить себе при протилежному напрямі параметричної зміни. Такий режим є важливим, бо дозволяє побачити не лише факт зміни стилю, а й залежність цієї зміни від сили керування.

Для композицій використано чотири умови:

`formal_concise_1_1`

`formal_concise_2_2`

`informal_verbose_1_1`

`informal_verbose_2_2`

Композиція перевіряє, чи можна поєднати дві властивості відповіді. Наприклад, `formal + concise` має створювати відповідь, яка одночасно є більш офіційною і коротшою. `Informal + verbose`, навпаки, має поєднувати менш офіційний тон із більшою деталізацією. Порівняння ваг `1_1` і `2_2` дозволяє оцінити, чи посилення композиції покращує керованість або, навпаки, погіршує збереження змісту.

Фінальний запуск містить 25 умов керування, 123 eval-промпти та 3075 згенерованих відповідей. Параметри генерації: `temperature = 0.2`, `top_p = 0.9`, `max_new_tokens = 256`. Однакові параметри генерації використовуються для всіх умов, щоб різниця у відповідях була пов'язана насамперед з адаптерами, а не зі зміною `sampling`-налаштувань.

4.6 Автоматичне оцінювання результатів

Оцінювання виконується після генерації та не створює новий текст. Для кожної відповіді розраховуються такі метрики:

- `language_ok` — чи збігається визначена мова відповіді з мовою промпта;
- `word_count` — кількість слів у відповіді;

- `length_ratio` — співвідношення довжини відповіді до нейтрального референсу;
- `semantic_similarity` — семантична подібність до нейтрального референсу;
- `formality_score` — оцінка формальності;
- `invalid_output` — ознака порожньої або некоректної відповіді;
- `truncation_proxu` — евристична ознака можливої обрізаної генерації.

Автоматичне оцінювання необхідне тому, що кількість генерацій є занадто великою для повністю ручного аналізу. У фінальному експерименті отримано 3075 відповідей, тому потрібні метрики, які дозволяють швидко порівнювати умови керування. Водночас автоматичні метрики не розглядаються як абсолютна істина. Вони використовуються як інструмент попереднього аналізу, після якого окремі приклади можна переглядати вручну.

Метрика `language_ok` є базовою для двомовної системи. Якщо відповідь сформована не тією мовою, якою був промпт, то така генерація є проблемною незалежно від інших показників. Наприклад, висока семантична подібність не компенсує ситуацію, коли український промпт отримує відповідь англійською.

Метрика `length_ratio` використовується для аналізу `concise` і `verbose` адаптерів. Якщо адаптер `concise` працює правильно, очікується зменшення `length_ratio` порівняно з `baseline` або нейтральною відповіддю. Якщо адаптер `verbose` працює правильно, очікується збільшення `length_ratio`. При цьому зміна довжини має розглядатися разом із `semantic_similarity`, оскільки надто коротка відповідь може втратити зміст, а надто довга — додати зайву або неточну інформацію.

Для семантичної подібності використовується `multilingual sentence embedding model sentence-transformers/paraphrase-multilingual-mpnet-base-v2`. [4, 11] Це дає змогу однаково оцінювати українські та англійські відповіді. `Semantic similarity` потрібна для перевірки того, чи стилізована відповідь залишається

близькою до нейтрального референсу. Висока подібність означає, що модель переважно зберігає зміст, а низька може вказувати на семантичний зсув.

Formality_score використовується для аналізу formal та informal адаптерів. Очікується, що formal-адаптер підвищуватиме або зберігатиме високий рівень формальності, тоді як informal-адаптер має знижувати цей показник. Оцінка формальності не є повною заміною людського сприйняття стилю, але вона дає числовий індикатор для порівняння умов.

Invalid_output і truncation_proху використовуються як метрики технічної якості. Вони показують, чи не створила модель порожню відповідь і чи немає ознак обрізання тексту. Нульові значення цих показників у фінальних результатах є позитивним сигналом, оскільки означають, що pipeline не породив очевидно некоректних або технічно зламаних генерацій.

4.7 Експорт артефактів для вебінтерфейсу

Після оцінювання результати експортуються у формат, придатний для статичного вебзастосунку. [13] Основні файли:

- web/public/data/viewer-prompts.json
- web/public/data/viewer-meta.json
- web/public/data/viewer-validation.json
- web/public/data/manual-reviews.json
- web/public/plots/en_formality_score_vs_alpha.png
- web/public/plots/en_length_ratio_vs_alpha.png
- web/public/plots/ua_formality_score_vs_alpha.png
- web/public/plots/ua_length_ratio_vs_alpha.png
- web/public/downloads/composition_summary.csv
- web/public/downloads/gemma-3-4b-final.csv
- web/public/downloads/scored.parquet

Це дозволяє переглядати результати без повторного запуску моделі або Python-скриптів.

Експорт артефактів також відокремлює дослідницьку частину від демонстраційної. Модель і адаптери потрібні для генерації, але після завершення експерименту достатньо мати готові файли з результатами. Це робить демонстрацію стабільнішою: під час перегляду у вебінтерфейсі не змінюються відповіді, не витрачається GPU-час і не виникають випадкові відмінності через повторну генерацію.

Файли `viewer-prompts.json`, `viewer-validation.json` і `viewer-meta.json` виконують різні ролі. Перший містить інформацію про промпти, другий — результати та метрики, третій — допоміжні метадані для відображення. Графіки у `web/public/plots` дають можливість швидко побачити залежності `formality_score` і `length_ratio` від `alpha`. Файли у `web/public/downloads` потрібні для того, щоб результати можна було додатково аналізувати поза вебінтерфейсом.

Висновки до розділу

У четвертому розділі було описано архітектуру та реалізацію системи. Система побудована як послідовний `pipeline`, у якому кожен етап створює артефакти для наступного етапу. Така структура забезпечує відтворюваність, модульність і можливість окремого аналізу кожної частини роботи.

Було розглянуто підготовку двомовного набору даних, формування нейтральних і стилізованих відповідей, очищення `target`-відповідей, навчання LoRA-адаптерів, інференс з різними умовами керування, автоматичне оцінювання результатів та експорт артефактів для вебінтерфейсу.

Особливу увагу приділено тому, що базова модель залишається замороженою, а для кожного стилю навчається окремий адаптер. Під час інференсу ці адаптери можуть використовуватися окремо, масштабуватися через `alpha` або

комбінуватися. Це дозволяє дослідити параметричне керування відповідями моделі без повного донавчання.

5 РОБОТА СИСТЕМИ ТА ІНТЕРФЕЙС

5.1 Сценарій використання системи

Робота системи складається з двох режимів: експериментального pipeline та демонстраційного перегляду результатів. Pipeline запускається розробником і створює дані, адаптери, генерації, оцінки та графіки. Вебінтерфейс використовується для аналізу вже готових результатів.

Типовий сценарій роботи:

1. Підготувати або оновити банк промптів.
2. Згенерувати нейтральні та стилізовані відповіді.
3. Очистити дані та сформувати clean subset.
4. Навчити LoRA-адаптери для чотирьох стилів.
5. Запустити інференс для baseline, alpha-умов і композицій.
6. Оцінити відповіді автоматичними метриками.
7. Експортувати результати у вебзастосунок.
8. Переглянути приклади, графіки та ручні оцінки у Results Explorer.

З точки зору користувача вебінтерфейсу, основний сценарій є простішим. Користувач не взаємодіє з навчанням моделі та не запускає інференс. Він відкриває Results Explorer, обирає мову, промпт і умову керування, після чого переглядає згенеровану відповідь і відповідні метрики. Такий поділ ролей є важливим: розробник працює з pipeline, а користувач або дослідник аналізує готові результати.

У межах дипломної роботи вебінтерфейс виконує роль інструмента валідації результатів. Автоматичні метрики показують загальні закономірності, але окремі приклади дозволяють краще зрозуміти, як саме адаптер змінив відповідь. Наприклад, можна перевірити, чи formal-адаптер справді робить відповідь офіційнішою, чи concise-адаптер не видаляє важливі деталі, чи composition не створює неприродний текст.

Окремою перевагою є те, що результати можна переглядати без повторного запуску моделі. Це знижує технічні вимоги до демонстрації і дозволяє показувати роботу системи навіть тоді, коли GPU-середовище недоступне. Для дипломної роботи це особливо зручно, оскільки під час захисту важливо швидко показати приклади, графіки та підсумкові результати.

5.2 Головна сторінка вебзастосунку Results Explorer

Вебзастосунок Results Explorer є основним інтерфейсом демонстрації. Він показує підсумки експерименту, дає змогу обрати мову, промпт, сімейство керування та коефіцієнт α , а також порівняти відповідь із нейтральним референсом. На рисунку 6.1 наведено головну сторінку вебзастосунку з вибором промпта, шкалою α , перемикачами стилю та блоком згенерованої відповіді.

Головна сторінка побудована так, щоб користувач міг швидко перейти від загального огляду до конкретної генерації. У верхній частині інтерфейсу розташовані елементи вибору мови та промпта. Це дозволяє окремо аналізувати українські й англійські приклади. Далі користувач може обрати режим керування: базовий режим, один із стильових адаптерів або композицію. Для режимів з одиночним адаптером доступний вибір α .

Центральним елементом сторінки є блок Output, у якому відображається згенерована відповідь. Саме цей блок дозволяє якісно оцінити результат: чи відповідь природна, чи зберегла зміст, чи змінився стиль, чи не з'явилися зайві або повторювані фрагменти. Поруч або нижче розташовуються метрики, які дають кількісне пояснення поточної генерації.

Візуальний інтерфейс має не замінювати автоматичне оцінювання, а доповнювати його. Наприклад, якщо `formality_score` показує зниження формальності для `informal`-адаптера, користувач може відкрити конкретну відповідь і перевірити, чи це зниження справді виглядає природно. Якщо

semantic_similarity нижча для певної композиції, можна переглянути текст і визначити, чи відбулася втрата змісту.

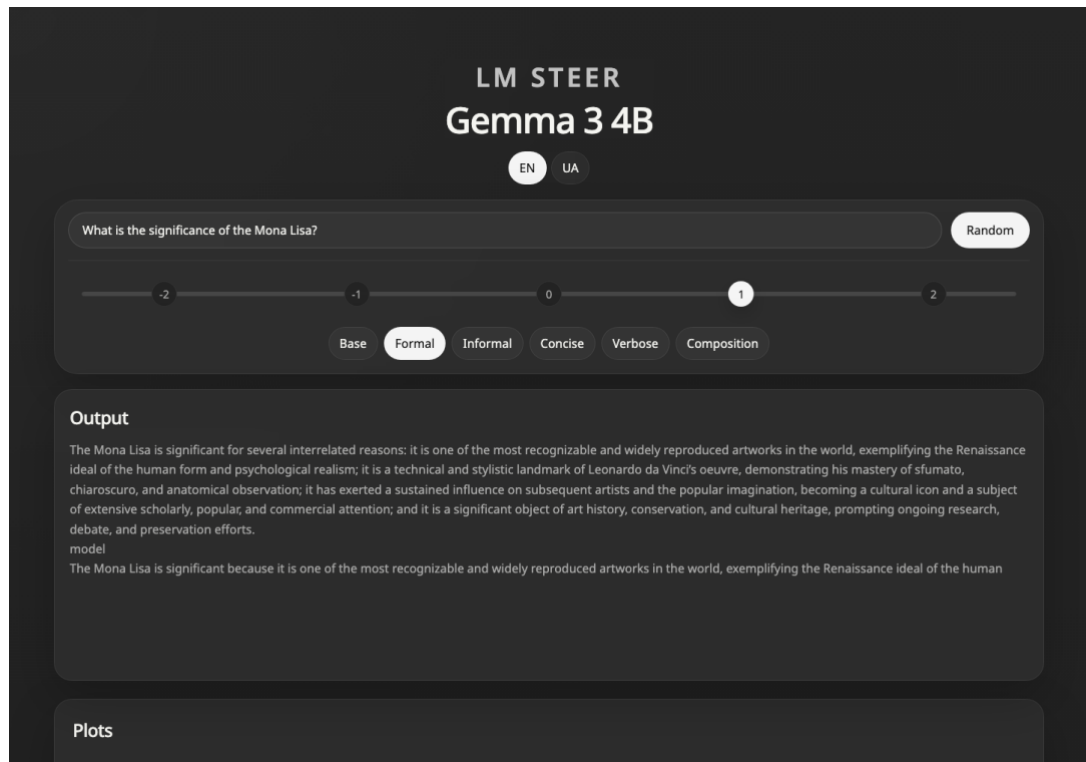


Рисунок 6.1 — Головна сторінка вебзастосунку Results Explorer.

5.3 Перегляд промптів і умов керування

Інтерфейс дозволяє перемикатися між англійськими та українськими промптами. Для кожного промпта доступні:

- текст початкового запиту;
- базова відповідь без адаптера;
- відповідь для вибраної steering-умови;
- нейтральний референс;
- значення метрик для поточної генерації.

Користувач може переглядати сімейства **formal**, **informal**, **concise**, **verbose** і **composition**. Для одиночних адаптерів доступний вибір **alpha**. Для композицій доступні пари **formal** + **concise** та **informal** + **verbose**.

Цей інтерфейс важливий для дипломної роботи, оскільки автоматичні метрики не завжди повністю пояснюють якість відповіді. Перегляд конкретних прикладів дозволяє побачити, чи справді модель змінила стиль, чи зберегла зміст і чи не погіршила природність тексту.

Наприклад, для одного промпта можна порівняти baseline-відповідь із formal-відповіддю. Якщо formal-відповідь має більш офіційну лексику, але передає ту саму думку, то адаптер працює очікувано. Якщо ж відповідь стала формальною, але втратила важливу інформацію, це означає, що одного formality_score недостатньо для позитивної оцінки. Аналогічно, concise-відповідь може мати бажану меншу довжину, але її потрібно перевіряти на збереження ключового змісту.

Для композицій ручний перегляд є ще важливішим. Коли одночасно активуються два адаптери, модель може реагувати складніше, ніж у випадку одного адаптера. Відповідь може справді поєднати дві властивості, а може стати менш стабільною. Тому вебінтерфейс дає змогу швидко знайти приклади для певної composition-умови та порівняти їх із baseline або single-adapter режимом.

Інтерфейс також корисний для аналізу відмінностей між мовами. Одна й та сама умова керування може мати різний ефект для українських і англійських промптів. Через перемикач мови можна окремо переглядати результати та оцінювати, чи зберігається загальна закономірність для обох мов.

5.4 Візуалізація метрик

У вебзастосунку використовуються графіки, згенеровані на етапі оцінювання. Вони показують, як змінюються формальність і довжина відповіді залежно від коефіцієнта **alpha**.

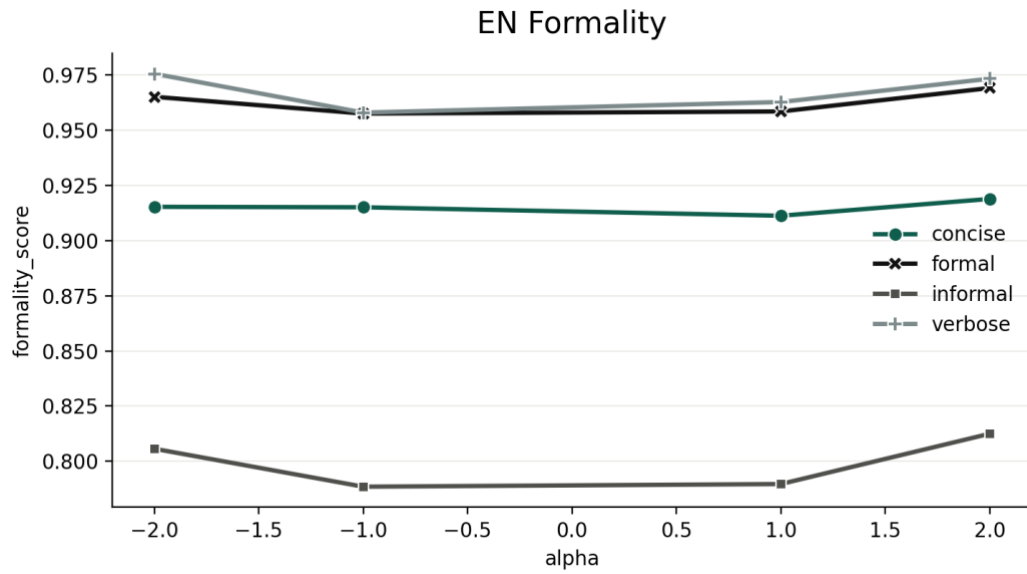


Рисунок 6.2 — Залежність оцінки формальності від **alpha** для англійської мови.

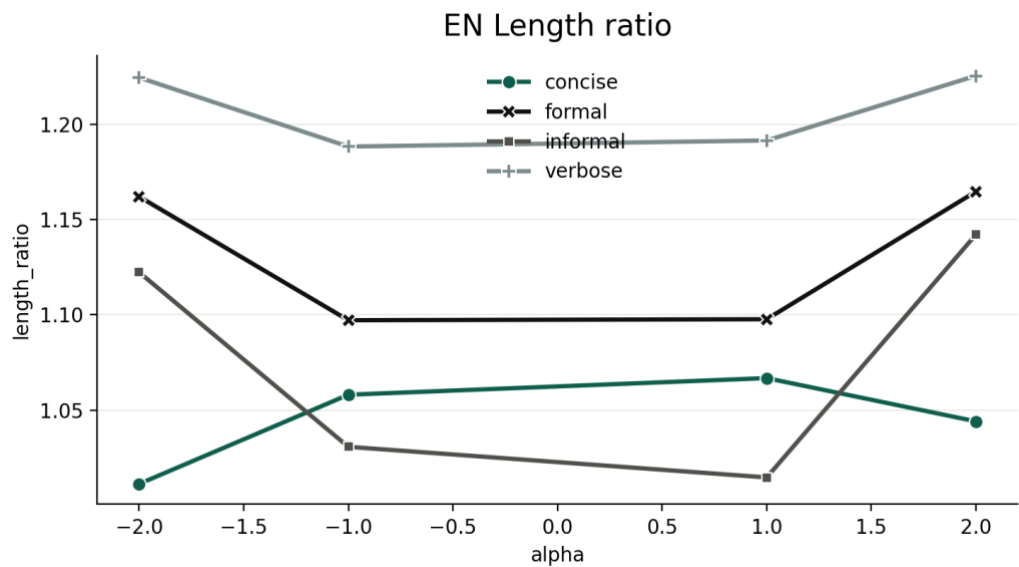


Рисунок 6.3 — Залежність відношення довжини відповіді від **alpha** для англійської мови.

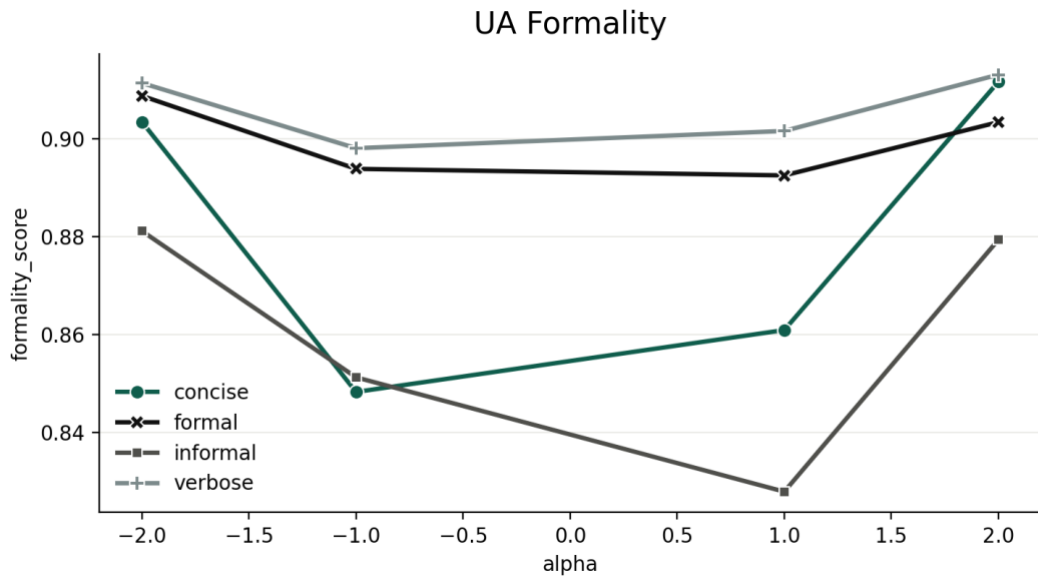


Рисунок 6.4 — Залежність оцінки формальності від **alpha** для української мови.

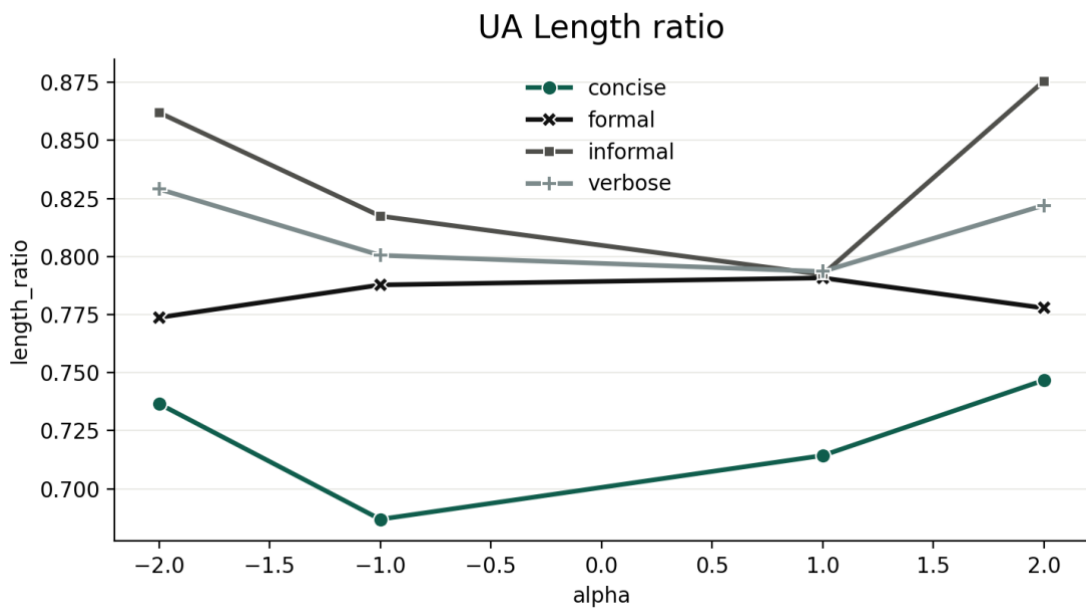


Рисунок 6.5 — Залежність відношення довжини відповіді від **alpha** для української мови.

Ці графіки використовуються для аналізу того, чи має параметричне керування передбачуваний ефект. Для стилів *concise* і *verbose* основною метрикою є *length_ratio*. Для стилів *formal* і *informal* важливішою є *formality_score*.

Графіки залежності від *alpha* дозволяють оцінити не лише результат окремої генерації, а й загальну тенденцію. Якщо збільшення *alpha* для певного адаптера стабільно змінює відповідну метрику в очікуваному напрямі, це свідчить про

наявність керованого ефекту. Якщо ж метрика змінюється хаотично або не реагує на α , це може означати, що адаптер недостатньо добре навчився потрібної властивості або що метрика не повністю відображає зміну стилю.

Для `length_ratio` важливо аналізувати не абсолютну довжину відповіді, а співвідношення з нейтральним референсом. Це робить порівняння справедливішим, оскільки різні промпти природно потребують відповідей різної довжини. Наприклад, короткий запит може мати коротку нейтральну відповідь, а складний запит — довшу. `Length_ratio` дозволяє нормалізувати це порівняння.

Для `formality_score` важливо враховувати, що формальність є більш суб'єктивною властивістю, ніж довжина. Довжину можна порахувати безпосередньо, а формальність оцінюється через модель або класифікатор. Тому результати `formality_score` потрібно інтерпретувати разом із прикладами відповідей і ручною перевіркою.

Окремо аналізуються українські та англійські графіки. Це потрібно тому, що мовна модель і метрики можуть мати різну чутливість до стильових змін у різних мовах. Якщо для обох мов спостерігається схожий напрям зміни, це посилює довіру до висновку про працездатність параметричного керування. Якщо ефект відрізняється, це вказує на необхідність додаткового аналізу або розширення навчального набору для відповідної мови.

5.5 Ручна оцінка відповідей

Окрім автоматичних метрик, система підтримує ручну оцінку відповідей. Для цього підготовлено експорт у CSV та JSON-файли для вебінтерфейсу. Ручна перевірка корисна для виявлення випадків, які складно оцінити автоматично: неприродний стиль, часткова втрата сенсу, надмірна формальність або погана відповідність промπτу.

У вебінтерфейсі ручні оцінки показуються як додатковий шар контролю якості. Вони не замінюють автоматичні метрики, але допомагають інтерпретувати результати там, де числова оцінка не дає повної картини.

Ручна оцінка особливо корисна для стилістичних властивостей. Наприклад, дві відповіді можуть мати схожий `formality_score`, але одна з них виглядатиме природніше для людини. Так само `concise`-відповідь може мати правильний `length_ratio`, але бути надто сухою або втратити важливий контекст. У таких випадках людський перегляд дозволяє краще зрозуміти реальну якість генерації.

У межах цієї роботи ручна оцінка розглядається як допоміжний інструмент, який доповнює автоматичний аналіз. Вона потрібна не для того, щоб повністю замінити метрики, а для перевірки найбільш показових або сумнівних прикладів. Такий підхід є практичним, оскільки повна ручна оцінка всіх 3075 генерацій потребувала б значно більше часу, тоді як вибірковий перегляд дозволяє швидко виявляти характерні проблеми.

Висновки до розділу

У п'ятому розділі було описано роботу системи та вебінтерфейс Results Explorer. Показано, що система має два рівні використання: експериментальний pipeline для розробника та демонстраційний вебінтерфейс для перегляду готових результатів.

Було розглянуто сценарій використання системи, головну сторінку вебзастосунку, можливість перегляду промптів і умов керування, візуалізацію метрик і підтримку ручної оцінки відповідей. Results Explorer дозволяє аналізувати як окремі приклади генерації, так і загальні закономірності через графіки.

Вебінтерфейс є важливою частиною дипломної роботи, оскільки він робить результати експерименту зрозумілими та доступними для перегляду без повторного запуску моделі. Це дозволяє поєднати автоматичне оцінювання з якісним аналізом конкретних відповідей.

6 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОЇ ПЕРЕВІРКИ

6.1 Підсумкові результати експериментів

Фінальний експеримент виконано для моделі **google/gemma-3-4b-it**. [5 – 7] Було використано чотири LoRA-адаптери: **formal**, **informal**, **concise** і **verbose**. Для оцінювання сформовано **3075** генерацій на основі **123** eval-промптів і **25** умов керування.

Таблиця 6.1 – Загальні підсумкові метрики:

Метрика	Значення
Кількість генерацій	3075
Кількість eval-промптів	123
Кількість умов керування	25
language_ok_rate	0.985
Середня semantic_similarity	0.814
invalid_output_rate	0.0

Ці результати показують, що система здебільшого зберігає правильну мову відповіді, не створює порожніх генерацій і в середньому зберігає високу семантичну подібність до нейтрального референсу.

Показник `language_ok_rate = 0.985` означає, що більшість відповідей згенерована тією мовою, яка очікувалася за промптом. Для двомовної системи це є одним із головних критеріїв якості, оскільки стиліове керування не повинно призводити до мовного перемикання. Високе значення цієї метрики свідчить про те, що використані адаптери переважно не руйнують мовну стабільність моделі.

Середня `semantic_similarity` = 0.814 показує, що відповіді в середньому залишаються близькими до нейтрального референсу. Це важливо, оскільки ціль роботи полягає не в генерації довільно інших відповідей, а в зміні форми подання того самого змісту. Значення `semantic_similarity` не слід трактувати як абсолютну гарантію збереження змісту для кожного окремого прикладу, але воно є корисним узагальненим показником для порівняння умов.

Показник `invalid_output_rate` = 0.0 означає, що в оціненому наборі не зафіксовано порожніх або очевидно некоректних генерацій за цією ознакою. Це є позитивним технічним результатом `pipeline`, оскільки свідчить про стабільність запуску інференсу та обробки відповідей.

Важливо, що результати оцінюються не лише за середніми значеннями, а й через порівняння окремих умов керування. Середнє значення може приховувати відмінності між адаптерами, мовами або композиціями. Тому подальший аналіз розглядає, як саме змінюються показники для різних значень `alpha` та для композицій адаптерів.

6.2 Інтерпретація впливу коефіцієнтів керування

Порівняння умов керування показує, що LoRA-адаптери можуть змінювати властивості відповіді без повного донавчання базової моделі. [2, 8] Найважливішим результатом є те, що керування можна виконувати не тільки дискретно, а й через числову вагу адаптера.

Для одиночних адаптерів значення **alpha** дає змогу перевірити, як змінюється поведінка моделі при посиленні або зміні напрямку адаптера. Базовий режим відповідає **alpha** = 0. Умови з **alpha** = 1 близькі до стандартної сили адаптера, а **alpha** = 2 посилює вплив.

Композиції демонструють додатковий ефект. Для пар **formal** + **concise** і **informal** + **verbose** слабша композиція **1_1** зберігає семантику краще, ніж сильніша композиція **2_2**.

Таблиця 6.2 – Композиція пар

Мова	Умова	Semantic similarity	Length ratio	Language OK
EN	formal_concise_1_1	0.844	1.116	1.000
EN	formal_concise_2_2	0.693	1.050	0.795
EN	informal_verbose_1_1	0.854	1.147	1.000
EN	informal_verbose_2_2	0.732	1.217	0.936
UA	formal_concise_1_1	0.771	0.746	1.000
UA	formal_concise_2_2	0.544	0.889	0.956
UA	informal_verbose_1_1	0.809	0.847	1.000
UA	informal_verbose_2_2	0.593	0.943	0.956

Це означає, що сильніше параметричне втручання не завжди є кращим. Воно може посилювати бажану властивість, але водночас збільшувати ризик втрати змісту або мовної стабільності. Для практичного застосування доцільно використовувати помірні значення ваги та перевіряти їх на валідаційному наборі.

Отримані результати добре узгоджуються з інтуїцією щодо параметричного керування. Якщо адаптер задає певний напрям зміни поведінки, то збільшення його ваги має посилювати цей напрям. Проте модель є складною нелінійною системою, тому надто велика вага може призводити не лише до сильнішої стильової зміни, а й до побічних ефектів. Такими побічними ефектами можуть бути зниження semantic_similarity, зменшення language_ok або погіршення природності відповіді.

Для композицій це особливо помітно. Поєднання двох адаптерів означає, що модель одночасно отримує два напрями зміни. Якщо ці напрями добре сумісні,

результат може бути корисним: наприклад, формальна і стисла відповідь. Якщо ж вплив занадто сильний, модель може втратити баланс між стилем і змістом. Тому композиції з вагою 1_1 у багатьох випадках є більш стабільними, ніж 2_2.

З практичної точки зору це означає, що параметричне керування потребує калібрування. Недостатньо просто навчити адаптер і завжди використовувати максимальну вагу. Потрібно підібрати таке значення α , яке забезпечує помітну стильову зміну, але не погіршує ключові метрики. У реальній системі це можна робити на валідаційному наборі або через поєднання автоматичних метрик і ручної перевірки.

6.3 Обмеження реалізованого підходу

Реалізована система має кілька обмежень.

По-перше, експеримент виконано тільки для Gemma 3 4B. У проєкті підготовлено конфігурацію для Gemma 3 12B, але її результати не включаються до фінального аналізу як завершений експеримент.

По-друге, Qwen2.5-7B не використовується як реалізований контрольний baseline у цьому звіті. Його доцільно додати в майбутньому для порівняння того, чи є спостережувані ефекти властивими лише Gemma, чи повторюються на іншій архітектурі.

По-третє, автоматичні метрики не повністю відображають людське сприйняття стилю. Наприклад, висока семантична подібність не гарантує, що відповідь справді звучить природно або доречно для конкретного бізнес-контексту. [4, 11] Саме тому до системи додано ручну оцінку.

По-четверте, стилізовані training targets створені синтетично. Це прискорює підготовку даних, але якість адаптерів залежить від якості згенерованих еталонних відповідей і процедур очищення.

По-п'яте, автоматична оцінка формальності та семантичної подібності має власні обмеження. Semantic similarity добре підходить для приблизного порівняння

змісту, але не завжди виявляє дрібні фактичні зміни або втрату важливих деталей. Formality_score дає числовий сигнал про тон відповіді, але людське сприйняття формальності може залежати від контексту, предметної області та очікувань користувача.

По-шосте, у роботі розглядаються лише чотири стильові напрями: formal, informal, concise і verbose. Вони покривають важливі властивості відповіді, але не вичерпують усі можливі варіанти керування. У прикладних системах можуть бути потрібні інші властивості: технічність, простота пояснення, ввічливість, емоційність, рівень деталізації або відповідність конкретному корпоративному стилю.

По-сьоме, результати залежать від якості clean subset. Якщо певна мова або певний тип промптів представлений слабше, адаптер може працювати для них менш стабільно. У цій роботі використано двомовний набір, однак баланс після очищення не є повністю рівним між англійськими та українськими прикладами. Це потрібно враховувати під час інтерпретації результатів.

Наведені обмеження не скасовують отриманих результатів, але визначають межі їх застосування. Система демонструє працездатність підходу для обраної моделі, набору даних і чотирьох властивостей, проте для ширшого практичного використання потрібні додаткові експерименти, більший набір даних і розширена ручна оцінка.

6.4 Напрями подальшого розвитку

Можливі напрями продовження роботи пов'язані з розширенням експериментальної бази, покращенням якості даних і розвитком механізмів керування.

Першим напрямом є перевірка підходу на інших базових мовних моделях. У межах цієї роботи фінальний експеримент виконано для google/gemma-3-4b-it, тому отримані результати безпосередньо стосуються саме цієї моделі та використаного

набору даних. Подальші експерименти з іншими моделями дозволили б оцінити, наскільки спостережувані закономірності є загальними для adapter-based steering, а не лише властивими конкретній архітектурі або конкретному чекпойнту.

Другим напрямом є розширення набору керованих властивостей. У роботі розглянуто formal, informal, concise і verbose, однак на практиці можуть бути корисними й інші стилі: ввічливий, технічний, навчальний, спрощений, емоційно нейтральний або орієнтований на конкретний тип користувача. Для додавання таких властивостей потрібно підготувати нові target-відповіді, очистити їх і навчити відповідні адаптери.

Третім напрямом є збільшення та покращення української частини clean subset. Оскільки робота має двомовний характер, якість українських прикладів є важливою для стабільності керування українською мовою. Розширення українського набору може покращити якість адаптерів і зробити порівняння між мовами більш збалансованим.

Четвертим напрямом є розширення human evaluation. Автоматичні метрики зручні для великої кількості відповідей, але вони не повністю відображають людське сприйняття стилю, природності та доречності. Тому в подальшому можна організувати оцінювання з кількома оцінювачами та чіткими критеріями: збереження змісту, відповідність стилю, мовна природність, корисність відповіді.

П'ятим напрямом є дослідження інших способів композиції адаптерів. У цій роботі розглянуто композиції formal + concise та informal + verbose з однаковими вагами. Надалі можна перевіряти нерівні ваги, автоматичний підбір alpha, обмеження максимальної сили композиції або адаптивний вибір ваги залежно від типу промпта.

Таким чином, подальший розвиток системи може рухатися у двох напрямках: поглиблення дослідницької частини та наближення до прикладного використання. З дослідницького боку важливо краще зрозуміти поведінку адаптерів і композицій. З прикладного боку важливо зробити керування більш стабільним, зручним і придатним для інтеграції у реальні програмні системи.

ВИСНОВКИ

У дипломному проєкті реалізовано систему LM-Steer-подібного керування стилем відповідей мовної моделі через LoRA-адаптери. Система підтримує українську та англійську мови, чотири керовані стилі, базовий режим, дискретне керування, неперервне масштабування коефіцієнта α і композицію адаптерів.

У процесі виконання роботи було проаналізовано предметну область керування поведінкою великих мовних моделей. Було розглянуто різницю між prompt engineering, повним донавчанням, RLHF, activation steering і параметрично ефективним донавчанням. На основі цього аналізу як основний підхід було обрано LoRA/PEFT, оскільки він дозволяє навчати невеликі адаптери, залишаючи базову модель незмінною.

Під час роботи підготовлено двомовний набір даних, сформовано стилізовані цільові відповіді, навчено адаптери для formal, informal, concise і verbose, виконано генерацію для 25 умов керування та проведено автоматичне оцінювання. Фінальні метрики показали високу частку мовно коректних відповідей (0.985), відсутність порожніх генерацій (0.0) і достатню середню семантичну подібність (0.814).

Було реалізовано повний pipeline: від підготовки промптів до експорту результатів у вебінтерфейс. Такий pipeline дозволяє повторно запускати окремі етапи, зберігати проміжні артефакти та аналізувати результати без повторного запуску моделі. Це підвищує відтворюваність роботи та спрощує подальший розвиток системи.

Окремим результатом є вебзастосунок Results Explorer. Він дозволяє переглядати промпти, мови, умови керування, згенеровані відповіді, автоматичні метрики, графіки та ручні оцінки. Завдяки цьому результати експерименту можна аналізувати не лише у вигляді таблиць, а й через зручний візуальний інтерфейс.

Експеримент підтвердив, що LoRA-адаптери можуть використовуватися як практичний інструмент параметричного керування стилем відповіді. [2, 8] Водночас результати композицій показують, що надмірне посилення адаптерів

може погіршувати збереження змісту. Тому для реальних застосувань важливо не тільки навчати адаптери, а й підбирати силу їхнього впливу на основі метрик та ручної перевірки.

Практична цінність роботи полягає у тому, що одна базова модель може працювати в кількох режимах відповіді без повного донавчання під кожен режим. Це може бути корисним для систем підтримки користувачів, генерації службових повідомлень, навчальних пояснень, інформаційних сервісів і програмних продуктів, де важлива зміна стилю відповіді залежно від сценарію.

Отже, мету дипломної роботи досягнуто. Розроблена система демонструє можливість параметричного керування стилем відповідей великої мовної моделі, забезпечує двомовний експеримент, підтримує чотири стильові напрями, надає автоматичне оцінювання та має вебінтерфейс для аналізу результатів. Подальший розвиток може бути спрямований на розширення набору стилів, збільшення обсягу даних, удосконалення human evaluation і дослідження нових способів композиції адаптерів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser Ł., Polosukhin I. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017. Vol. 30. P. 5998–6008.
2. Hu E.J., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang S., Wang L., Chen W. LoRA: Low-Rank Adaptation of Large Language Models. arXiv preprint, 2021. URL: <https://arxiv.org/abs/2106.09685> (дата звернення: 08.05.2026).
3. Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient Finetuning of Quantized LLMs. Advances in Neural Information Processing Systems, 2023. Vol. 36. URL: <https://arxiv.org/abs/2305.14314> (дата звернення: 08.05.2026).
4. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Hong Kong : Association for Computational Linguistics, 2019. P. 3982–3992.
5. Gemma Team. Gemma 3 Technical Report. arXiv preprint, 2025. URL: <https://arxiv.org/abs/2503.19786> (дата звернення: 08.05.2026).
6. Gemma 3 model overview. Google AI for Developers. URL: <https://ai.google.dev/gemma/docs/core> (дата звернення: 08.05.2026).
7. Gemma 3 model card. Google AI for Developers. URL: https://ai.google.dev/gemma/docs/core/model_card_3 (дата звернення: 08.05.2026).
8. LoRA. Hugging Face PEFT Documentation. URL: https://huggingface.co/docs/peft/developer_guides/lora (дата звернення: 08.05.2026).
9. Text generation. Hugging Face Transformers Documentation. URL: https://huggingface.co/docs/transformers/en/llm_tutorial (дата звернення: 08.05.2026).
10. Batch API. OpenAI API Documentation. URL: <https://platform.openai.com/docs/guides/batch> (дата звернення: 08.05.2026).

11. sentence-transformers/paraphrase-multilingual-mpnet-base-v2. Hugging Face. URL: <https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2> (дата звернення: 08.05.2026).

12. Storage options. RunPod Documentation. URL: <https://docs.runpod.io/pods/storage/types> (дата звернення: 08.05.2026).

13. Static Exports. Next.js Documentation. URL: <https://nextjs.org/docs/app/guides/static-exports> (дата звернення: 08.05.2026).

ДОДАТОК А Лістинг розробленої системи

```
from conditions.py:
from future import annotations

from dataclasses import dataclass

ALPHAS = [-2, -1, 0, 1, 2]

@dataclass(frozen=True, slots=True)
class SteeringCondition:
    name: str
    weights: dict[str, float]
    kind: str

def build_experiment_conditions() -> list[SteeringCondition]:
    conditions = [SteeringCondition(name="baseline", weights={}, kind="baseline")]
    for attribute in ("formal", "informal", "concise", "verbose"):
        conditions.append(SteeringCondition(name=f"{attribute} discrete", weights={attribute: 1.0}, kind="discrete"))
    for alpha in ALPHAS:
        if alpha == 0:
            continue
        conditions.append(
            SteeringCondition(
                name=f"{attribute} alpha {alpha}",
                weights={attribute: float(alpha)},
                kind="continuous",
            )
        )
    conditions.append(
        SteeringCondition(
            name="formal concise 1 1",
            weights={"formal": 1.0, "concise": 1.0},
            kind="composition",
        )
    )
    conditions.append(
        SteeringCondition(
            name="formal concise 2 2",
            weights={"formal": 2.0, "concise": 2.0},
            kind="composition",
        )
    )
    conditions.append(
        SteeringCondition(
            name="informal verbose 1 1",
            weights={"informal": 1.0, "verbose": 1.0},
            kind="composition",
        )
    )
    conditions.append(
        SteeringCondition(
            name="informal verbose 2 2",
            weights={"informal": 2.0, "verbose": 2.0},
            kind="composition",
        )
    )
    return conditions

from config.py:
from future import annotations

import json
from dataclasses import dataclass, field
from pathlib import Path
from typing import Any

DEFAULT_TARGET_MODULES = [
    "q proj",
    "k proj",
    "v proj",
    "o proj",
    "gate proj",
    "up proj",
    "down proj",
]
```

```

@dataclass(slots=True)
class ModelConfig:
    model_id: str
    revision: str | None = None
    torch_dtype: str = "bfloat16"
    use_4bit: bool = False
    device_map: str = "auto"
    lora_r: int = 16
    lora_alpha: int = 32
    lora_dropout: float = 0.05
    use_dora: bool = False
    learning_rate: float = 2e-4
    epochs: int = 2
    max_seq_length: int = 1024
    gradient_accumulation_steps: int = 4
    per_device_train_batch_size: int = 1
    target_modules: list[str] = field(default_factory=lambda: list(DEFAULT_TARGET_MODULES))

def load_json(path: str | Path) -> dict[str, Any]:
    with Path(path).open("r", encoding="utf-8") as handle:
        return json.load(handle)

def load_model_config(path: str | Path) -> ModelConfig:
    return ModelConfig(*load_json(path))

from evaluation.py:
from future import annotations

from dataclasses import dataclass

import numpy as np
import pandas as pd
from langdetect import DetectorFactory, detect
from sentence_transformers import SentenceTransformer
from transformers import pipeline

DetectorFactory.seed = 0

@dataclass(slots=True)
class ScoreComponents:
    language_ok: bool
    word_count: int
    token_count: int
    length_ratio: float
    semantic_similarity: float
    formality_score: float | None
    invalid_output: bool
    truncation_proxy: bool

def safe_detect_language(text: str) -> str:
    try:
        return detect(text)
    except Exception:
        return "unknown"

def compute_word_count(text: str) -> int:
    return len([chunk for chunk in text.split() if chunk])

def build_embedding_model(model_name: str = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2") -> SentenceTransformer:
    return SentenceTransformer(model_name)

def cosine_similarity(model: SentenceTransformer, texts_a: list[str], texts_b: list[str]) -> list[float]:
    embeddings_a = model.encode(texts_a, normalize_embeddings=True)
    embeddings_b = model.encode(texts_b, normalize_embeddings=True)
    scores = np.sum(embeddings_a * embeddings_b, axis=1)
    return [float(score) for score in scores]

def build_en_formality_scorer(model_id: str = "s-nlp/roberta-base-formality-ranker"):
    return pipeline("text-classification", model=model_id, tokenizer=model_id, top_k=None)

def score_formality_english(classifier, texts: list[str]) -> list[float]:
    raw_outputs = classifier(texts, truncation=True)

```

```

scores = []
for output in raw_outputs:
    positive = next((item["score"] for item in output if item["label"].lower().startswith("formal")), None)
    scores.append(float(positive) if positive is not None else float("nan"))
return scores

def build_ua_formality_scorer(model_id: str):
    return pipeline("text-classification", model=model_id, tokenizer=model_id, top_k=None)

def score_formality_generic(classifier, texts: list[str]) -> list[float]:
    raw_outputs = classifier(texts, truncation=True)
    scores = []
    for output in raw_outputs:
        positive = next(
            (
                item["score"]
                for item in output
                if item["label"].lower().startswith("formal") or item["label"].upper() in {"LABEL 1", "1"}
            ),
            None,
        )
        scores.append(float(positive) if positive is not None else float("nan"))
    return scores

def build_metric_frame(
    generations: pd.DataFrame,
    embedding_model: SentenceTransformer,
    english_formality_classifier=None,
    ukrainian_formality_model_id: str | None = None,
) -> pd.DataFrame:
    frame = generations.copy()
    frame["detected_language"] = frame["generation"].apply(safe_detect_language)
    frame["language_ok"] = (
        ((frame["language"] == "EN") & (frame["detected_language"] == "en"))
        | ((frame["language"] == "UA") & (frame["detected_language"] == "uk"))
    )
    frame["word_count"] = frame["generation"].apply(compute_word_count)
    frame["neutral_word_count"] = frame["neutral_text"].apply(compute_word_count)
    frame["length_ratio"] = frame["word_count"] / frame["neutral_word_count"].clip(lower=1)
    frame["semantic_similarity"] = cosine_similarity(
        embedding_model,
        frame["generation"].tolist(),
        frame["neutral_text"].tolist(),
    )
    frame["invalid_output"] = frame["generation"].str.strip().eq("")
    frame["truncation_proxy"] = frame["generation"].str.endswith(("...", "..."))
    frame["formality_score"] = np.nan
    if english_formality_classifier is not None:
        en_mask = frame["language"] == "EN"
        frame.loc[en_mask, "formality_score"] = score_formality_english(
            english_formality_classifier,
            frame.loc[en_mask, "generation"].tolist(),
        )
    if ukrainian_formality_model_id:
        ua_mask = frame["language"] == "UA"
        ukrainian_formality_classifier = build_ua_formality_scorer(ukrainian_formality_model_id)
        frame.loc[ua_mask, "formality_score"] = score_formality_generic(
            ukrainian_formality_classifier,
            frame.loc[ua_mask, "generation"].tolist(),
        )
    return frame

filtering.py:
from future import annotations

import hashlib
import re
import unicodedata
from dataclasses import dataclass

import pandas as pd

try:
    from langdetect import detect as langdetect_detect
except ImportError: # pragma: no cover
    langdetect_detect = None

SUPPORTED_LANGUAGES = {"en": "EN", "uk": "UA"}
UKRAINIAN_LETTERS = set("ІіЇїЄєІі")

DISALLOWED_PATTERNS = [

```

```

re.compile(r"\btranslate\b", re.IGNORECASE),
re.compile(r"\bпереклади\b", re.IGNORECASE),
re.compile(r"\bsolve\b", re.IGNORECASE),
re.compile(r"\bпрозв'яжи\b", re.IGNORECASE),
re.compile(r"\bpython\b", re.IGNORECASE),
re.compile(r"\bcode\b", re.IGNORECASE),
re.compile(r"\bsql\b", re.IGNORECASE),
re.compile(r"\bmath\b", re.IGNORECASE),
re.compile(r"\bconvert\b.{0,40}\b(unit|time|value|measurement)\b", re.IGNORECASE),
re.compile(r"\bперетвор\w*\b.{0,40}\b(одиниц\w*|час\w*|значенн\w*|величин\w*)\b", re.IGNORECASE),
re.compile(r"\bfill in\b", re.IGNORECASE),
re.compile(r"\bcomplete\b.{0,30}\b(sentence|table|blank|form)\b", re.IGNORECASE),
re.compile(r"\b(заповн\w*|доповн\w*)\b.{0,30}\b(реченн\w*|таблиц\w*|пропуск\w*|форм\w*)\b",
re.IGNORECASE),
re.compile(r"\b(insert|add|place)\b.{0,30}\b(comma|commas|punctuation|apostrophe|quote|quotes)\b",
re.IGNORECASE),
re.compile(r"\b(встав\w*|дода\w*|розстав\w*)\b.{0,30}\b(кому|коми|розділові знаки|апостроф)\b",
re.IGNORECASE),
re.compile(r"\b(correct|fix|proofread)\b.{0,40}\b(grammar|punctuation|spelling|typos?)\b", re.IGNORECASE),
re.compile(r"\b(виправ|перевір)\b.{0,40}\b(граматик|пунктуац|орфограф|помилк)\b", re.IGNORECASE),
re.compile(r"\bgrammar issues?\b", re.IGNORECASE),
re.compile(r"\bspelling\b.{0,30}\b(errors?|issues?)\b", re.IGNORECASE),
re.compile(r"\bis it a good message\b", re.IGNORECASE),
re.compile(r"\bwrite a context that contains the answer\b", re.IGNORECASE),
re.compile(r"\bgiven the previous question\b", re.IGNORECASE),
re.compile(r"\bmultiple choice\b", re.IGNORECASE),
re.compile(r"\btrue or false\b", re.IGNORECASE),
re.compile(r"\b(yes or no|так чи ні)\b", re.IGNORECASE),
]

```

```

TRIVIAL FACTOID PATTERNS = [
re.compile(r"^\s*what is the capital of\b", re.IGNORECASE),
re.compile(r"^\s*who (is|was|wrote)\b", re.IGNORECASE),
re.compile(r"^\s*when did\b", re.IGNORECASE),
re.compile(r"^\s*what year\b", re.IGNORECASE),
re.compile(r"^\s*which year\b", re.IGNORECASE),
re.compile(r"^\s*find the result of\b", re.IGNORECASE),
re.compile(r"^\s*answer this question:\s*what is\b", re.IGNORECASE),
re.compile(r"^\s*(яка|який|хто)\b.{0,30}\b(столиц|першим президентом)\b", re.IGNORECASE),
re.compile(r"^\s*знайди наступне число\b", re.IGNORECASE),
re.compile(r"^\s*визначте[, ]", re.IGNORECASE),
]

```

```

OPEN ENDED HINTS = {

```

```

  "EN": [
    r"\bexplain\b",
    r"\bdescribe\b",
    r"\bcompare\b",
    r"\brecommend\b",
    r"\badvise\b",
    r"\bwrite\b",
    r"\bdraft\b",
    r"\bdiscuss\b",
    r"\boutline\b",
    r"\bplan\b",
    r"\breview\b",
    r"\bemail\b",
    r"\bmessage\b",
    r"\bopinion\b",
    r"\bpros and cons\b",
  ]

```

```

  "UA": [
    r"\bпоясн\w*\b",
    r"\бопиш\w*\b",
    r"\бпорівн\w*\b",
    r"\бпорад\w*\b",
    r"\бнапиш\w*\b",
    r"\бсклад\w*\b",
    r"\бобговор\w*\b",
    r"\бсплан\w*\b",
    r"\бвідгук\b",
    r"\блист\b",
    r"\бповідомлен\w*\b",
    r"\бдумк\w*\b",
  ]

```

```

}

```

```

ENGLISH TOKEN HINTS = {

```

```

  "a",
  "an",
  "and",
  "article",
  "compare",
  "describe",
  "discuss",
  "draft",

```

$$\}$$
$$\}$$

```
class PromptRecord:
```

```
def normalize_whitespace(text: str) -> str:
    return " ".join(text.strip().split())
```

```
def normalize_prompt_key(text: str) -> str:
    normalized = normalize_whitespace(text).lower()
    normalized = re.sub(r"[\\"", "", normalized)
    normalized = re.sub(r"\\s+", " ", normalized)
    return normalized
```

```
def stable_prompt_id(language: str, prompt: str) -> str:
    payload = f'{{language}}::normalize_whitespace(prompt)'
    digest = hashlib.sha1(payload.encode("utf-8")).hexdigest()[1:16]
    return f'{{language.lower()}}_{{digest}}'
```

```
def detect_prompt_language(text: str) -> str:
    profile = script_profile(text)
    total_letters = profile["latin"] + profile["cyrillic"] + profile["other"]
    if total_letters > 0:
        if profile["cyrillic"] / total_letters >= 0.85 and profile["ukrainian specific"] > 0:
            return "uk"
        if profile["latin"] / total_letters >= 0.9 and profile["other"] == 0 and profile["cyrillic"] == 0:
            return "en"
    if langdetect.detect is not None:
        try:
            return langdetect.detect(text)
        except Exception:
            pass
    if re.search(r"[А-Яа-яІіїЄєЇї]", text):
        return "uk"
    if re.search(r"[A-Za-z]", text):
        return "en"
    return "unknown"
```

```
def script_profile(text: str) -> dict[str, int]:
    profile = {"latin": 0, "cyrillic": 0, "other": 0, "ukrainian specific": 0}
    for char in text:
        if not char.isalpha():
            continue
```

```

    name = unicodedata.name(char, "")
    if "LATIN" in name:
        profile["latin"] += 1
    elif "CYRILLIC" in name:
        profile["cyrillic"] += 1
    elif char in UKRAINIAN_LETTERS:
        profile["ukrainian_specific"] += 1
    else:
        profile["other"] += 1
    return profile

def script_rejection_reason(text: str, expected_language: str | None) -> str | None:
    if expected_language is None:
        return None
    profile = script_profile(text)
    total_letters = profile["latin"] + profile["cyrillic"] + profile["other"]
    if total_letters == 0:
        return None
    if expected_language == "EN":
        if profile["latin"] / total_letters < 0.85:
            return "script:non_latin"
        if profile["cyrillic"] > 0 or profile["other"] > 0:
            return "script:mixed_non_latin"
    if expected_language == "UA":
        if profile["cyrillic"] / total_letters < 0.85:
            return "script:non_cyrillic"
        if profile["other"] / total_letters > 0.05:
            return "script:mixed_non_cyrillic"
    return None

def lexical_language_rejection_reason(text: str, expected_language: str | None) -> str | None:
    if expected_language != "EN":
        return None
    tokens = re.findall(r"[a-z]+", text.lower())
    if len(tokens) < 3:
        return None
    english_hint_count = sum(token in ENGLISH_TOKEN_HINTS for token in tokens)
    if english_hint_count == 0:
        return "language:unknown"
    return None

def has_multiple_choice_shape(text: str) -> bool:
    normalized = normalize_whitespace(text)
    return bool(re.search(r"(?i)\b[a-d]\b|(?i)\b[a-d]\s*\b|(?i)\b1\b|(?i)\b2\b|(?i)\b3\b|(?i)\b4\b)", normalized, re.IGNORECASE))

def infer_domain(prompt: str) -> str:
    lowered = prompt.lower()
    for domain, patterns in DOMAIN_RULES.items():
        if any(re.search(pattern, lowered) for pattern in patterns):
            return domain
    return "general"

def prompt_quality_score(prompt: str, expected_language: str) -> int:
    text = normalize_whitespace(prompt)
    word_count = len(text.split())
    score = 0
    if 6 <= word_count <= 40:
        score += 2
    elif 4 <= word_count <= 60:
        score += 1
    if infer_domain(text) != "general":
        score += 2
    if any(re.search(pattern, text, re.IGNORECASE) for pattern in OPEN_ENDED_HINTS.get(expected_language, [])):
        score += 2
    if "?" in text:
        score += 1
    if has_multiple_choice_shape(text):
        score -= 3
    if any(pattern.search(text) for pattern in TRIVIAL_FACTOID_PATTERNS):
        score -= 4
    return score

def prompt_rejection_reason(prompt: str, expected_language: str | None = None) -> str | None:
    text = normalize_whitespace(prompt)
    if len(text) < 15:
        return "too_short"
    if len(text) > 320:
        return "too_long"

```

```

if "\n\n" in prompt:
    return "multi turn or block"
if has_multiple_choice_shape(text):
    return "multiple choice"
script_reason = script_rejection_reason(text, expected_language)
if script_reason is not None:
    return script_reason
lexical_reason = lexical_language_rejection_reason(text, expected_language)
if lexical_reason is not None:
    return lexical_reason
detected_language = detect_prompt_language(text)
if expected_language is not None:
    expected_code = "en" if expected_language == "EN" else "uk"
    if detected_language != expected_code:
        return f"language:{detected_language}"
for pattern in DISALLOWED_PATTERNS:
    if pattern.search(text):
        return "disallowed pattern"
for pattern in TRIVIAL_FACTOID_PATTERNS:
    if pattern.search(text):
        return "trivial factoid"
words = text.split()
lowered = text.lower()
if (
    expected_language == "EN"
    and len(words) <= 18
    and re.match(r"^(what|which|when|where|who|how many|in which)\b", lowered)
    and not re.search(r"\b(explain|describe|discuss|compare|why|how does|how do|implications|significance)\b", lowered)
):
    return "trivial factoid"
if prompt_quality_score(text, expected_language or "EN") < 1:
    return "low style flexibility"
return None

def prompt_is_eligible(prompt: str, expected_language: str | None = None) -> bool:
    return prompt_rejection_reason(prompt, expected_language) is None

def build_prompt_frame(records: list[PromptRecord]) -> pd.DataFrame:
    return pd.DataFrame(
        {
            "prompt_id": [record.prompt_id for record in records],
            "language": [record.language for record in records],
            "prompt": [record.prompt for record in records],
            "source_dataset": [record.source_dataset for record in records],
            "domain": [record.domain for record in records],
            "detected_language": [record.detected_language for record in records],
            "quality_score": [record.quality_score for record in records],
        }
    )

def stratified_split(frame: pd.DataFrame, eval_per_language: int, seed: int) -> pd.DataFrame:
    sampled_eval_ids: list[str] = []
    for language in sorted(frame["language"].unique()):
        language_frame = frame[frame["language"] == language]
        domain_frames = []
        eval_remaining = eval_per_language
        domain_counts = language_frame["domain"].value_counts().sort_values(ascending=False)
        for domain, count in domain_counts.items():
            if eval_remaining <= 0:
                break
            take = min(max(1, round(eval_per_language * (count / max(len(language_frame), 1)))), count, eval_remaining)
            domain_subset = language_frame[language_frame["domain"] == domain]
            domain_frames.append(domain_subset.sample(n=take, random_state=seed))
            eval_remaining -= take
        if eval_remaining > 0:
            remaining_subset = language_frame[~language_frame["prompt_id"].isin(pd.concat(domain_frames)["prompt_id"])]
    if domain_frames else []
    if not remaining_subset.empty:
        domain_frames.append(
            remaining_subset.sample(n=min(eval_remaining, len(remaining_subset)), random_state=seed)
        )
    eval_frame = pd.concat(domain_frames, ignore_index=True) if domain_frames else
    language_frame.sample(n=eval_per_language, random_state=seed)
    sampled_eval_ids.extend(eval_frame["prompt_id"].tolist())
    split_frame = frame.copy()
    sampled_eval_id_set = set(sampled_eval_ids)
    split_frame["split"] = split_frame["prompt_id"].apply(lambda prompt_id: "eval" if prompt_id in sampled_eval_id_set
    else "train")
    return split_frame

from __future__ import annotations

```

```

import math
import re
from dataclasses import dataclass
from typing import Iterable

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

from lm_steer_thesis.config import ModelConfig

try:
    from langdetect import DetectorFactory, detect as langdetect_detect
except ImportError: # pragma: no cover - fallback is only for thin local environments
    DetectorFactory = None
    langdetect_detect = None

if DetectorFactory is not None:
    DetectorFactory.seed = 0

REFUSAL_PATTERNS = [
    re.compile(r"bi can"?t b", re.IGNORECASE),
    re.compile(r"bне можy\b", re.IGNORECASE),
    re.compile(r"\bas an ai\b", re.IGNORECASE),
]

@dataclass(slots=True)
class LoadedGenerator:
    tokenizer: AutoTokenizer
    model: AutoModelForCausalLM

def resolve_dtype(dtype_name: str) -> torch.dtype:
    mapping = {
        "float16": torch.float16,
        "bfloat16": torch.bfloat16,
        "float32": torch.float32,
    }
    return mapping[dtype_name]

def load_generator(config: ModelConfig) -> LoadedGenerator:
    quantization_config = None
    if config.use_4bit:
        quantization_config = BitsAndBytesConfig(
            load_in_4bit=True,
            bnb_4bit_use_double_quant=True,
            bnb_4bit_quant_type="nf4",
            bnb_4bit_compute_dtype=resolve_dtype(config.torch_dtype),
        )
    tokenizer = AutoTokenizer.from_pretrained(config.model_id, revision=config.revision, token=True)
    model = AutoModelForCausalLM.from_pretrained(
        config.model_id,
        revision=config.revision,
        token=True,
        dtype=resolve_dtype(config.torch_dtype),
        device_map=config.device_map,
        quantization_config=quantization_config,
    )
    if tokenizer.pad_token_id is None:
        tokenizer.pad_token = tokenizer.eos_token
    if getattr(model.config, "is_encoder_decoder", False) is False:
        tokenizer.padding_side = "left"
    return LoadedGenerator(tokenizer=tokenizer, model=model)

def render_messages(tokenizer: AutoTokenizer, messages: list[dict[str, str]]) -> str:
    if hasattr(tokenizer, "apply_chat_template"):
        return tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
    rendered = []
    for message in messages:
        rendered.append(f"{message['role'].upper()}: {message['content']}")
    rendered.append("ASSISTANT:")
    return "\n".join(rendered)

def generate_text(
    generator: LoadedGenerator,
    messages: list[dict[str, str]],
    temperature: float = 0.2,
    top_p: float = 0.9,
    max_new_tokens: int = 256,
) -> str:

```

```

prompt_text = render_messages(generator.tokenizer, messages)
encoded = generator.tokenizer(prompt_text, return_tensors="pt").to(generator.model.device)
with torch.inference_mode():
    output = generator.model.generate(
        **encoded,
        do_sample=temperature > 0,
        temperature=temperature,
        top_p=top_p,
        max_new_tokens=max_new_tokens,
        pad_token_id=generator.tokenizer.pad_token_id,
        eos_token_id=generator.tokenizer.eos_token_id,
    )
generated_tokens = output[0][encoded["input_ids"].shape[-1]:]
return generator.tokenizer.decode(generated_tokens, skip_special_tokens=True).strip()

def generate_texts(
    generator: LoadedGenerator,
    messages_batch: list[list[dict[str, str]]],
    temperature: float = 0.2,
    top_p: float = 0.9,
    max_new_tokens: int = 256,
) -> list[str]:
    prompt_texts = [render_messages(generator.tokenizer, messages) for messages in messages_batch]
    encoded = generator.tokenizer(prompt_texts, return_tensors="pt", padding=True).to(generator.model.device)
    with torch.inference_mode():
        output = generator.model.generate(
            **encoded,
            do_sample=temperature > 0,
            temperature=temperature,
            top_p=top_p,
            max_new_tokens=max_new_tokens,
            pad_token_id=generator.tokenizer.pad_token_id,
            eos_token_id=generator.tokenizer.eos_token_id,
        )
    prompt_width = encoded["input_ids"].shape[-1]
    generations: list[str] = []
    for row_index in range(output.shape[0]):
        generated_tokens = output[row_index][prompt_width:]
        generations.append(generator.tokenizer.decode(generated_tokens, skip_special_tokens=True).strip())
    return generations

def detect_language(text: str) -> str:
    if langdetect.detect is None:
        # Simple fallback so unit tests can still run without optional runtime deps installed.
        if re.search(r"[A-Яa-яĲĳĴĵĶķĹłŀŁŃčĎđĚěĜgĤĥİıİıĲĳĴĵĶķĹłŀŁŃčĎđĚěĜgĤĥİıİı]", text):
            return "uk"
        if re.search(r"[A-Za-z]", text):
            return "en"
        return "unknown"
    try:
        return langdetect.detect(text)
    except Exception:
        return "unknown"

def text_is_refusal(text: str) -> bool:
    return any(pattern.search(text) for pattern in REFUSAL_PATTERNS)

def word_count(text: str) -> int:
    return len([token for token in re.split(r"\s+", text.strip()) if token])

def length_ratio(candidate: str, neutral_text: str) -> float:
    denominator = max(word_count(neutral_text), 1)
    return word_count(candidate) / denominator

def semantic_overlap_proxy(candidate: str, neutral_text: str) -> float:
    candidate_words = set(re.findall(r"\w+", candidate.lower()))
    neutral_words = set(re.findall(r"\w+", neutral_text.lower()))
    if not candidate_words or not neutral_words:
        return 0.0
    return len(candidate_words & neutral_words) / math.sqrt(len(candidate_words) * len(neutral_words))

def qc_flags(text: str, neutral_text: str, expected_language: str, attribute: str) -> list[str]:
    flags: list[str] = []
    if not text.strip():
        flags.append("empty")
    if text_is_refusal(text):

```

```

        flags.append("refusal")
    detected = detect_language(text)
    if expected_language == "EN" and detected != "en":
        flags.append(f"language:{detected}")
    if expected_language == "UA" and detected != "uk":
        flags.append(f"language:{detected}")
    ratio = length_ratio(text, neutral_text)
    if attribute == "concise" and not (0.5 <= ratio <= 0.75):
        flags.append(f"ratio:{ratio:.2f}")
    if attribute == "verbose" and not (1.3 <= ratio <= 1.8):
        flags.append(f"ratio:{ratio:.2f}")
    if attribute in {"formal", "informal"} and semantic_overlap_proxy(text, neutral_text) < 0.35:
        flags.append("semantic drift")
    return flags

def regenerate_until_pass(
    generator: LoadedGenerator,
    messages: list[dict[str, str]],
    expected_language: str,
    attribute: str,
    neutral_text: str,
    max_attempts: int = 3,
) -> tuple[str, list[str], int]:
    candidate = ""
    flags: list[str] = ["empty"]
    for attempt in range(1, max_attempts + 1):
        candidate = generate_text(generator, messages)
        flags = qc_flags(candidate, neutral_text, expected_language, attribute)
        if not flags:
            return candidate, flags, attempt
    return candidate, flags, max_attempts

from io.py:
from future import annotations

import json
from pathlib import Path
from typing import Any

import pandas as pd

def ensure_parent_dir(path: str | Path) -> Path:
    target = Path(path)
    target.parent.mkdir(parents=True, exist_ok=True)
    return target

def read_table(path: str | Path) -> pd.DataFrame:
    source = Path(path)
    if source.suffix == ".parquet":
        return pd.read_parquet(source)
    if source.suffix == ".csv":
        return pd.read_csv(source)
    raise ValueError(f"Unsupported table format: {source}")

def write_table(frame: pd.DataFrame, path: str | Path) -> None:
    target = ensure_parent_dir(path)
    if target.suffix == ".parquet":
        frame.to_parquet(target, index=False)
        return
    if target.suffix == ".csv":
        frame.to_csv(target, index=False)
        return
    raise ValueError(f"Unsupported table format: {target}")

def write_json(payload: dict[str, Any], path: str | Path) -> None:
    target = ensure_parent_dir(path)
    with target.open("w", encoding="utf-8") as handle:
        json.dump(payload, handle, indent=2, ensure_ascii=False)

from openai_data.py:
from future import annotations

import json
import math
import os
import re
from collections.abc import Mapping
from dataclasses import asdict, dataclass
from typing import Any

```

```

ATTRIBUTES = ("formal", "informal", "concise", "verbose")
RESPONSE_SCHEMA_NAME = "styled target bundle"
REWRITE_SCHEMA_NAME = "styled rewrite bundle"
NEUTRAL_BATCH_STAGE = "neutral"
REWRITE_BATCH_STAGE = "rewrite"
ATTRIBUTE_BATCH_STAGE = "attribute"
REPAIR_BATCH_STAGE = "repair"

```

```

RESPONSE_SCHEMA: dict[str, Any] = {
    "type": "object",
    "additionalProperties": False,
    "properties": {
        "prompt_id": {"type": "string"},
        "language": {"type": "string", "enum": ["EN", "UA"]},
        "neutral": {"type": "string"},
        "formal": {"type": "string"},
        "informal": {"type": "string"},
        "concise": {"type": "string"},
        "verbose": {"type": "string"},
    },
    "required": ["prompt_id", "language", "neutral", "formal", "informal", "concise", "verbose"],
}

```

```

REWRITE_SCHEMA: dict[str, Any] = {
    "type": "object",
    "additionalProperties": False,
    "properties": {
        "prompt_id": {"type": "string"},
        "language": {"type": "string", "enum": ["EN", "UA"]},
        "formal": {"type": "string"},
        "informal": {"type": "string"},
        "concise": {"type": "string"},
        "verbose": {"type": "string"},
    },
    "required": ["prompt_id", "language", "formal", "informal", "concise", "verbose"],
}

```

```

try:
    from langdetect import detect as langdetect_detect
except ImportError: # pragma: no cover - fallback is only for thin local environments
    langdetect_detect = None

```

```

REFUSAL_PATTERNS = [
    re.compile(r"\bI can't\b", re.IGNORECASE),
    re.compile(r"\bне можу\b", re.IGNORECASE),
    re.compile(r"\bas an ai\b", re.IGNORECASE),
]

```

```

def require_openai_api_key() -> str:
    api_key = os.environ.get("OPENAI_API_KEY")
    if not api_key:
        raise RuntimeError("OPENAI_API_KEY is required for OpenAI data generation.")
    return api_key

```

```

def detect_language(text: str) -> str:
    if langdetect_detect is None:
        if re.search(r"[А-Яа-яІііЄєІі]", text):
            return "uk"
        if re.search(r"[A-Za-z]", text):
            return "en"
        return "unknown"
    try:
        return langdetect_detect(text)
    except Exception:
        return "unknown"

```

```

def text_is_refusal(text: str) -> bool:
    return any(pattern.search(text) for pattern in REFUSAL_PATTERNS)

```

```

def word_count(text: str) -> int:
    return len([token for token in re.split(r"\s+", text.strip()) if token])

```

```

def length_ratio(candidate: str, neutral_text: str) -> float:
    denominator = max(word_count(neutral_text), 1)
    return word_count(candidate) / denominator

```

```

def semantic_overlap_proxy(candidate: str, neutral_text: str) -> float:

```

```

candidate_words = set(re.findall(r"\w+", candidate.lower()))
neutral_words = set(re.findall(r"\w+", neutral_text.lower()))
if not candidate_words or not neutral_words:
    return 0.0
return len(candidate_words & neutral_words) / math.sqrt(len(candidate_words) * len(neutral_words))

```

```

def qc_flags(text: str, neutral_text: str, expected_language: str, attribute: str) -> list[str]:
    flags: list[str] = []
    if not text.strip():
        flags.append("empty")
    if text is refusal(text):
        flags.append("refusal")
    detected = detect_language(text)
    if expected_language == "EN" and detected != "en":
        flags.append(f"language: {detected}")
    if expected_language == "UA" and detected != "uk":
        flags.append(f"language: {detected}")
    ratio = length_ratio(text, neutral_text)
    if attribute == "concise" and not (0.5 <= ratio <= 0.75):
        flags.append(f"ratio: {ratio:.2f}")
    if attribute == "verbose" and not (1.3 <= ratio <= 1.8):
        flags.append(f"ratio: {ratio:.2f}")
    if attribute in {"formal", "informal"} and semantic_overlap_proxy(text, neutral_text) < 0.35:
        flags.append("semantic drift")
    return flags

```

```

def response_format_schema(*, name: str, schema: dict[str, Any]) -> dict[str, Any]:
    return {
        "type": "json schema",
        "name": name,
        "schema": schema,
        "strict": True,
    }

```

```

def build_system_prompt() -> str:
    return (
        "You are generating high-quality thesis training data for controllable style transfer. "
        "Return a JSON object that matches the provided schema exactly. "
        "Write every field in the same language as the prompt. "
        "Keep the meaning aligned across all variants. "
        "Do not add markdown, comments, or extra keys. "
        "The concise version should usually be about 50%-75% of the neutral answer length. "
        "The verbose version should usually be about 130%-180% of the neutral answer length."
    )

```

```

def build_user_prompt(prompt_id: str, language: str, prompt: str) -> str:
    return (
        "Create five aligned answers for the following prompt.\n\n"
        f"prompt id: {prompt_id}\n"
        f"language: {language}\n"
        f"prompt: \n{prompt}\n\n"
        "Output rules:\n"
        "- neutral: faithful answer, no explicit style marker\n"
        "- formal: more professional and formal\n"
        "- informal: more conversational and natural\n"
        "- concise: shorter but still adequate\n"
        "- verbose: longer and more detailed\n"
        "- keep facts and intent consistent across variants\n"
        "- use the same language as the prompt in every field\n"
        "- avoid refusals unless the prompt is inherently unsafe, which should be rare after filtering\n"
    )

```

```

def build_neutral_system_prompt() -> str:
    return (
        "You are generating high-quality thesis training data. "
        "Write one faithful, helpful answer to the user's prompt in the same language as the prompt. "
        "Do not add markdown, bullet points, labels, or commentary about style."
    )

```

```

def build_neutral_user_prompt(prompt: str, language: str) -> str:
    return (
        f"Language: {language}\n"
        "Task: write a neutral reference answer in the same language.\n\n"
        f"Prompt: \n{prompt}\n"
    )

```

```

def word_budget_bounds(neutral text: str) -> dict[str, int]:
    neutral_words = max(word_count(neutral text), 1)
    concise_min = max(1, math.floor(neutral_words * 0.50))
    concise_max = max(concise_min, math.floor(neutral_words * 0.75))
    verbose_min = max(concise_max + 1, math.ceil(neutral_words * 1.30))
    verbose_max = max(verbose_min, math.ceil(neutral_words * 1.80))
    return {
        "neutral words": neutral_words,
        "concise min": concise_min,
        "concise max": concise_max,
        "verbose min": verbose_min,
        "verbose max": verbose_max,
    }

def build_rewrite_system_prompt() -> str:
    return (
        "You are generating high-quality thesis training data for controllable style transfer. "
        "Return a JSON object that matches the provided schema exactly. "
        "Keep meaning, facts, and prompt relevance aligned with the neutral draft. "
        "Write every field in the same language as the prompt. "
        "Do not add markdown, comments, or extra keys."
    )

def build_rewrite_user_prompt(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    neutral_text: str,
    requested_attributes: tuple[str, ...] = ATTRIBUTES,
) -> str:
    bounds = word_budget_bounds(neutral_text)
    requested = ", ".join(requested_attributes)
    concise_rule = (
        f"concise must be between {bounds['concise min']} and {bounds['concise max']} words, inclusive."
    )
    verbose_rule = (
        f"verbose must be between {bounds['verbose min']} and {bounds['verbose max']} words, inclusive."
    )
    attribute_rules = {
        "formal": (
            "formal must sound more professional and formal while preserving meaning and staying close to the "
            "neutral draft length."
        ),
        "informal": (
            "informal must sound more conversational and natural while preserving meaning and staying close to the "
            "neutral draft length."
        ),
        "concise": concise_rule,
        "verbose": verbose_rule,
    }
    selected_rules = "\n".join(f"- {attribute_rules[attribute]}" for attribute in requested_attributes)
    return (
        "Rewrite the neutral draft into controlled style variants.\n\n"
        f"prompt id: {prompt_id}\n"
        f"language: {language}\n"
        f"requested fields: {requested}\n"
        f"neutral word count: {bounds['neutral words']}\n\n"
        f"Original prompt:\n{prompt}\n\n"
        f"Neutral draft:\n{neutral_text}\n\n"
        "Rules:\n"
        "- keep the same core information and answer the same prompt\n"
        "- keep the same language as the neutral draft\n"
        "- avoid refusals and safety boilerplate\n"
        f"{selected_rules}\n"
    )

def single_attribute_schema(attribute: str) -> dict[str, Any]:
    return {
        "type": "object",
        "additionalProperties": False,
        "properties": {
            "prompt_id": {"type": "string"},
            "language": {"type": "string", "enum": ["EN", "UA"]},
            attribute: {"type": "string"},
        },
        "required": ["prompt_id", "language", attribute],
    }

def build_responses(input(prompt_id: str, language: str, prompt: str) -> list[dict[str, Any]]:
    return [

```

```

    {
      "role": "developer",
      "content": [{"type": "input text", "text": build system prompt()}],
    },
    {
      "role": "user",
      "content": [{"type": "input text", "text": build user prompt(prompt id, language, prompt)}],
    },
  ],
]

```

```

def build_neutral_responses input(prompt: str, language: str) -> list[dict[str, Any]]:
  return [
    {
      "role": "developer",
      "content": [{"type": "input text", "text": build_neutral_system_prompt()}],
    },
    {
      "role": "user",
      "content": [{"type": "input text", "text": build_neutral_user_prompt(prompt, language)}],
    },
  ],
]

```

```

def build_rewrite_responses input(
  *,
  prompt_id: str,
  language: str,
  prompt: str,
  neutral_text: str,
  requested_attributes: tuple[str, ...] = ATTRIBUTES,
) -> list[dict[str, Any]]:
  return [
    {
      "role": "developer",
      "content": [{"type": "input text", "text": build_rewrite_system_prompt()}],
    },
    {
      "role": "user",
      "content": [
        {
          "type": "input text",
          "text": build_rewrite_user_prompt(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            neutral_text=neutral_text,
            requested_attributes=requested_attributes,
          ),
        },
      ],
    },
  ],
]

```

```

def build_responses_body(
  *,
  prompt_id: str,
  language: str,
  prompt: str,
  model: str,
  max_output_tokens: int,
  reasoning_effort: str,
  store: bool = False,
) -> dict[str, Any]:
  return {
    "model": model,
    "input": build_responses_input(prompt_id, language, prompt),
    "reasoning": {"effort": reasoning_effort},
    "max_output_tokens": max_output_tokens,
    "store": store,
    "text": {"format": response_format_schema(name=RESPONSE_SCHEMA_NAME,
schema=RESPONSE_SCHEMA)},
    "metadata": {
      "prompt_id": prompt_id,
      "language": language,
    },
  },
]

```

```

def build_neutral_response_body(
  *,
  language: str,
  prompt: str,
  model: str,

```

```

max_output_tokens: int,
reasoning_effort: str,
store: bool = False,
) -> dict[str, Any]:
    return {
        "model": model,
        "input": build_neutral_responses_input(prompt, language),
        "reasoning": {"effort": reasoning_effort},
        "max_output_tokens": max_output_tokens,
        "store": store,
        "metadata": {
            "language": language,
            "stage": "neutral",
        },
    },
}

```

```

def build_rewrite_response_body(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    neutral_text: str,
    model: str,
    max_output_tokens: int,
    reasoning_effort: str,
    requested_attributes: tuple[str, ...] = ATTRIBUTES,
    store: bool = False,
) -> dict[str, Any]:
    schema = REWRITE_SCHEMA if requested_attributes == ATTRIBUTES else
single_attribute_schema(requested_attributes[0])
    schema_name = REWRITE_SCHEMA_NAME if requested_attributes == ATTRIBUTES else
f"requested_attributes[0]"
    return {
        "model": model,
        "input": build_rewrite_responses_input(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            neutral_text=neutral_text,
            requested_attributes=requested_attributes,
        ),
        "reasoning": {"effort": reasoning_effort},
        "max_output_tokens": max_output_tokens,
        "store": store,
        "text": {"format": response_format_schema(name=schema_name, schema=schema)},
        "metadata": {
            "prompt_id": prompt_id,
            "language": language,
            "stage": "rewrite",
            "requested_attributes": ",".join(requested_attributes),
        },
    },
}

```

```

def build_batch_request_line(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    model: str,
    max_output_tokens: int,
    reasoning_effort: str,
) -> dict[str, Any]:
    return {
        "custom_id": prompt_id,
        "method": "POST",
        "url": "/v1/responses",
        "body": build_responses_body(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            model=model,
            max_output_tokens=max_output_tokens,
            reasoning_effort=reasoning_effort,
            store=False,
        ),
    },
}

```

```

def make_batch_custom_id(stage: str, prompt_id: str) -> str:
    return f"{stage}::{prompt_id}"

```

```

def parse_batch_custom_id(custom_id: str) -> tuple[str, str]:

```

```

if "::" not in custom_id:
    raise ValueError(f"Unsupported batch custom_id format: {custom_id}")
stage, prompt_id = custom_id.split("::", 1)
if not stage or not prompt_id:
    raise ValueError(f"Unsupported batch custom_id format: {custom_id}")
return stage, prompt_id

```

```

def build_neutral_batch_request_line(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    model: str,
    max_output_tokens: int,
    reasoning_effort: str,
) -> dict[str, Any]:
    return {
        "custom_id": make_batch_custom_id(NEUTRAL_BATCH_STAGE, prompt_id),
        "method": "POST",
        "url": "/v1/responses",
        "body": build_neutral_response_body(
            language=language,
            prompt=prompt,
            model=model,
            max_output_tokens=max_output_tokens,
            reasoning_effort=reasoning_effort,
            store=False,
        ),
    }

```

```

def build_rewrite_batch_request_line(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    neutral_text: str,
    model: str,
    max_output_tokens: int,
    reasoning_effort: str,
    requested_attributes: tuple[str, ...] = ATTRIBUTES,
) -> dict[str, Any]:
    return {
        "custom_id": make_batch_custom_id(REWRITE_BATCH_STAGE, prompt_id),
        "method": "POST",
        "url": "/v1/responses",
        "body": build_rewrite_response_body(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            neutral_text=neutral_text,
            model=model,
            max_output_tokens=max_output_tokens,
            reasoning_effort=reasoning_effort,
            requested_attributes=requested_attributes,
            store=False,
        ),
    }

```

```

def make_attribute_batch_custom_id(prompt_id: str, attribute: str) -> str:
    return f"{ATTRIBUTE_BATCH_STAGE}::{prompt_id}::{attribute}"

```

```

def parse_attribute_batch_custom_id(custom_id: str) -> tuple[str, str]:
    parts = custom_id.split("::", 2)
    if len(parts) != 3 or parts[0] != ATTRIBUTE_BATCH_STAGE:
        raise ValueError(f"Unsupported attribute batch custom_id format: {custom_id}")
    prompt_id, attribute = parts
    if attribute not in ATTRIBUTES:
        raise ValueError(f"Unsupported attribute in custom_id: {custom_id}")
    return prompt_id, attribute

```

```

def build_attribute_batch_request_line(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    neutral_text: str,
    attribute: str,
    model: str,
    max_output_tokens: int,

```

```

    reasoning_effort: str,
) -> dict[str, Any]:
    return {
        "custom_id": make_attribute_batch_custom_id(prompt_id, attribute),
        "method": "POST",
        "url": "/v1/responses",
        "body": build_rewrite_response_body(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            neutral_text=neutral_text,
            model=model,
            max_output_tokens=max_output_tokens,
            reasoning_effort=reasoning_effort,
            requested_attributes=(attribute,),
            store=False,
        ),
    }

```

```

def make_repair_batch_custom_id(prompt_id: str, attribute: str) -> str:
    return f'{REPAIR_BATCH_STAGE}::{prompt_id}::{attribute}'

```

```

def parse_repair_batch_custom_id(custom_id: str) -> tuple[str, str]:
    parts = custom_id.split("::", 2)
    if len(parts) != 3 or parts[0] != REPAIR_BATCH_STAGE:
        raise ValueError(f"Unsupported repair batch custom id format: {custom_id}")
    prompt_id, attribute = parts
    if attribute not in ATTRIBUTES:
        raise ValueError(f"Unsupported repair attribute in custom id: {custom_id}")
    return prompt_id, attribute

```

```

def build_repair_batch_request_line(
    *,
    prompt_id: str,
    language: str,
    prompt: str,
    neutral_text: str,
    attribute: str,
    model: str,
    max_output_tokens: int,
    reasoning_effort: str,
) -> dict[str, Any]:
    return {
        "custom_id": make_repair_batch_custom_id(prompt_id, attribute),
        "method": "POST",
        "url": "/v1/responses",
        "body": build_rewrite_response_body(
            prompt_id=prompt_id,
            language=language,
            prompt=prompt,
            neutral_text=neutral_text,
            model=model,
            max_output_tokens=max_output_tokens,
            reasoning_effort=reasoning_effort,
            requested_attributes=(attribute,),
            store=False,
        ),
    }

```

```

def extract_response_text(response_body: Mapping[str, Any]) -> str:
    fragments: list[str] = []
    for item in response_body.get("output", []):
        if item.get("type") != "message":
            continue
        for content_part in item.get("content", []):
            if content_part.get("type") == "output_text":
                text = content_part.get("text", "")
                if text:
                    fragments.append(text)
    return "".join(fragments).strip()

```

```

def parse_styled_bundle(response_body: Mapping[str, Any]) -> dict[str, str]:
    raw_text = extract_response_text(response_body)
    if not raw_text:
        raise ValueError("OpenAI response did not contain output text content.")
    payload = json.loads(raw_text)
    required = ["prompt_id", "language", "neutral", "formal", "informal", "concise", "verbose"]
    missing = [field for field in required if field not in payload]
    if missing:

```

```

        raise ValueError(f"Structured response missing fields: {missing}")
    bundle = {field: str(payload[field]).strip() for field in required}
    if bundle["language"] not in {"EN", "UA"}:
        raise ValueError(f"Unsupported language value in structured response: {bundle['language']}")
    return bundle

def parse_rewrite_bundle(response body: Mapping[str, Any]) -> dict[str, str]:
    raw_text = extract_response_text(response body)
    if not raw_text:
        raise ValueError("OpenAI response did not contain output text content.")
    payload = json.loads(raw_text)
    required = ["prompt_id", "language", "formal", "informal", "concise", "verbose"]
    missing = [field for field in required if field not in payload]
    if missing:
        raise ValueError(f"Structured rewrite response missing fields: {missing}")
    bundle = {field: str(payload[field]).strip() for field in required}
    if bundle["language"] not in {"EN", "UA"}:
        raise ValueError(f"Unsupported language value in structured rewrite response: {bundle['language']}")
    return bundle

def parse_single_attribute_bundle(response body: Mapping[str, Any], attribute: str) -> dict[str, str]:
    raw_text = extract_response_text(response body)
    if not raw_text:
        raise ValueError("OpenAI response did not contain output text content.")
    payload = json.loads(raw_text)
    required = ["prompt_id", "language", attribute]
    missing = [field for field in required if field not in payload]
    if missing:
        raise ValueError(f"Single-attribute rewrite response missing fields: {missing}")
    bundle = {field: str(payload[field]).strip() for field in required}
    if bundle["language"] not in {"EN", "UA"}:
        raise ValueError(f"Unsupported language value in single-attribute rewrite response: {bundle['language']}")
    return bundle

def extract_batch_response_body(batch_line: Mapping[str, Any]) -> dict[str, Any]:
    if "response" in batch_line and isinstance(batch_line["response"], Mapping):
        response = batch_line["response"]
        if "body" in response and isinstance(response["body"], Mapping):
            return dict(response["body"])
        return dict(response)
    if "body" in batch_line and isinstance(batch_line["body"], Mapping):
        return dict(batch_line["body"])
    return dict(batch_line)

@dataclass(slots=True)
class StyledExampleRow:
    example_id: str
    prompt_id: str
    language: str
    prompt: str
    attribute: str
    target_text: str
    neutral_text: str
    generator_model: str
    qc_flags: str
    rewrite_attempts: int
    neutral_attempts: int
    neutral_qc_flags: str
    split: str
    domain: str

@dataclass(slots=True)
class NeutralExampleRow:
    prompt_id: str
    language: str
    prompt: str
    neutral_text: str
    generator_model: str
    neutral_qc_flags: str
    neutral_attempts: int
    split: str
    domain: str

def bundle_to_rows(
    prompt_record: Mapping[str, Any],
    bundle: Mapping[str, str],
    *,
    generator_model: str,

```

```

) -> list[dict[str, Any]]:
    prompt_id = str(prompt_record["prompt_id"])
    prompt_language = str(prompt_record["language"])
    if bundle["prompt_id"] != prompt_id:
        raise ValueError(f"Structured response prompt_id mismatch: {bundle['prompt_id']} != {prompt_id}")
    if bundle["language"] != prompt_language:
        raise ValueError(f"Structured response language mismatch: {bundle['language']} != {prompt_language}")

    neutral_text = bundle["neutral"]
    neutral_flag_list = qc_flags(neutral_text, neutral_text, prompt_language, "neutral")
    rows: list[dict[str, Any]] = []
    for attribute in ATTRIBUTES:
        flag_list = qc_flags(bundle[attribute], neutral_text, prompt_language, attribute)
        rows.append(
            asdict(
                StyledExampleRow(
                    example_id=f"{prompt_id} {attribute}",
                    prompt_id=prompt_id,
                    language=prompt_language,
                    prompt=str(prompt_record["prompt"]),
                    attribute=attribute,
                    target_text=neutral_text,
                    neutral_text=neutral_text,
                    generator_model=generator_model,
                    qc_flags="".join(flag_list),
                    rewrite_attempts=1,
                    neutral_attempts=1,
                    neutral_qc_flags="".join(neutral_flag_list),
                    split=str(prompt_record["split"]),
                    domain=str(prompt_record["domain"]),
                )
            )
        )
    return rows

```

```

def neutral_text_to_row(
    prompt_record: Mapping[str, Any],
    neutral_text: str,
    *,
    generator_model: str,
    neutral_attempts: int = 1,
) -> dict[str, Any]:
    prompt_language = str(prompt_record["language"])
    neutral_flag_list = qc_flags(neutral_text, neutral_text, prompt_language, "neutral")
    return asdict(
        NeutralExampleRow(
            prompt_id=str(prompt_record["prompt_id"]),
            language=prompt_language,
            prompt=str(prompt_record["prompt"]),
            neutral_text=neutral_text,
            generator_model=generator_model,
            neutral_qc_flags="".join(neutral_flag_list),
            neutral_attempts=neutral_attempts,
            split=str(prompt_record["split"]),
            domain=str(prompt_record["domain"]),
        )
    )

```

```

def rewrite_bundle_to_rows(
    prompt_record: Mapping[str, Any],
    rewrite_bundle: Mapping[str, str],
    *,
    neutral_text: str,
    generator_model: str,
    rewrite_attempts: int = 1,
    neutral_attempts: int = 1,
) -> list[dict[str, Any]]:
    prompt_id = str(prompt_record["prompt_id"])
    prompt_language = str(prompt_record["language"])
    if rewrite_bundle["prompt_id"] != prompt_id:
        raise ValueError(f"Structured rewrite response prompt_id mismatch: {rewrite_bundle['prompt_id']} != {prompt_id}")
    if rewrite_bundle["language"] != prompt_language:
        raise ValueError(
            f"Structured rewrite response language mismatch: {rewrite_bundle['language']} != {prompt_language}"
        )

    neutral_flag_list = qc_flags(neutral_text, neutral_text, prompt_language, "neutral")
    rows: list[dict[str, Any]] = []
    for attribute in ATTRIBUTES:
        flag_list = qc_flags(rewrite_bundle[attribute], neutral_text, prompt_language, attribute)
        rows.append(
            asdict(
                StyledExampleRow(

```

```

        example_id=f"{prompt_id} {attribute}",
        prompt_id=prompt_id,
        language=prompt_language,
        prompt=str(prompt_record["prompt"]),
        attribute=attribute,
        target_text=rewrite_bundle[attribute],
        neutral_text=neutral_text,
        generator_model=generator_model,
        qc_flags="".join(flag_list),
        rewrite_attempts=rewrite_attempts,
        neutral_attempts=neutral_attempts,
        neutral_qc_flags="".join(neutral_flag_list),
        split=str(prompt_record["split"]),
        domain=str(prompt_record["domain"]),
    )
)
)
return rows

def single_attribute_to_row(
    prompt_record: Mapping[str, Any],
    attribute_bundle: Mapping[str, str],
    *,
    attribute: str,
    neutral_text: str,
    generator_model: str,
    rewrite_attempts: int = 1,
    neutral_attempts: int = 1,
) -> dict[str, Any]:
    prompt_id = str(prompt_record["prompt_id"])
    prompt_language = str(prompt_record["language"])
    if attribute_bundle["prompt_id"] != prompt_id:
        raise ValueError(f"Single-attribute response prompt id mismatch: {attribute_bundle['prompt_id']} != {prompt_id}")
    if attribute_bundle["language"] != prompt_language:
        raise ValueError(
            f"Single-attribute response language mismatch: {attribute_bundle['language']} != {prompt_language}"
        )
    target_text = str(attribute_bundle[attribute])
    neutral_flag_list = qc_flags(neutral_text, neutral_text, prompt_language, "neutral")
    flag_list = qc_flags(target_text, neutral_text, prompt_language, attribute)
    return asdict(
        StyledExampleRow(
            example_id=f"{prompt_id} {attribute}",
            prompt_id=prompt_id,
            language=prompt_language,
            prompt=str(prompt_record["prompt"]),
            attribute=attribute,
            target_text=target_text,
            neutral_text=neutral_text,
            generator_model=generator_model,
            qc_flags="".join(flag_list),
            rewrite_attempts=rewrite_attempts,
            neutral_attempts=neutral_attempts,
            neutral_qc_flags="".join(neutral_flag_list),
            split=str(prompt_record["split"]),
            domain=str(prompt_record["domain"]),
        )
    )

def summarize_rows(rows: list[dict[str, Any]], *, model: str, prompt_count: int, failures: int) -> dict[str, Any]:
    language_counts: dict[str, int] = {}
    attribute_counts: dict[str, int] = {}
    qc_failures = 0
    for row in rows:
        language = str(row["language"])
        attribute = str(row["attribute"])
        language_counts[language] = language_counts.get(language, 0) + 1
        attribute_counts[attribute] = attribute_counts.get(attribute, 0) + 1
        if row["qc_flags"]:
            qc_failures += 1
    return {
        "model": model,
        "prompt_count": prompt_count,
        "rows": len(rows),
        "failures": failures,
        "by_language": language_counts,
        "by_attribute": attribute_counts,
        "qc_failures": qc_failures,
    }

def summarize_neutral_rows(rows: list[dict[str, Any]], *, model: str, prompt_count: int, failures: int) -> dict[str, Any]:
    language_counts: dict[str, int] = {}

```

```

qc failures = 0
for row in rows:
    language = str(row["language"])
    language_counts[language] = language_counts.get(language, 0) + 1
    if row["neutral_qc_flags"]:
        qc_failures += 1
return {
    "model": model,
    "prompt count": prompt_count,
    "rows": len(rows),
    "failures": failures,
    "by language": language_counts,
    "qc failures": qc_failures,
}

from prompting.py:
from future import annotations

ATTRIBUTE_PROMPTS = {
    "neutral": (
        "You are preparing high-quality thesis data. Answer the user's prompt faithfully in the same language, "
        "with clear content and no extra formatting unless requested."
    ),
    "formal": (
        "Rewrite the draft into a more formal, professional register. Keep the meaning intact, keep the same language, "
        "and avoid slang."
    ),
    "informal": (
        "Rewrite the draft into a more informal, conversational register. Keep the meaning intact, keep the same language, "
        "and allow natural colloquial phrasing."
    ),
    "concise": (
        "Rewrite the draft so it is concise and substantially shorter while preserving the core meaning and the same language."
    ),
    "verbose": (
        "Rewrite the draft so it is more detailed and noticeably longer while preserving the core meaning and the same
language."
    ),
}

def build_neutral_messages(prompt: str) -> list[dict[str, str]]:
    return [
        {"role": "system", "content": ATTRIBUTE_PROMPTS["neutral"]},
        {"role": "user", "content": prompt},
    ]

def build_style_rewrite_messages(attribute: str, prompt: str, neutral_text: str) -> list[dict[str, str]]:
    return [
        {"role": "system", "content": ATTRIBUTE_PROMPTS[attribute]},
        {
            "role": "user",
            "content": (
                f"Original prompt:\n{prompt}\n\nDraft answer:\n{neutral_text}\n\n"
                "Return only the rewritten answer."
            )
        },
    ]

from training.py:
from future import annotations

from dataclasses import dataclass
import inspect

import pandas as pd
import torch
from datasets import Dataset
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    DataCollatorForLanguageModeling,
    Trainer,
    TrainingArguments,
)

from lm_steer.thesis.config import ModelConfig
from lm_steer.thesis.generation import resolve_dtype

@dataclass(slots=True)
class TrainingBundle:

```

```
tokenizer: AutoTokenizer
model: AutoModelForCausalLM
```

```
def build_chat_example(tokenizer: AutoTokenizer, prompt: str, target_text: str) -> str:
    messages = [
        {"role": "user", "content": prompt},
        {"role": "assistant", "content": target_text},
    ]
    if hasattr(tokenizer, "apply_chat_template"):
        return tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=False)
    return f"USER: {prompt}\nASSISTANT: {target_text}"
```

```
def load_training_bundle(config: ModelConfig) -> TrainingBundle:
    quantization_config = None
    if config.use_4bit:
        quantization_config = BitsAndBytesConfig(
            load_in_4bit=True,
            bnb_4bit_use_double_quant=True,
            bnb_4bit_quant_type="nf4",
            bnb_4bit_compute_dtype=resolve_dtype(config.torch_dtype),
        )
    tokenizer = AutoTokenizer.from_pretrained(config.model_id, revision=config.revision, token=True)
    if tokenizer.pad_token_id is None:
        tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(
        config.model_id,
        revision=config.revision,
        token=True,
        dtype=resolve_dtype(config.torch_dtype),
        device_map="auto",
        quantization_config=quantization_config,
    )
    if config.use_4bit:
        model = prepare_model_for_kbit_training(model)
    lora_config = LoraConfig(
        r=config.lora_r,
        lora_alpha=config.lora_alpha,
        lora_dropout=config.lora_dropout,
        bias="none",
        target_modules=config.target_modules,
        task_type="CAUSAL_LM",
        use_dora=config.use_dora,
    )
    return TrainingBundle(tokenizer=tokenizer, model=get_peft_model(model, lora_config))
```

```
def build_attribute_dataset(frame: pd.DataFrame, tokenizer: AutoTokenizer, attribute: str, max_seq_length: int) -> Dataset:
    subset = frame[frame["attribute"] == attribute].copy()
    subset["text"] = subset.apply(lambda row: build_chat_example(tokenizer, row["prompt"], row["target_text"]), axis=1)
    dataset = Dataset.from_pandas(subset[["text"]], preserve_index=False)
```

```
def tokenize(batch: dict[str, list[str]]) -> dict[str, list[list[int]]]:
    encoded = tokenizer(
        batch["text"],
        truncation=True,
        max_length=max_seq_length,
        padding=False,
    )
    if "token_type_ids" not in encoded:
        encoded["token_type_ids"] = [[0] * len(input_ids) for input_ids in encoded["input_ids"]]
    encoded["labels"] = encoded["input_ids"].copy()
    return encoded

return dataset.map(tokenize, batched=True, remove_columns=["text"])
```

```
def build_trainer(
    model: AutoModelForCausalLM,
    tokenizer: AutoTokenizer,
    train_dataset: Dataset,
    eval_dataset: Dataset,
    config: ModelConfig,
    output_dir: str,
) -> Trainer:
    arguments kwargs = {
        "output_dir": output_dir,
        "learning_rate": config.learning_rate,
        "num_train_epochs": config.epochs,
        "per_device_train_batch_size": config.per_device_train_batch_size,
        "per_device_eval_batch_size": 1,
        "gradient_accumulation_steps": config.gradient_accumulation_steps,
        "save_strategy": "epoch",
        "logging_steps": 10,
```

```

        "report to": [],
        "disable tqdm": False,
        "bf16": config.torch_dtype == "bfloat16",
        "fp16": config.torch_dtype == "float16",
        "remove unused columns": False,
        "load best model at end": True,
        "metric for best model": "eval loss",
        "greater is better": False,
    }
    signature = inspect.signature(TrainingArguments.__init__)
    if "evaluation_strategy" in signature.parameters:
        arguments.kwargs["evaluation_strategy"] = "epoch"
    elif "eval_strategy" in signature.parameters:
        arguments.kwargs["eval_strategy"] = "epoch"
    else:
        raise RuntimeError("Installed transformers.TrainingArguments has neither evaluation_strategy nor eval_strategy.")
    arguments = TrainingArguments(**arguments.kwargs)
    trainer_kwargs = {
        "model": model,
        "args": arguments,
        "train_dataset": train_dataset,
        "eval_dataset": eval_dataset,
        "data_collator": DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False),
    }
    trainer_signature = inspect.signature(Trainer.__init__)
    if "tokenizer" in trainer_signature.parameters:
        trainer_kwargs["tokenizer"] = tokenizer
    elif "processing_class" in trainer_signature.parameters:
        trainer_kwargs["processing_class"] = tokenizer
    return Trainer(**trainer_kwargs)

train_adapter.py:
from future import import_annotations

import argparse
from pathlib import Path

import pandas as pd

from lm_steer_thesis.config import load_model_config
from lm_steer_thesis.io import read_table, write_json
from lm_steer_thesis.training import build_attribute_dataset, build_trainer, load_training_bundle

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Train a single multilingual LoRA adapter for one attribute.")
    parser.add_argument("--model-config", required=True)
    parser.add_argument("--dataset-path", required=True)
    parser.add_argument("--attribute", required=True, choices=["formal", "informal", "concise", "verbose"])
    parser.add_argument("--output-dir", required=True)
    return parser.parse_args()

def main() -> None:
    args = parse_args()
    config = load_model_config(args.model_config)
    frame = read_table(args.dataset_path)
    train_frame = frame[frame["split"] == "train"].reset_index(drop=True)
    eval_frame = frame[frame["split"] == "eval"].reset_index(drop=True)
    if train_frame.empty or eval_frame.empty:
        raise RuntimeError("Both train and eval splits are required in styled_targets.parquet")

    bundle = load_training_bundle(config)
    train_dataset = build_attribute_dataset(train_frame, bundle.tokenizer, args.attribute, config.max_seq_length)
    eval_dataset = build_attribute_dataset(eval_frame, bundle.tokenizer, args.attribute, config.max_seq_length)

    print(
        f"Training adapter='{args.attribute}' "
        f"train rows={len(train_frame[train_frame['attribute'] == args.attribute])} "
        f"eval rows={len(eval_frame[eval_frame['attribute'] == args.attribute])} "
        f"model={config.model_id}"
    )

    output_dir = Path(args.output_dir)
    output_dir.mkdir(parents=True, exist_ok=True)

    trainer = build_trainer(
        model=bundle.model,
        tokenizer=bundle.tokenizer,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        config=config,
        output_dir=str(output_dir),
    )
    trainer.train()

```

```

trainer.save_model(str(output_dir))
bundle.tokenizer.save_pretrained(output_dir)

summary = {
    "attribute": args.attribute,
    "train_rows": int(len(train_frame[train_frame["attribute"] == args.attribute])),
    "eval_rows": int(len(eval_frame[eval_frame["attribute"] == args.attribute])),
    "model_id": config.model_id,
    "use_4bit": config.use_4bit,
}
write_json(summary, output_dir / "training_summary.json")
print(f"Saved adapter to {output_dir}")

if __name__ == "__main__":
    main()

    test_conditions.py:
import unittest

from lm_steer_thesis.conditions import build_experiment_conditions

class ConditionTests(unittest.TestCase):
    def test_baseline_condition_exists(self) -> None:
        baseline = build_experiment_conditions()[0]
        self.assertEqual(baseline.name, "baseline")
        self.assertEqual(baseline.weights, {})

    def test_negative_alpha_condition_exists(self) -> None:
        names = [condition.name for condition in build_experiment_conditions()]
        self.assertIn("formal_alpha -2", names)

    def test_composition_condition_exists(self) -> None:
        names = [condition.name for condition in build_experiment_conditions()]
        self.assertIn("informal_verbose 2 2", names)

if __name__ == "__main__":
    unittest.main()

    test_filtering.py:
import unittest

from lm_steer_thesis.filtering import (
    detect_prompt_language,
    infer_domain,
    normalize_prompt_key,
    prompt_is_eligible,
    prompt_rejection_reason,
    prompt_quality_score,
    stable_prompt_id,
)

class FilteringTests(unittest.TestCase):
    def test_prompt_rejects_translation_requests(self) -> None:
        self.assertFalse(prompt_is_eligible("Translate this English paragraph into Ukrainian."))

    def test_prompt_rejects_punctuation_edit_request_in_english(self) -> None:
        self.assertFalse(prompt_is_eligible("Insert two commas into the following sentence."))

    def test_prompt_rejects_punctuation_edit_request_in_ukrainian(self) -> None:
        self.assertFalse(prompt_is_eligible("У наступному реченні вставте дві коми."))

    def test_prompt_rejects_conversion_request_in_ukrainian(self) -> None:
        self.assertFalse(prompt_is_eligible("Перетворіть наступну одиницю часу."))

    def test_prompt_rejects_trivial_factoid(self) -> None:
        self.assertFalse(prompt_is_eligible("What is the capital of Armenia?", expected_language="EN"))

    def test_prompt_rejects_short_factoid_style_question(self) -> None:
        self.assertFalse(prompt_is_eligible("In which month does the Oktoberfest traditionally begin?",
        expected_language="EN"))

    def test_prompt_rejects_grammar_check_prompt(self) -> None:
        self.assertFalse(
            prompt_is_eligible(
                "Good morning, all. Tomorrow at 7:00 p.m. works well for me. Is it a good message to send to my new group? And does this message contain grammar issues?",
                expected_language="EN",
            )
        )

    def test_prompt_rejects_wrong_language(self) -> None:

```

```

        self.assertEqual(
            prompt_rejection_reason("Berikan contoh ayat untuk peribahasa adu lengan:", expected_language="EN"),
            "language:unknown",
        )

    def test_prompt_rejects_non_latin_script_in_english_pool(self) -> None:
        self.assertEqual(
            prompt_rejection_reason("ΛΗΥ Ω·Η·ΠΗ·Π ΤΡΦ ΣΟΑΗ ΗΤ:", expected_language="EN"),
            "script:non latin",
        )

    def test_prompt_rejects_mixed_non_latin_english_prompt(self) -> None:
        self.assertEqual(
            prompt_rejection_reason("write a funny stanze/ شعر in urdu.", expected_language="EN"),
            "script:mixed non latin",
        )

    def test_prompt_keeps_regular_generation_prompt(self) -> None:
        self.assertTrue(prompt_is_eligible("Write a short review of a hotel for a travel blog.", expected_language="EN"))

    def test_stable_prompt_id_is_deterministic(self) -> None:
        prompt = "Write a polite email to a colleague."
        self.assertEqual(stable_prompt_id("EN", prompt), stable_prompt_id("EN", prompt))

    def test_domain_inference(self) -> None:
        self.assertEqual(infer_domain("Write an email to a colleague about a meeting."), "email")

    def test_language_detection_for_ukrainian_text(self) -> None:
        self.assertEqual(detect_prompt_language("Поясни, як працює сонячна батарея."), "uk")

    def test_quality_score_prefers_open_ended_prompt(self) -> None:
        self.assertGreater(
            prompt_quality_score("Explain how community composting can reduce waste in a city.", "EN"),
            prompt_quality_score("What is the capital of Armenia?", "EN"),
        )

    def test_normalize_prompt_key_collapses_spacing(self) -> None:
        self.assertEqual(normalize_prompt_key("Write an email "), "write an email")

if __name__ == "__main__":
    unittest.main()

from test_openai_data.py
from future import annotations
import unittest

from lm_steer_thesis.openai_data import (
    build_batch_request_line,
    build_attribute_batch_request_line,
    build_neutral_batch_request_line,
    build_repair_batch_request_line,
    build_rewrite_batch_request_line,
    bundle_to_rows,
    extract_batch_response_body,
    neutral_text_to_row,
    parse_attribute_batch_custom_id,
    parse_batch_custom_id,
    parse_repair_batch_custom_id,
    parse_rewrite_bundle,
    parse_styled_bundle,
    single_attribute_to_row,
    word_budget_bounds,
)

class OpenAIDataTests(unittest.TestCase):
    def test_build_batch_request_line_targets_responses_api(self) -> None:
        entry = build_batch_request_line(
            prompt_id="EN 0001",
            language="EN",
            prompt="Explain photosynthesis in simple terms.",
            model="gpt-5-mini",
            max_output_tokens=1800,
            reasoning_effort="minimal",
        )

        self.assertEqual(entry["custom id"], "EN 0001")
        self.assertEqual(entry["url"], "/v1/responses")
        self.assertEqual(entry["body"]["model"], "gpt-5-mini")
        self.assertEqual(entry["body"]["text"]["format"]["type"], "json schema")

    def test_build_neutral_batch_request_line_uses_stage_custom_id(self) -> None:
        entry = build_neutral_batch_request_line(

```

```

        prompt_id="EN 0001",
        language="EN",
        prompt="Explain photosynthesis in simple terms.",
        model="gpt-5-mini",
        max_output_tokens=400,
        reasoning_effort="minimal",
    )

    self.assertEqual(entry["custom_id"], "neutral::EN 0001")
    self.assertEqual(entry["url"], "/v1/responses")
    self.assertNotIn("text", entry["body"])

def test build rewrite batch request line uses stage custom id(self) -> None:
    entry = build_rewrite_batch_request_line(
        prompt_id="UA 0001",
        language="UA",
        prompt="Поясни різницю між сонцем і зіркою.",
        neutral_text="Сонце є зіркою.",
        model="gpt-5-mini",
        max_output_tokens=800,
        reasoning_effort="minimal",
    )

    self.assertEqual(entry["custom_id"], "rewrite::UA 0001")
    self.assertEqual(entry["body"]["text"]["format"]["type"], "json_schema")
    self.assertEqual(entry["body"]["metadata"]["stage"], "rewrite")

def test parse batch custom id round trips(self) -> None:
    stage, prompt_id = parse_batch_custom_id("rewrite::EN 0001")
    self.assertEqual(stage, "rewrite")
    self.assertEqual(prompt_id, "EN 0001")

def test build repair batch request line uses attribute custom id(self) -> None:
    entry = build_repair_batch_request_line(
        prompt_id="EN 0001",
        language="EN",
        prompt="Explain photosynthesis in simple terms.",
        neutral_text="Photosynthesis is how plants make food from sunlight.",
        attribute="verbose",
        model="gpt-5-mini",
        max_output_tokens=1200,
        reasoning_effort="minimal",
    )

    self.assertEqual(entry["custom_id"], "repair::EN 0001::verbose")
    self.assertEqual(entry["body"]["metadata"]["requested_attributes"], "verbose")

def test build attribute batch request line uses attribute custom id(self) -> None:
    entry = build_attribute_batch_request_line(
        prompt_id="EN 0001",
        language="EN",
        prompt="Explain photosynthesis in simple terms.",
        neutral_text="Photosynthesis is how plants make food from sunlight.",
        attribute="formal",
        model="gpt-5-mini",
        max_output_tokens=900,
        reasoning_effort="minimal",
    )

    self.assertEqual(entry["custom_id"], "attribute::EN 0001::formal")
    self.assertEqual(entry["body"]["metadata"]["requested_attributes"], "formal")

def test parse repair batch custom id round trips(self) -> None:
    prompt_id, attribute = parse_repair_batch_custom_id("repair::UA 0001::concise")
    self.assertEqual(prompt_id, "UA 0001")
    self.assertEqual(attribute, "concise")

def test parse attribute batch custom id round trips(self) -> None:
    prompt_id, attribute = parse_attribute_batch_custom_id("attribute::UA 0001::concise")
    self.assertEqual(prompt_id, "UA 0001")
    self.assertEqual(attribute, "concise")

def test parse styled bundle from response body(self) -> None:
    response_body = {
        "output": [
            {
                "type": "message",
                "content": [
                    {
                        "type": "output text",
                        "text": (
                            '{"prompt_id": "EN 0001", "language": "EN", "neutral": "Neutral answer.", '
                            '"formal": "Formal answer.", "informal": "Informal answer.", '
                            '"concise": "Short answer.", "verbose": "Longer answer."}'
                        ),
                    }
                ],
            }
        ],
    }

```

```

    },
    ],
}

bundle = parse_styled_bundle(response_body)
self.assertEqual(bundle["language"], "EN")
self.assertEqual(bundle["concise"], "Short answer.")

def test_extract_batch_response_body_prefers_nested_body(self) -> None:
    batch_line = {
        "custom_id": "EN 0001",
        "response": {
            "status_code": 200,
            "body": {"id": "resp 123", "output": []},
        },
    },
    response_body = extract_batch_response_body(batch_line)
    self.assertEqual(response_body["id"], "resp 123")

def test_parse_rewrite_bundle_from_response_body(self) -> None:
    response_body = {
        "output": [
            {
                "type": "message",
                "content": [
                    {
                        "type": "output_text",
                        "text": (
                            {"prompt_id": "EN 0001", "language": "EN", "formal": "Formal answer.",
                             "informal": "Informal answer.", "concise": "Short answer.",
                             "verbose": "Longer answer."},
                        ),
                    },
                ],
            },
        ],
    },
    ],
}

bundle = parse_rewrite_bundle(response_body)
self.assertEqual(bundle["language"], "EN")
self.assertEqual(bundle["verbose"], "Longer answer.")

def test_word_budget_bounds_follow_expected_ranges(self) -> None:
    bounds = word_budget_bounds("one two three four five six seven eight ten")
    self.assertEqual(bounds["neutral_words"], 9)
    self.assertEqual(bounds["concise_min"], 4)
    self.assertEqual(bounds["concise_max"], 6)
    self.assertEqual(bounds["verbose_min"], 12)
    self.assertEqual(bounds["verbose_max"], 17)

def test_bundle_to_rows_preserves_training_shape(self) -> None:
    prompt_record = {
        "prompt_id": "UA 0001",
        "language": "UA",
        "prompt": "Поясни різницю між сонцем і зіркою.",
        "split": "train",
        "domain": "explanation",
    }
    bundle = {
        "prompt_id": "UA 0001",
        "language": "UA",
        "neutral": "Сонце є зіркою.",
        "formal": "Сонце належить до класу зірок.",
        "informal": "Сонце, по суті, теж зірка.",
        "concise": "Сонце - це зірка.",
        "verbose": "Сонце є найближчою до Землі зіркою, тому ми бачимо його інакше.",
    }

    rows = bundle_to_rows(prompt_record, bundle, generator_model="gpt-5-mini")
    self.assertEqual(len(rows), 4)
    self.assertEqual(rows[0]["neutral_attempts"], 1)
    self.assertIn("example_id", rows[0])

def test_neutral_text_to_row_preserves_prompt_metadata(self) -> None:
    prompt_record = {
        "prompt_id": "EN 0001",
        "language": "EN",
        "prompt": "Who wrote Wealth of Nations?",
        "split": "eval",
        "domain": "general",
    }
    row = neutral_text_to_row(
        prompt_record,
        "The Wealth of Nations was written by Adam Smith.",
        generator_model="gpt-5-mini",
    )

```

```

    )

    self.assertEqual(row["prompt id"], "EN 0001")
    self.assertEqual(row["split"], "eval")
    self.assertEqual(row["neutral attempts"], 1)

def test single attribute to row preserves training shape(self) -> None:
    prompt_record = {
        "prompt id": "EN 0001",
        "language": "EN",
        "prompt": "Explain photosynthesis in simple terms.",
        "split": "train",
        "domain": "explanation",
    }
    bundle = {
        "prompt id": "EN 0001",
        "language": "EN",
        "formal": "Photosynthesis is the biochemical process by which plants convert light into stored energy.",
    }
    row = single_attribute_to_row(
        prompt_record,
        bundle,
        attribute="formal",
        neutral_text="Photosynthesis is how plants turn sunlight into food.",
        generator_model="gpt-5-mini",
    )

    self.assertEqual(row["attribute"], "formal")
    self.assertEqual(row["prompt id"], "EN 0001")
    self.assertEqual(row["rewrite attempts"], 1)

if __name__ == "__main__":
    unittest.main()

    test_run_experiments.py:
import unittest

import pandas as pd

from infer.run_experiments import batched_frame, select_conditions, select_eval_prompts
from lm_steer_thesis.conditions import build_experiment_conditions

class RunExperimentsTests(unittest.TestCase):
    def test select eval prompts samples per language(self) -> None:
        frame = pd.DataFrame(
            [
                {"prompt id": "en1", "language": "EN", "split": "eval"},
                {"prompt id": "en2", "language": "EN", "split": "eval"},
                {"prompt id": "ua1", "language": "UA", "split": "eval"},
                {"prompt id": "ua2", "language": "UA", "split": "eval"},
                {"prompt id": "en_train", "language": "EN", "split": "train"},
            ]
        )
        selected = select_eval_prompts(frame, sample_size_per_language=1)
        self.assertEqual(len(selected), 2)
        self.assertEqual(set(selected["language"]), {"EN", "UA"})

    def test select conditions by kind and name(self) -> None:
        conditions = build_experiment_conditions()
        selected = select_conditions(
            conditions,
            condition_names=["baseline", "formal alpha 2"],
            condition_kinds=["baseline", "continuous"],
        )
        self.assertEqual([condition.name for condition in selected], ["baseline", "formal alpha 2"])

    def test batched frame(self) -> None:
        frame = pd.DataFrame([{"value": index} for index in range(5)])
        batches = batched_frame(frame, batch_size=2)
        self.assertEqual([len(batch) for batch in batches], [2, 2, 1])
        self.assertEqual(batches[0][0]["value"].tolist(), [0, 1])
        self.assertEqual(batches[-1][0]["value"].tolist(), [4])

if __name__ == "__main__":
    unittest.main()

    export_human_eval.py:
from future import annotations

import argparse

from lm_steer_thesis.io import read_table, write_table

```

```
TARGET_CONDITIONS = {
    "baseline",
    "formal alpha 2",
    "informal alpha 2",
    "concise alpha 2",
    "verbose alpha 2",
    "formal concise 1 1",
    "informal verbose 1 1",
}
```

```
def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Export a compact human-evaluation sheet.")
    parser.add_argument("--scored-path", required=True)
    parser.add_argument("--output-path", required=True)
    parser.add_argument("--sample-size-per-language", type=int, default=20)
    parser.add_argument("--seed", type=int, default=42)
    return parser.parse_args()
```

```
def main() -> None:
    args = parse_args()
    frame = read_table(args.scored_path)
    frame = frame[frame["condition name"].isin(TARGET_CONDITIONS)].copy()

    prompt_ids = []
    for language in ("EN", "UA"):
        available = frame[frame["language"] == language][["prompt id"]].drop_duplicates()
        language_prompt_ids = (
            available
            .sample(n=min(args.sample_size_per_language, len(available)), random_state=args.seed)
            .tolist()
        )
        prompt_ids.extend(language_prompt_ids)

    export = frame[frame["prompt id"].isin(prompt_ids)].copy()
    export["attribute match"] = ""
    export["fluency"] = ""
    export["adequacy"] = ""
    export["notes"] = ""
    write_table(export, args.output_path)
    print(f"Saved human evaluation sheet to {args.output_path}")
```

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

```
from manual_review.py:
from future import annotations

import argparse
import csv
import hashlib
import json
import math
import shutil
import sys
import textwrap
from datetime import datetime, timezone
from pathlib import Path
from typing import Any
```

```
import pandas as pd
```

```
try:
    from lm_steer_thesis.io import ensure_parent_dir, read_table
except ModuleNotFoundError: # pragma: no cover - convenience for direct repo-root execution
    sys.path.append(str(Path(__file__).resolve().parents[1] / "src"))
    from lm_steer_thesis.io import ensure_parent_dir, read_table
```

```
DISPLAY_COLUMNS = [
    "prompt id",
    "language",
    "domain",
    "condition name",
    "condition kind",
    "weights",
    "prompt",
    "generation",
    "neutral text",
]
```

```

OUTPUT_COLUMNS = [
    "review_id",
    "reviewed_at",
    "action",
    "steering_score",
    "notes",
    *DISPLAY_COLUMNS,
]

```

```

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description=(
            "Manually review model generations from a generations/scored table and "
            "append 1-10 steering scores to a CSV."
        )
    )
    parser.add_argument("--input-path", required=True, help="Input .parquet or .csv table with generated rows.")
    parser.add_argument("--output-csv", required=True, help="CSV file where review decisions are appended.")
    parser.add_argument(
        "--sample-count",
        type=int,
        help="Number of scored items to collect in this session. Omit to continue until rows run out or you quit.",
    )
    parser.add_argument("--seed", type=int, help="Random seed for reproducible review order.")
    parser.add_argument(
        "--condition",
        action="append",
        help="Only review this condition name. Can be repeated or comma-separated.",
    )
    parser.add_argument(
        "--language",
        action="append",
        help="Only review this language, e.g. EN or UA. Can be repeated or comma-separated.",
    )
    return parser.parse_args()

```

```

def split_filter_values(values: list[str] | None) -> set[str] | None:
    if not values:
        return None
    parsed = {part.strip() for value in values for part in value.split(",") if part.strip()}
    return parsed or None

```

```

def require_columns(frame: pd.DataFrame, columns: list[str], path: str) -> None:
    missing = [column for column in columns if column not in frame.columns]
    if missing:
        joined = ", ".join(missing)
        raise ValueError(f"{path} is missing required column(s): {joined}")

```

```

def normalize_value(value: Any) -> str:
    if value is None:
        return ""
    if isinstance(value, float) and math.isnan(value):
        return ""
    if isinstance(value, dict):
        return json.dumps(value, sort_keys=True, ensure_ascii=False)
    return str(value)

```

```

def make_review_id(row: pd.Series) -> str:
    key_columns = [
        "prompt_id",
        "language",
        "condition_name",
        "condition_kind",
        "weights",
        "prompt",
        "generation",
    ]
    payload = {column: normalize_value(row.get(column, "")) for column in key_columns}
    encoded = json.dumps(payload, sort_keys=True, ensure_ascii=False).encode("utf-8")
    return hashlib.sha256(encoded).hexdigest()[16]

```

```

def load_reviewed_ids(output_csv: Path) -> set[str]:
    if not output_csv.exists():
        return set()
    with output_csv.open("r", encoding="utf-8", newline="") as handle:
        reader = csv.DictReader(handle)
        if not reader.fieldnames or "review_id" not in reader.fieldnames:

```

```

        return set()
    return {row["review_id"] for row in reader if row.get("review_id")}

def format_block(title: str, value: Any, width: int) -> str:
    text = normalize_value(value).strip() or "<empty>"
    wrapped = textwrap.fill(
        text,
        width=width,
        replace_whitespace=False,
        break_long_words=False,
        break_on_hyphens=False,
    )
    return f"\n{title}\n{'-' * len(title)}\n{wrapped}"

def print_item(row: pd.Series, index: int, total: int) -> None:
    width = max(60, min(shutil.get_terminal_size((100, 24)).columns, 120))
    weights = normalize_value(row.get("weights", ""))
    header = (
        f"[{index}/{total}] "
        f"review_id={row['review_id']} | "
        f"language={normalize_value(row.get('language', ''))} | "
        f"condition={normalize_value(row.get('condition_name', ''))} | "
        f"kind={normalize_value(row.get('condition_kind', ''))} | "
        f"weights={weights or '{}'}"
    )
    print("\n" + "=" * min(len(header), width))
    print(textwrap.fill(header, width=width, break_long_words=False, break_on_hyphens=False))
    if "domain" in row.index and normalize_value(row.get("domain", "")):
        print(f"domain={normalize_value(row.get('domain', ''))}")
    print(format_block("PROMPT", row.get("prompt", ""), width))
    print(format_block("GENERATION", row.get("generation", ""), width))
    if "neutral_text" in row.index and normalize_value(row.get("neutral_text", "")):
        print(format_block("NEUTRAL CONTEXT", row.get("neutral_text", ""), width))

def prompt_decision() -> tuple[str, int | None, str]:
    while True:
        try:
            raw = input("\nScore 1-10, s=skip, q=quit: ").strip()
        except EOFError:
            print("\nInput closed; quitting without saving this item.")
            return "quit", None, ""
        lowered = raw.lower()
        if lowered in {"q", "quit", "exit"}:
            return "quit", None, ""
        if lowered in {"s", "skip"}:
            try:
                note = input("Optional skip note: ").strip()
            except EOFError:
                note = ""
            return "skip", None, note
        try:
            score = int(raw)
        except ValueError:
            print("Please enter an integer score from 1 to 10, s to skip, or q to quit.")
            continue
        if 1 <= score <= 10:
            try:
                note = input("Optional note: ").strip()
            except EOFError:
                note = ""
            return "score", score, note
        print("Score must be between 1 and 10.")

def output_row(row: pd.Series, action: str, score: int | None, notes: str) -> dict[str, str]:
    payload = {
        "review_id": normalize_value(row["review_id"]),
        "reviewed_at": datetime.now(timezone.utc).isoformat(),
        "action": action,
        "steering_score": "" if score is None else str(score),
        "notes": notes,
    }
    for column in DISPLAY_COLUMNS:
        payload[column] = normalize_value(row.get(column, ""))
    return payload

def append_review(output_csv: Path, row: pd.Series, action: str, score: int | None, notes: str) -> None:
    target = ensure_parent_dir(output_csv)
    write_header = not target.exists() or target.stat().st_size == 0
    with target.open("a", encoding="utf-8", newline="") as handle:

```

```

writer = csv.DictWriter(handle, fieldnames=OUTPUT_COLUMNS)
if write_header:
    writer.writeheader()
writer.writerow(output_row(row, action, score, notes))
handle.flush()

def filtered_frame(frame: pd.DataFrame, args: argparse.Namespace) -> pd.DataFrame:
    require_columns(frame, ["prompt", "generation", "condition name", "condition kind", "weights"], args.input_path)
    conditions = split_filter_values(args.condition)
    languages = split_filter_values(args.language)
    if conditions is not None:
        frame = frame[frame["condition name"].isin(conditions)]
    if languages is not None:
        require_columns(frame, ["language"], args.input_path)
        frame = frame[frame["language"].isin(languages)]
    return frame.copy()

def main() -> None:
    args = parse_args()
    if args.sample_count is not None and args.sample_count <= 0:
        raise ValueError("--sample-count must be greater than zero when provided.")

    frame = filtered_frame(read_table(args.input_path), args)
    frame["review_id"] = frame.apply(make_review_id, axis=1)

    output_csv = Path(args.output_csv)
    reviewed_ids = load_reviewed_ids(output_csv)
    remaining = frame[~frame["review_id"].isin(reviewed_ids)].copy()
    if remaining.empty:
        print("No unreviewed rows remain for the selected filters.")
        return

    remaining = remaining.sample(frac=1, random_state=args.seed).reset_index(drop=True)
    target_scores = args.sample_count
    saved_scores = 0
    saved_decisions = 0

    print(
        f"Loaded {len(frame)} selected row(s); "
        f"{len(reviewed_ids)} already reviewed/skipped; {len(remaining)} remaining."
    )
    for position, (row, row) in enumerate(remaining.iterrows(), start=1):
        if target_scores is not None and saved_scores >= target_scores:
            break
        print_item(row, position, len(remaining))
        action, score, notes = prompt_decision()
        if action == "quit":
            break
        append_review(output_csv, row, action, score, notes)
        saved_decisions += 1
        if action == "score":
            saved_scores += 1
            print(f"Saved score {score} to {output_csv}")
        else:
            print(f"Saved skip to {output_csv}")

    print(f"\nSession complete. Saved {saved_scores} score(s) and {saved_decisions - saved_scores} skip(s).")

if __name__ == "__main__":
    main()

plot_results.py:
from future import annotations

import argparse
from pathlib import Path

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from lm_steer_thesis.io import read_table

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Plot alpha versus steering metrics.")
    parser.add_argument("--scored-path", required=True)
    parser.add_argument("--output-dir", required=True)
    return parser.parse_args()

```

```

def parse_alpha(condition_name: str) -> int | None:
    if "alpha" not in condition_name:
        return None
    return int(condition_name.split("alpha ")[-1])

def parse_family(condition_name: str) -> str | None:
    for family in ("formal", "informal", "concise", "verbose"):
        if condition_name.startswith(family):
            return family
    return None

def line_plot(frame: pd.DataFrame, language: str, metric: str, output_dir: Path) -> None:
    subset = frame[(frame["language"] == language) & (frame["condition_name"].str.contains("alpha"))].copy()
    subset["alpha"] = subset["condition_name"].apply(parse_alpha)
    subset["family"] = subset["condition_name"].apply(parse_family)
    subset = subset.dropna(subset=[metric])
    if subset.empty:
        return
    subset = (
        subset.groupby(["language", "family", "alpha"], as_index=False)
        .agg(metric_value=(metric, "mean"))
        .sort_values(["family", "alpha"])
    )
    palette = {
        "formal": "#111111",
        "informal": "#4f4f4b",
        "concise": "#0f5d4c",
        "verbose": "#7c8a8b",
    }
    plt.figure(figsize=(7.4, 4.2))
    ax = sns.lineplot(
        data=subset,
        x="alpha",
        y="metric_value",
        hue="family",
        style="family",
        markers=True,
        dashes=False,
        linewidth=2.2,
        markersize=7,
        palette=palette,
        errorbar=None,
    )
    ax.set_title(f'{language} {{"Formality" if metric == "formality score" else "Length ratio"}', fontsize=15, pad=10)
    ax.set_xlabel("alpha")
    ax.set_ylabel(metric)
    ax.grid(axis="y", color="#d9d9d2", linewidth=0.7, alpha=0.55)
    ax.grid(axis="x", visible=False)
    legend = ax.legend(title=None, frameon=False, loc="best")
    if legend is not None:
        for text in legend.get_texts():
            text.set_fontsize(10)
    sns.despine(ax=ax, left=False, bottom=False)
    plt.tight_layout()
    plt.savefig(output_dir / f'{language.lower()} {metric} vs alpha.png', dpi=200)
    plt.close()

def composition_summary(frame: pd.DataFrame, output_dir: Path) -> None:
    subset = frame[frame["condition_kind"] == "composition"].copy()
    if subset.empty:
        return
    summary = subset.groupby(["language", "condition_name"], as_index=False).agg(
        semantic_similarity=("semantic similarity", "mean"),
        length_ratio=("length ratio", "mean"),
        language_ok=("language ok", "mean"),
    )
    summary.to_csv(output_dir / "composition summary.csv", index=False)

def main() -> None:
    args = parse_args()
    frame = read_table(args.scores_path)
    output_dir = Path(args.output_dir)
    output_dir.mkdir(parents=True, exist_ok=True)

    for language in ("EN", "UA"):
        line_plot(frame, language, "formality score", output_dir)
        line_plot(frame, language, "length ratio", output_dir)

    composition_summary(frame, output_dir)
    print(f'Saved plots to {output_dir}')

```

```

if __name__ == "__main__":
    main()
score_outputs.py;
from future import annotations

import argparse
from pathlib import Path

from lm_steer_thesis.evaluation import build_embedding_model, build_en_formality_scorer, build_metric_frame
from lm_steer_thesis.io import read_table, write_json, write_table

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Score generation outputs with automatic metrics.")
    parser.add_argument("--generations-path", required=True)
    parser.add_argument("--scored-output-path", required=True)
    parser.add_argument("--skip-en-formality", action="store_true")
    parser.add_argument("--ua-formality-model")
    parser.add_argument("--embedding-model", default="sentence-transformers/paraphrase-multilingual-mpnet-base-v2")
    return parser.parse_args()

def main() -> None:
    args = parse_args()
    generations = read_table(args.generations_path)
    embedding_model = build_embedding_model(args.embedding_model)
    english_formality_classifier = None if args.skip_en_formality else build_en_formality_scorer()
    ua_formality_model = args.ua_formality_model

    scored = build_metric_frame(
        generations=generations,
        embedding_model=embedding_model,
        english_formality_classifier=english_formality_classifier,
        ukrainian_formality_model_id=ua_formality_model,
    )
    write_table(scored, args.scored_output_path)

    summary = {
        "rows": int(len(scored)),
        "language_ok_rate": float(scored["language_ok"].mean()),
        "semantic_similarity_mean": float(scored["semantic_similarity"].mean()),
        "invalid_output_rate": float(scored["invalid_output"].mean()),
    }
    write_json(summary, Path(args.scored_output_path).with_name("scoring_summary.json"))
    print(f"Saved scored outputs to {args.scored_output_path}")

if __name__ == "__main__":
    main()
train_ua_formality_classifier.py;
from future import annotations

import argparse
import inspect
from pathlib import Path

from datasets import load_dataset
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    DataCollatorWithPadding,
    Trainer,
    TrainingArguments,
)

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Train a Ukrainian formality classifier for evaluation.")
    parser.add_argument("--dataset-id", default="ukr-detect/ukr-formality-dataset-translated-gyafc")
    parser.add_argument("--text-column", default="text")
    parser.add_argument("--label-column", default="label")
    parser.add_argument("--model-id", default="xlm-roberta-base")
    parser.add_argument("--output-dir", default="artifacts/models/ua_formality_classifier")
    return parser.parse_args()

def main() -> None:
    args = parse_args()
    dataset = load_dataset(args.dataset_id, split="train", token=True).train_test_split(test_size=0.1, seed=42)
    tokenizer = AutoTokenizer.from_pretrained(args.model_id, token=True)
    model = AutoModelForSequenceClassification.from_pretrained(args.model_id, num_labels=2, token=True)

    def tokenize(batch):
        return tokenizer(batch[args.text_column], truncation=True, max_length=256)

```

```

train_dataset = dataset["train"].map(tokenize, batched=True)
eval_dataset = dataset["test"].map(tokenize, batched=True)

trainer = Trainer(
    model=model,
    args=TrainingArguments(
        **{
            "output_dir": args.output_dir,
            "num_train_epochs": 2,
            "learning_rate": 2e-5,
            "per_device_train_batch_size": 8,
            "per_device_eval_batch_size": 8,
            "save_strategy": "epoch",
            "report_to": [],
            **{
                "evaluation_strategy": "epoch"
                if "evaluation_strategy" in inspect.signature(TrainingArguments.__init__).parameters
                else {"eval_strategy": "epoch"}
            }
        },
    ),
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=DataCollatorWithPadding(tokenizer=tokenizer),
    **{
        "tokenizer": tokenizer
        if "tokenizer" in inspect.signature(Trainer.__init__).parameters
        else {"processing_class": tokenizer}
    },
)
trainer.train()
trainer.save_model(args.output_dir)
tokenizer.save_pretrained(Path(args.output_dir))
print(f"Saved classifier to {args.output_dir}")

```

```

if __name__ == "__main__":
    main()

```

```

run_experiments.py:
from future import annotations

```

```

import argparse
from pathlib import Path

```

```

import pandas as pd
from peft import PeftModel
from tqdm.auto import tqdm

```

```

from lm_steer_thesis.conditions import SteeringCondition, build_experiment_conditions
from lm_steer_thesis.config import load_model_config
from lm_steer_thesis.generation import LoadedGenerator, generate_texts, load_generator
from lm_steer_thesis.io import read_table, write_json, write_table
from lm_steer_thesis.prompting import build_neutral_messages

```

```

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(description="Run baseline, continuous, and composition steering experiments.")
    parser.add_argument("--model-config", required=True)
    parser.add_argument("--prompts-path", required=True)
    parser.add_argument("--references-path", required=True)
    parser.add_argument("--adapters-root", required=True)
    parser.add_argument("--output-path", required=True)
    parser.add_argument("--temperature", type=float, default=0.2)
    parser.add_argument("--top-p", type=float, default=0.9)
    parser.add_argument("--max-new-tokens", type=int, default=256)
    parser.add_argument("--batch-size", type=int, default=8)
    parser.add_argument("--sample-size-per-language", type=int)
    parser.add_argument("--condition-names", nargs="+")
    parser.add_argument("--condition-kinds", nargs="+", choices=["baseline", "discrete", "continuous", "composition"])
    return parser.parse_args()

```

```

def select_eval_prompts(prompts: pd.DataFrame, sample_size_per_language: int | None) -> pd.DataFrame:
    eval_prompts = prompts[prompts["split"] == "eval"].reset_index(drop=True)
    if sample_size_per_language is None:
        return eval_prompts
    sampled_frames = []
    for language in ("EN", "UA"):
        language_subset = eval_prompts[eval_prompts["language"] == language]
        if language_subset.empty:
            continue

```

```

        sampled_frames.append(
            language_subset.sample(
                n=min(sample_size_per_language, len(language_subset)),
                random_state=42,
            )
        )
    if not sampled_frames:
        return eval_prompts.iloc[0:0].copy()
    return pd.concat(sampled_frames, ignore_index=True)

def select_conditions(
    conditions: list[SteeringCondition],
    condition_names: list[str] | None,
    condition_kinds: list[str] | None,
) -> list[SteeringCondition]:
    selected = conditions
    if condition_kinds:
        allowed_kinds = set(condition_kinds)
        selected = [condition for condition in selected if condition.kind in allowed_kinds]
    if condition_names:
        allowed_names = set(condition_names)
        selected = [condition for condition in selected if condition.name in allowed_names]
    return selected

def batched_frame(frame: pd.DataFrame, batch_size: int) -> list[pd.DataFrame]:
    if batch_size <= 0:
        raise ValueError("batch_size must be positive")
    return [frame.iloc[start : start + batch_size].reset_index(drop=True) for start in range(0, len(frame), batch_size)]

def build_adapter_map(adapters_root: Path) -> dict[str, str]:
    return {attribute: str(adapters_root / attribute) for attribute in ("formal", "informal", "concise", "verbose")}

def attach_adapters(generator: LoadedGenerator, adapters_root: Path) -> PeftModel:
    adapter_paths = build_adapter_map(adapters_root)
    if not Path(adapter_paths["formal"]).exists():
        raise FileNotFoundError(f"Could not find adapter directory {adapter_paths['formal']}")
    peft_model = PeftModel.from_pretrained(generator.model, adapter_paths["formal"], adapter_name="formal",
is trainable=False)
    for adapter_name, adapter_path in adapter_paths.items():
        if adapter_name == "formal":
            continue
        peft_model.load_adapter(adapter_path, adapter_name=adapter_name, is_trainable=False)
    return peft_model

def activate_condition(model: PeftModel, condition: SteeringCondition) -> None:
    if not condition.weights:
        model.disable_adapter()
        return
    weighted_name = f"tmp_{condition.name}"
    if weighted_name in model.peft_config:
        model.set_adapter(weighted_name)
        return
    model.add_weighted_adapter(
        adapters=list(condition.weights.keys()),
        weights=list(condition.weights.values()),
        adapter_name=weighted_name,
        combination_type="linear",
    )
    model.set_adapter(weighted_name)

def main() -> None:
    args = parse_args()
    config = load_model_config(args.model_config)
    prompts = read_table(args.prompts_path)
    prompts = select_eval_prompts(prompts, args.sample_size_per_language)
    references = read_table(args.references_path)
    reference_lookup = (
        references[references["split"] == "eval"][["prompt_id", "neutral_text"]]
        .drop_duplicates(subset=["prompt_id"])
        .set_index("prompt_id")["neutral_text"]
        .to_dict()
    )

    generator = load_generator(config)
    peft_model = attach_adapters(generator, Path(args.adapters_root))
    generator.model = peft_model

    rows: list[dict[str, object]] = []

```

```

conditions = select_conditions(
    build_experiment_conditions(),
    args.condition_names,
    args.condition_kinds,
)
if not conditions:
    raise RuntimeError("No steering conditions selected.")
if prompts.empty():
    raise RuntimeError("No eval prompts selected.")
total_steps = len(conditions) * len(prompts)
progress = tqdm(total=total_steps, desc="Running steering experiments", unit="generation")
for condition in conditions:
    activate_condition(peft_model, condition)
    for prompt_batch in batched_frame(prompts, args.batch_size):
        messages_batch = [build_neutral_messages(prompt) for prompt in prompt_batch["prompt"]]
        generations = generate_texts(
            generator,
            messages_batch,
            temperature=args.temperature,
            top_p=args.top_p,
            max_new_tokens=args.max_new_tokens,
        )
    for prompt_row, generation in zip(prompt_batch.itertuples(index=False), generations):
        rows.append(
            {
                "prompt_id": prompt_row.prompt_id,
                "language": prompt_row.language,
                "prompt": prompt_row.prompt,
                "domain": prompt_row.domain,
                "condition_name": condition.name,
                "condition_kind": condition.kind,
                "weights": str(condition.weights),
                "generation": generation,
                "neutral_text": reference_lookup.get(prompt_row.prompt_id, ""),
            }
        )
    progress.update(1)
    progress.set_postfix(condition=condition.name, language=prompt_row.language)
progress.close()

frame = pd.DataFrame(rows)
write_table(frame, args.output_path)
summary = {
    "rows": int(len(frame)),
    "conditions": frame["condition_name"].nunique(),
    "prompts": frame["prompt_id"].nunique(),
    "output_path": args.output_path,
}
write_json(summary, Path(args.output_path).with_name("generation_summary.json"))
print(f"Saved generations to {args.output_path}")

```

```

if __name__ == "__main__":
    main()

```

ДОДАТОК Б Презентація



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

ПАРАМЕТРИЧНЕ КЕРУВАННЯ ВЕЛИКИМИ МОВНИМИ МОДЕЛЯМИ

Бакалаврська дипломна робота

виконав: студент групи ТВ-23
Гайдук Максим Олександрович

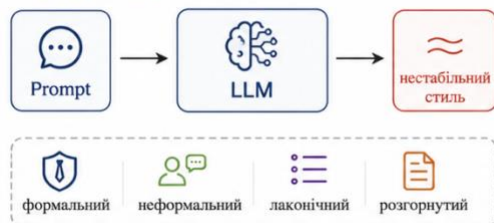
керівник:
Стативка Юрій Іванович

2026

Актуальність теми

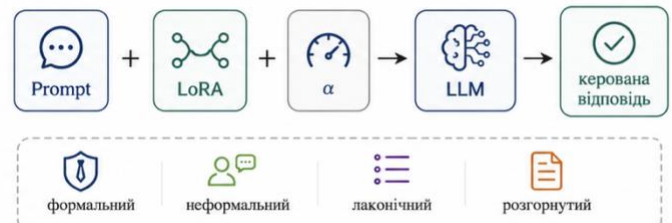
Проблема

- великі мовні моделі не завжди стабільно дотримуються потрібного стилю відповіді;
- один і той самий зміст може вимагати різного стилю подання;
- prompt engineering нестабільний і займає контекст;
- повне донавчання моделі є ресурсно дорогим.



Ідея роботи

- керувати стилем відповіді через LoRA-адаптери;
- змінювати інтенсивність стилю через вагу адаптера α ;
- підтримати українську та англійську мови;
- отримати компактне та гнучке рішення без повного fine-tuning.



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Постановка задачі

Мета роботи

реалізувати та експериментально перевірити параметричне керування стилем відповідей BMM українською та англійською мовами.

Об'єкт дослідження

процес генерації текстових відповідей великими мовними моделями.

Предмет дослідження

методи адаптерного керування стилем, довжиною та формальністю відповідей.

Основні завдання

1. Проаналізувати існуючі підходи до керування генерацією.
2. Підготувати двомовний набір промптів і цільових відповідей.
3. Навчити LoRA-адаптери для окремих стилів.
4. Реалізувати генерацію з різними вагами та комбінаціями адаптерів.
5. Оцінити якість результатів автоматичними метриками.
6. Створити вебінтерфейс для перегляду експериментів.



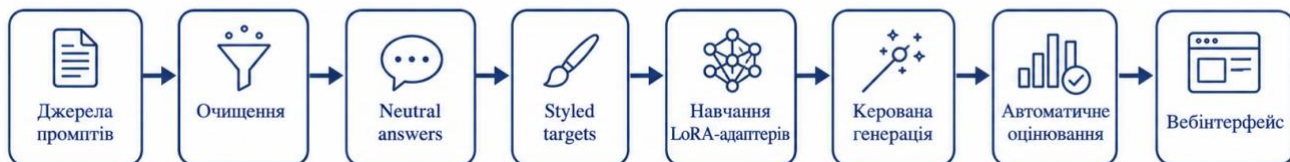
Аналіз предметної області

Підхід	Переваги	Обмеження
Prompt engineering	простий у використанні	нестабільний, займає контекст
Full fine-tuning	суттєво змінює поведінку моделі	висока обчислювальна вартість
RLHF / instruction tuning	добре вирівнює поведінку моделі	складна організація, потреба у feedback-даних
Activation steering	пряме втручання у внутрішні представлення	складніша інтеграція в прикладний пайплайн
LoRA / PEFT	компактні адаптери, менша вартість навчання, можливість комбінування	потребує якісних даних і налаштування



Проектування системи

LM-Steer Thesis Pipeline



Функціональні вимоги

- підготовка EN/UA промптів;
- навчання 4 стильових адаптерів;
- генерація відповідей у різних умовах;
- обчислення метрик якості;
- перегляд результатів у браузері.

Нефункціональні вимоги

- відтворюваність експериментів;
- модульність пайплайну;
- підтримка двох мов;
- збереження артефактів для подальшого аналізу.



Підготовка даних



1200 початкові промпти 600 EN / 600 UA	836 clean prompts 520 EN / 316 UA	3344 styled rows 713 train / 123 eval
---	--	--

Джерела промптів: CohereForAI/aya_dataset, saillab/alpaca-ukrainian-cleaned

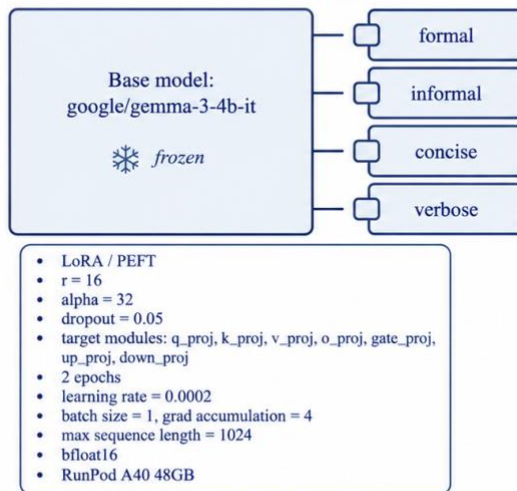
Очищення

- довжина та формат промпта;
- відповідність мови;
- відсів translation / coding / math задач;
- достатня гнучкість для зміни стилю.

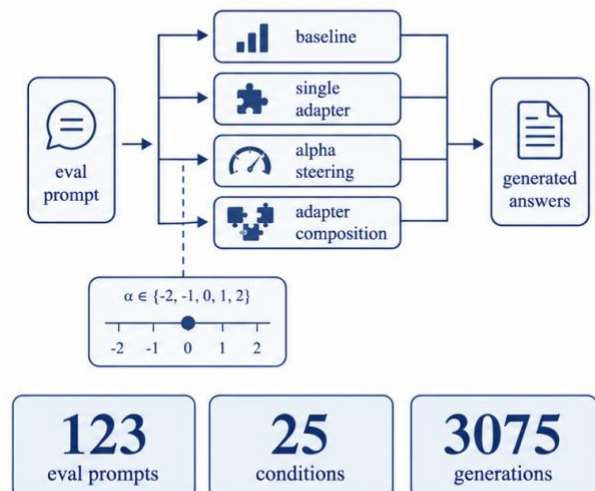


Реалізація системи

Навчання адаптерів



Керована генерація



Тестування та оцінювання

Unit-тестування

37 / 37 passed

pytest config, unittest-based tests

Покрито модулі:

- filtering
- conditions
- OpenAI batch / data parsing
- inference sampling / batching

Ключові результати

98.504%

language correctness

97.846% EN / 99.644% UA

0.8141

mean semantic similarity

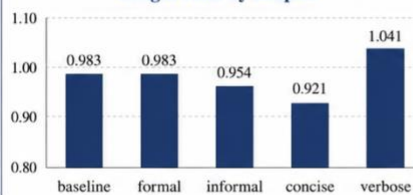
0%

invalid outputs

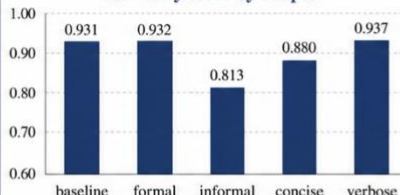
0%

truncation proxy

Length ratio by adapter



Formality score by adapter

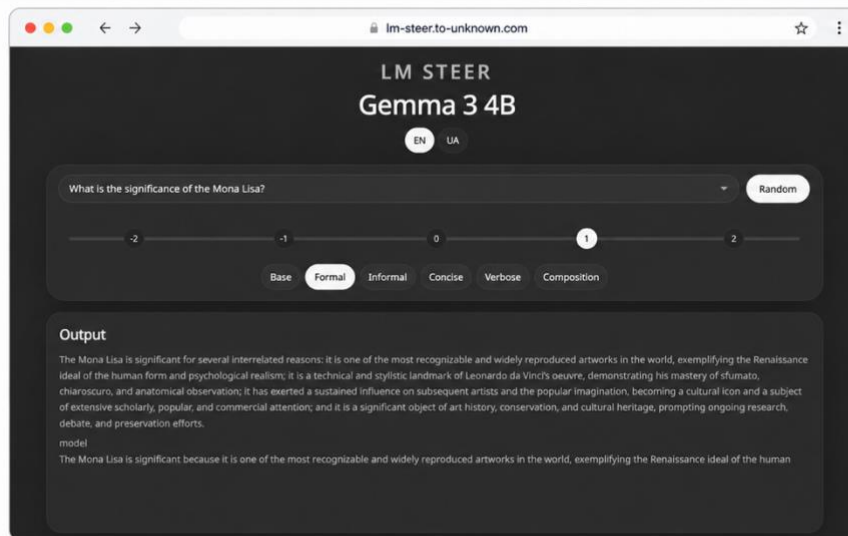


Спостереження:

- informal знижує formality score;
- verbose збільшує length ratio;
- сильна композиція 2_2 може погіршувати quality.



Впровадження та вебдемо



Технології:

Next.js, React, TypeScript

Дані інтерфейсу:

viewer-prompts.json,
viewer-validation.json,
viewer-meta.json

Можливості:

перегляд prompt-ib, conditions,
generated answers, scores,
plots, manual reviews

Deploy:

<https://lm-steer.to-unknown.com>



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

9

Висновки

- ✓ проаналізовано підходи до керування відповідями BMM;
- ✓ підготовлено двомовний набір даних EN/UA;
- ✓ сформовано clean subset: 836 prompts, 3344 styled rows;
- ✓ навчено 4 LoRA-адаптери: formal, informal, concise, verbose;
- ✓ реалізовано 25 умов керованої генерації;
- ✓ отримано 3075 generations для оцінювання;
- ✓ виконано автоматичне оцінювання;
- ✓ створено вебінтерфейс для демонстрації результатів.



Практична цінність

стиль відповіді можна змінювати через ваги
адаптерів без повного донавчання базової моделі.

836

clean prompts

3344

styled rows

4

LoRA adapters

25

conditions

3075

generations

98.504%

language correctness

Подальший розвиток

- розширення набору стилів;
- збільшення українського clean subset;
- більша ручна експертна оцінка;
- дослідження нових композицій адаптерів.

Дякую за увагу!



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

10