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

**ІЛИК ВІКТОР ВОЛОДИМИРОВИЧ**

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
__Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

# ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ПОВІДОМЛЕНЬ

Спеціальність 122 Комп’ютерні науки

**Кваліфікаційна (бакалаврська) робота**

Керівник:
Наталія ВЕСЕЛОВСЬКА, професор
кафедри інформаційних технологій,
д-р техн. наук

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця - 2026

## АНОТАЦІЯ

Ілик В. В. Використання штучного інтелекту для класифікації текстових повідомлень. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено проблеми автоматичного аналізу текстових повідомлень у багатомовному середовищі та розроблено десктопний застосунок ECHO. Система забезпечує комплексну класифікацію контенту Telegram-каналів, групових та особистих чатів: тематику, тональність, токсичність та агрегований профіль джерела з пояснювальним висновком. Продукт поєднує швидку локальну обробку ML-моделями з поглибленим аналізом через каскад великих мовних моделей, працює переважно офлайн, гарантує конфіденційність даних та адаптований до українського інформаційного простору з підтримкою кодового перемикання мов.

Ключові слова: класифікація тексту, штучний інтелект, обробка природної мови, токсичність, тональність, Telegram, гібридна система.

57 ст., 20 рис., 10 табл., 2 дод., 41 джерело.

## ABSTRACT

Ilyk V. V. Using artificial intelligence for classification of text messages. Specialty 122 «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification (bachelor’s) thesis examines automatic analysis of text messages in multilingual environments and develops the desktop application ECHO. The system provides comprehensive classification of Telegram channels, group and personal chats content: topics, sentiment, toxicity and aggregated source profile with explanatory conclusion. The product combines fast local ML-model processing with in-depth analysis via large language model cascade, operates primarily offline, ensures data privacy and is adapted to the Ukrainian information space with code-switching support.

Keywords: text classification, artificial intelligence, natural language processing, toxicity, sentiment, Telegram, hybrid system.

**ЗМІСТ**

# ВСТУП

Сучасне інформаційне суспільство характеризується стрімким зростанням обсягів текстових повідомлень у месенджерах, соціальних мережах та каналах зв'язку. Тексти стали основним способом обміну думками, новинами та емоціями, що створює нагальну потребу в ефективних інструментах їх автоматичного аналізу. Класифікація текстових повідомлень за тематикою, емоційним забарвленням та рівнем токсичності дозволяє модерувати контент, аналізувати громадську думку та захищати користувачів від негативного впливу. Розробка таких систем особливо актуальна для українського сегменту інтернету, де поширене кодове перемикання між українською, російською та англійською мовами, а також неформальна лексика.

Актуальність теми зумовлена експоненційним збільшенням обсягів цифрового контенту та необхідністю створення інструментів, які поєднують високу точність аналізу з дотриманням конфіденційності даних. Існуючі сервіси модерації часто працюють у хмарному режимі, що викликає занепокоєння щодо приватності, і недостатньо адаптовані до специфіки українського мовного середовища. Водночас локальні рішення рідко забезпечують комплексний аналіз, що включає тематичну класифікацію, оцінку тональності, токсичності та пояснювальний висновок. Тому залишається потреба у власній системі, яка працює переважно на пристрої користувача, ефективно обробляє тримовні тексти та надає зрозумілі рекомендації щодо інформаційного впливу джерел.

Мета роботи полягає у розробці десктопного програмного продукту ECHO для комплексної класифікації текстових повідомлень Telegram-каналів, групових та особистих чатів засобами штучного інтелекту.

Об'єктом дослідження є процеси аналізу та класифікації природномовних текстових повідомлень у багатомовному цифровому середовищі.

Предмет дослідження становлять методи та моделі штучного інтелекту, алгоритми обробки даних, архітектурні рішення та програмні засоби реалізації системи класифікації текстових повідомлень.

Визначено такі завдання дослідження:

– проаналізувати предметну область класифікації тексту, сучасні методи обробки природної мови та існуючі програмні аналоги;

– розробити концептуальну модель, алгоритми попередньої обробки, класифікації та агрегації результатів;

– сформувати модульну архітектуру програмної системи та підготувати її до реалізації;

– виконати програмну реалізацію функціональних модулів, забезпечити інтерфейс користувача та провести експериментальне тестування;

– проаналізувати результати роботи системи та визначити напрями її подальшого вдосконалення.

Практичне значення роботи полягає у створенні десктопного застосунку ECHO, який забезпечує локальну обробку даних, швидкий базовий вердикт за допомогою ML-моделей та поглиблений аналіз за допомогою каскаду великих мовних моделей. Система адаптована до українського інформаційного простору, підтримує різні джерела даних, візуалізує результати у зручному інтерфейсі та сприяє формуванню свідомого інформаційного раціону користувачів. Розроблений продукт може використовуватися журналістами, модераторами спільнот, аналітиками та звичайними користувачами для оцінки безпеки та якості текстових джерел.

Робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються теоретичні основи класифікації текстових повідомлень засобами штучного інтелекту, аналіз методів і моделей та огляд існуючих програмних рішень. Другий розділ присвячений розробці концептуальної моделі, алгоритмів обробки даних та структури програмної системи. Третій розділ містить опис архітектури та реалізації програмного продукту ECHO, результати тестування та аналіз ефективності. Робота включає рисунки, таблиці порівнянь та додатки з основними лістингами коду.

# РОЗДІЛ 1
# ТЕОРЕТИЧНІ ОСНОВИ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ПОВІДОМЛЕНЬ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

## 1.1 Дослідження предметної області та аналіз наукових підходів до класифікації текстових повідомлень

У сучасному інформаційному суспільстві текстові повідомлення в месенджерах, соціальних мережах та каналах зв'язку стали одним із основних способів обміну думками, новинами та емоціями. Обсяг такого контенту зростає експоненційно, що створює потребу в ефективних методах його автоматичного аналізу. Класифікація текстових повідомлень дозволяє визначати їхню тематику, емоційне забарвлення, рівень шкідливості чи відповідність певним категоріям, що має велике значення для модерації контенту, аналізу громадської думки та захисту користувачів від негативного впливу [1].

Предметна область класифікації тексту охоплює лінгвістику, штучний інтелект та обробку природної мови. Наукові підходи до цієї задачі еволюціонували від простих правил, заснованих на ключових словах, до складних статистичних моделей і нейронних мереж. Традиційні методи, такі як наївний байєсівський класифікатор чи метод опорних векторів, демонстрували добрі результати на відносно невеликих наборах даних з чітко визначеними категоріями. Однак вони погано справлялися з багатозначністю слів, контекстом та неформальною мовою повідомлень у чатах [2].

Перехід до глибокого навчання радикально змінив ситуацію. Моделі на основі трансформерів, зокрема архітектури BERT та її похідні, дозволяють враховувати контекст усього речення чи навіть цілого тексту завдяки механізму уваги. Це особливо важливо для повідомлень, які часто містять скорочення, емодзі, кодове перемикання між мовами та емоційно забарвлену лексику [3]. У багатосторонньому середовищі, де використовуються українська, російська та англійська мови одночасно, мультимовні моделі стають незамінними, оскільки

вони навчені на великих корпусах різних мов і можуть ефективно працювати з мовними міксами. Поріняння підходів до класифікації тексту наведене в таблиці 1.1.

Таблиця 1.1 – Порівняння основних підходів до класифікації тексту

| **Підхід** | **Переваги** | **Недоліки** | **Застосування** |
|---|---|---|---|
| Правила та ключові слова | Простота реалізації, швидкість | Низька точність на нестандартному тексті | Проста фільтрація спаму |
| Статистичні моделі (Naive Bayes, SVM) | Добрі результати на невеликих даних | Слабко враховують контекст | Тематична класифікація |
| Нейронні мережі (LSTM, GRU) | Врахування послідовності | Потребують багато даних | Аналіз емоцій |
| Трансформери (BERT-подібні) | Контекстне розуміння, мультимовність | Високі вимоги до обчислювальних ресурсів | Комплексний аналіз повідомлень |

Розвиток штучного інтелекту в цій сфері тісно пов’язаний із появою великих мовних моделей. Вони не лише класифікують текст, але й генерують пояснення своїх рішень, що підвищує довіру до результатів.

У контексті повідомлень особливу увагу приділяють визначенню токсичності – здатності тексту ображати, розпалювати ворожнечу чи спричиняти емоційне виснаження. Для цього використовують спеціалізовані моделі, навчені на анотованих наборах даних, де оцінюється ступінь шкідливості від нейтрального до вкрай негативного [4]. Схема загального процесу класифікації текстових повідомлень наведена рисунку 1.1.

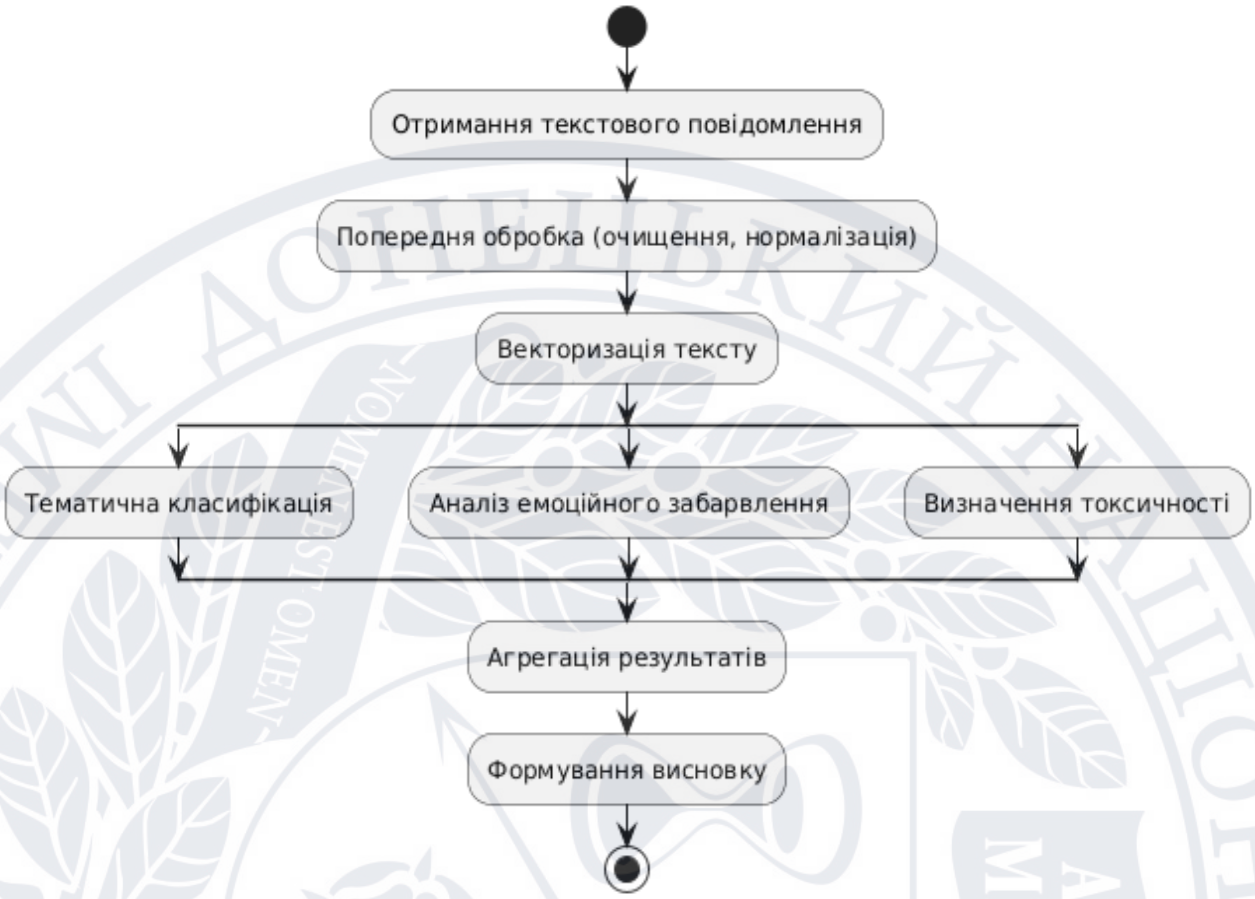


Рисунок 1.1 – Загальна схема процесу класифікації текстових повідомлень засобами штучного інтелекту

Ця схема ілюструє послідовні етапи, починаючи від отримання сирого тексту до формування комплексного висновку. Кожен етап може виконуватися окремими моделями, що працюють паралельно для підвищення швидкості.

Наукові дослідження останніх років акцентують увагу на етичних аспектах класифікації. Автоматичні системи повинні уникати упередженості щодо певних мов, діалектів чи соціальних груп.

Особливо це стосується української мови, яка має меншу представленість у глобальних тренувальних корпусах порівняно з англійською. Тому розробники створюють спеціалізовані українські та мультимовні моделі, що враховують особливості словотвору, граматики та культурного контексту [5]. Характеристики сучасних моделей для аналізу тексту показані в таблиці 1.2.

Таблиця 1.2 – Характеристики сучасних моделей для аналізу тексту

| **Модель** | **Кількість параметрів** | **Підтримувані мови** | **Основне призначення** | **Точність (приблизно)** |
|---|---|---|---|---|
| mDeBERTa-v3-base | 86 млн | Багатомовна (вкл. українську) | Тематична класифікація | 85–92% |
| XLM-RoBERTa (toxic) | 270 млн | 100+ мов | Виявлення токсичності | 88–94% |
| BERT multilingual uncased | 110 млн | Багатомовна | Аналіз емоцій | 82–90% |
| Великі мовні моделі (Llama, Gemini) | 7–235 млрд | Багатомовні | Комплексний аналіз з поясненнями | 90%+ |

Одним із перспективних напрямків є поєднання статистичних і нейромережевих методів. Гібридні системи спочатку застосовують швидкі евристичні правила для відсіювання очевидного спаму чи нейтрального контенту, а потім глибокі моделі працюють лише з найбільш складними випадками.

Такий підхід економить обчислювальні ресурси та підвищує загальну ефективність [6]. Крім того, важливим є врахування динаміки: аналіз не окремого повідомлення, а послідовності текстів дозволяє виявляти патерни поведінки, емоційні тренди чи спроби маніпуляції. На рисунку 1.2 наведене схематичне зображення архітектруи гібридної системи класифікації текстових повідомлень.


Рисунок 1.2 – Архітектура гібридної системи класифікації текстових повідомлень

На схемі видно, як прості евристики доповнюють потужні нейронні мережі, забезпечуючи баланс між швидкістю та точністю.

Сучасні дослідження також підкреслюють важливість інтерпретовності результатів. Користувач повинен розуміти, чому повідомлення віднесено до певної категорії. Для цього застосовують методи візуалізації уваги моделі, виділення ключових слів та фраз, що вплинули на рішення. Це особливо актуально в чутливих сферах, таких як модерація контенту чи аналіз суспільних настроїв [7].

Розвиток технологій штучного інтелекту в класифікації тексту продовжує активно тривати. Нові моделі стають ефективнішими, менш ресурсозатратними та чутливішими до культурних і мовних особливостей [8]. У перспективі очікується подальша інтеграція таких систем у повсякденні інструменти комунікації, що дозволить створювати безпечніше інформаційне середовище для всіх користувачів [9].

## 1.2 Аналіз методів і моделей штучного інтелекту для обробки та класифікації текстів

Обробка текстових повідомлень засобами штучного інтелекту становить одну з ключових галузей сучасної інформатики, оскільки величезні обсяги інформації, що циркулюють у цифровому просторі, вимагають ефективних інструментів для автоматичного розуміння змісту, емоційного забарвлення та тематичної спрямованості.

Традиційні підходи до класифікації текстів спиралися переважно на статистичні методи, такі як мішок слів або зважена частотність термінів, які дозволяли перетворювати природну мову на числові вектори для подальшого застосування алгоритмів машинного навчання. Ці методи, хоч і прості у реалізації, мали суттєві обмеження, зокрема слабку здатність враховувати контекст, семантичні зв’язки між словами та багатозначність мовних конструкцій. У зв’язку з цим науковці почали активно розвивати більш складні моделі, здатні вловлювати нюанси людської мови [10].

Перехід до глибоких нейронних мереж значно розширив можливості аналізу текстів. Зокрема, згорткові нейронні мережі та рекурентні архітектури, такі як довготривала короткочасна пам’ять, дозволили моделям краще працювати з послідовностями символів і слів, зберігаючи інформацію про попередній контекст.

Однак справжній прорив стався з появою архітектури трансформерів, яка базується на механізмі уваги. Ця технологія дає змогу моделі одночасно аналізувати всі частини речення, встановлюючи залежності між віддаленими елементами тексту без втрати інформації під час обробки довгих послідовностей.

Завдяки цьому трансформери стали основою для створення потужних мовних моделей, що демонструють високу точність у завданнях класифікації тональності, виявлення токсичності та розподілу за темами [11].

Рисунок 1.3 ілюструє спрощену архітектуру процесу класифікації текстових даних, починаючи від попередньої підготовки до отримання остаточного результату.

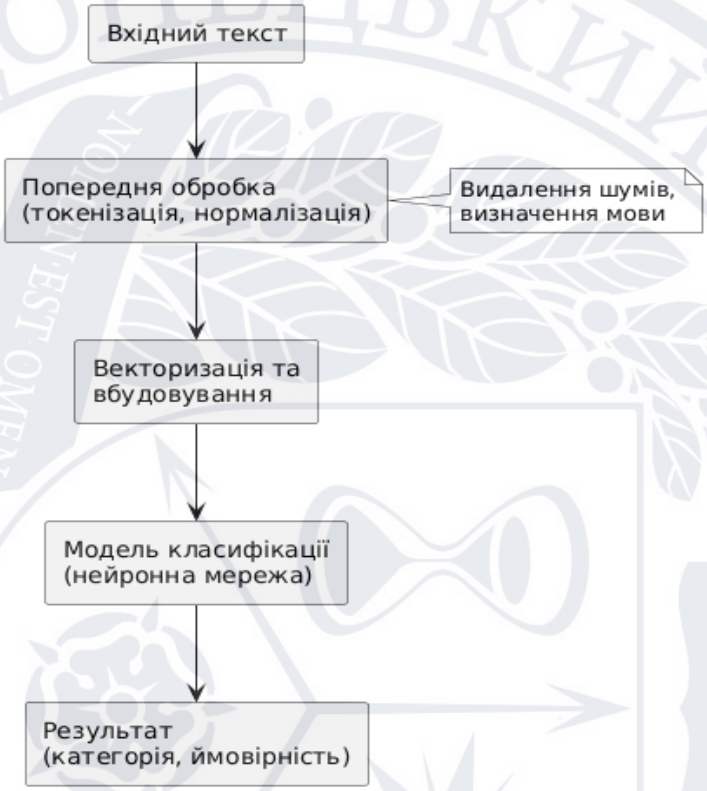


Рисунок 1.3 – Спрощена схема процесу класифікації текстових повідомлень засобами штучного інтелекту

Як видно з рисунку, етап попередньої обробки відіграє вирішальну роль, оскільки якість очищеного тексту безпосередньо впливає на точність подальших розрахунків. Сучасні системи все частіше поєднують кілька типів аналізу: статистичний, семантичний і контекстний, що дозволяє досягати більш надійних результатів навіть для текстів з неформальною лексикою чи кодовим перемиканням між мовами [12].

Важливим напрямком розвитку є створення багатомовних моделей, здатних працювати з текстами різними мовами без потреби в окремому навчанні для кожної.

Такі моделі навчаються на величезних корпусах даних, що охоплюють десятки мов, і демонструють добрі результати в аналізі тональності та токсичності для низькоресурсних мов, зокрема української. Вони враховують культурні особливості вираження емоцій і дозволяють проводити порівняльний аналіз контенту з різних мовних середовищ [13].

Для наочного порівняння основних підходів до класифікації текстів доцільно звернутися до узагальнених характеристик методів, наведених у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика методів класифікації текстів

| **Метод** | **Переваги** | **Недоліки** | **Сфера застосування** |
|---|---|---|---|
| Статистичні (TF-IDF) | Простота, швидкість | Не враховує контекст | Базова фільтрація спаму |
| Рекурентні мережі | Добре працює з послідовностями | Проблеми з довгими текстами | Аналіз коротких повідомлень |
| Трансформери | Відмінне розуміння контексту | Високі вимоги до обчислювальних ресурсів | Багатозадачна класифікація |

Таблиця демонструє, як еволюція методів від простих статистичних до складних нейромережевих архітектур дозволяє поступово долати обмеження попередніх підходів. Особливо цінними є трансформери, оскільки вони забезпечують високу точність у багатозадачних сценаріях, включаючи одночасне визначення тематики, емоційного забарвлення та рівня токсичності [14].

Рисунок 1.4 відображає спрощену архітектуру моделі на основі трансформерів з механізмом багатоголової уваги.


Рисунок 1.4 – Спрощена архітектура трансформерної моделі для класифікації текстів

Ця архітектура забезпечує ефективне захоплення довгострокових залежностей, що особливо важливо для аналізу складних повідомлень з іронією чи непрямими формулюваннями. Сучасні дослідження підкреслюють, що поєднання таких моделей з методами постійного навчання дозволяє адаптувати системи до нових мовних явищ без повного перенавчання [15].

Окремий інтерес становить аналіз стилістичних особливостей тексту, включаючи частоту використання розділових знаків, довжину речень та наявність посилань. Такі характеристики доповнюють основні моделі класифікації, роблячи висновки більш обґрунтованими. Універсальність трансформерних підходів підтверджується їх успішним застосуванням у різних галузях – від моніторингу суспільних настроїв до забезпечення безпечного інформаційного середовища [16].

Таким чином, розвиток методів і моделей штучного інтелекту для обробки текстової інформації демонструє стійку тенденцію до інтеграції статистичних і нейромережевих підходів. Це дозволяє створювати системи, які не лише точно класифікують повідомлення, але й пояснюють причини своїх висновків, сприяючи прозорості та довірі до автоматизованих рішень. Подальші дослідження зосереджуються на підвищенні енергоефективності моделей та їх адаптації до реальних умов багатоязычного середовища, що є особливо актуальним для країн із розвиненим цифровим простором [17].

## 1.3 Аналіз аналогів програмних рішень та формування вимог до програмного забезпечення

Сучасні програмні рішення для класифікації текстових повідомлень на основі штучного інтелекту відіграють важливу роль у забезпеченні безпечного інформаційного середовища, особливо в умовах швидкого поширення цифрових комунікацій [18]. Такі системи дозволяють автоматично виявляти токсичний вміст, визначати емоційне забарвлення тексту та класифікувати тематику повідомлень, що допомагає модераторам і користувачам ефективніше взаємодіяти з великими обсягами даних.

Аналіз наявних аналогів демонструє, що більшість інструментів орієнтована на англомовний простір, проте поступово розвивається підтримка багатослівних мов, зокрема української [19].

Одним із провідних напрямів є хмарні сервіси для модерації контенту. Вони пропонують швидку обробку в реальному часі, але часто вимагають передачі даних на зовнішні сервери, що викликає занепокоєння щодо конфіденційності. Інший підхід представлений відкритими бібліотеками та моделями, які можна розгортати локально, забезпечуючи повний контроль над інформацією [20]. Такі рішення зазвичай поєднують кілька завдань: визначення токсичності, аналіз тональності та тематичну класифікацію.

Для наочності на рисунку 1.5 розглянуто загальну структуру типового програмного рішення для класифікації текстових повідомлень.

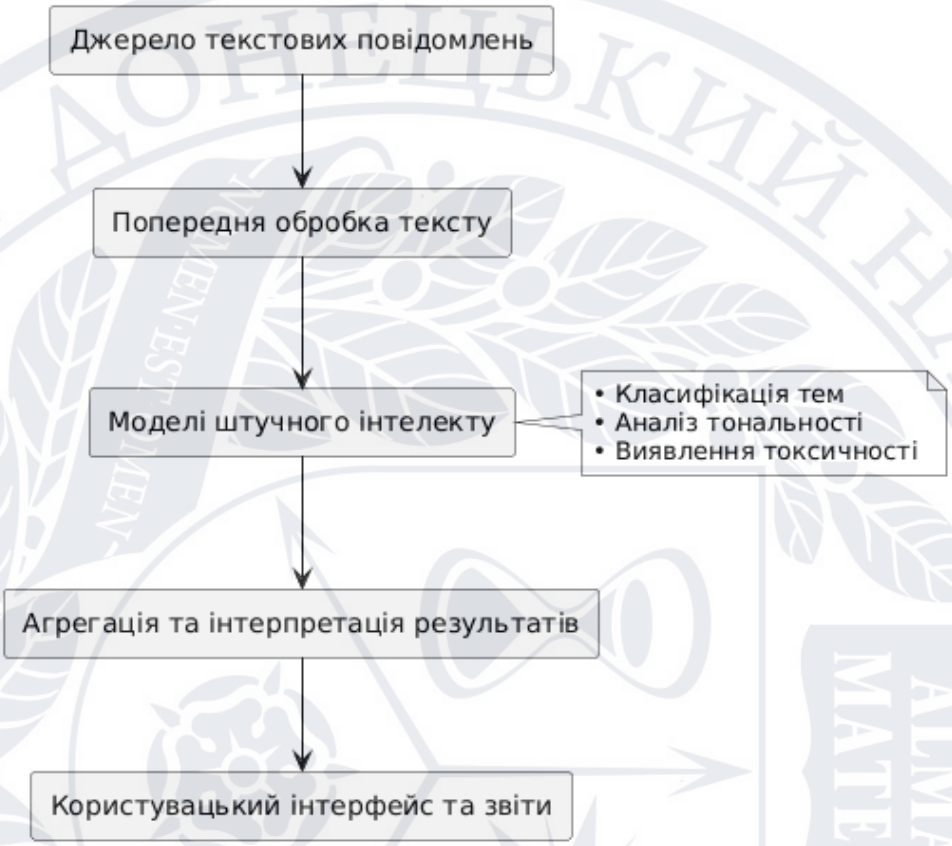


Рисунок 1.5 – Загальна архітектура програмного рішення для класифікації текстових повідомлень засобами штучного інтелекту

Як видно з рисунку 1.5, типова архітектура включає етапи попередньої обробки, застосування моделей штучного інтелекту та формування зрозумілих для користувача результатів. Така структура забезпечує послідовність обробки та можливість інтеграції різних компонентів.

Порівняльний аналіз аналогів (таблиця 1.4) дозволяє виділити ключові характеристики існуючих рішень [21]. Багато систем ефективно працюють з англійською та основними європейськими мовами, але підтримка української часто обмежена або вимагає додаткового донавчання моделей.

Таблиця 1.4 – Порівняльна характеристика програмних рішень для класифікації текстових повідомлень

| Назва рішення | Підтримка мов | Режим обробки | Конфіденційність даних | Основні функції | Переваги |
|---|---|---|---|---|---|
| Perspective API (Google) | Багатомовна (обмежена українська) | Хмарний, реальний час | Низька (передача даних) | Токсичність, агресія | Швидкість, масштабованість |
| Hugging Face Transformers | Багатомовна (включаючи українську) | Локальний/ хмарний | Висока (локально) | Тональність, теми, токсичність | Відкритість, гнучкість |
| Detoxify | Багатомовна | Локальний | Висока | Виявлення токсичності | Простота інтеграції |
| Комерційні модератори (наприклад, OpenAI Moderation) | Багатомовна | Хмарний | Середня | Токсичність, насильство | Інтеграція з API |

Таблиця 1.4 ілюструє, що локальні рішення вирізняються вищим рівнем захисту персональних даних, тоді як хмарні сервіси забезпечують вищу швидкість обробки великих обсягів інформації. Недоліком багатьох аналогів залишається недостатня адаптація до специфіки українського мовного середовища, зокрема кодового перемикання між українською, російською та англійською мовами, що характерно для вітчизняних комунікацій [22].

Ще одним важливим аспектом є пояснюваність результатів. Сучасні системи дедалі частіше включають механізми, які не лише видають вердикт, але й пояснюють, на основі яких ознак прийнято рішення [23]. Це підвищує довіру користувачів і дозволяє фахівцям верифікувати висновки.

Для кращого розуміння взаємозв’язків між основними функціональними вимогами до подібних систем можна представити їх у вигляді діаграми, що

наведена на рисунку 1.6.


Рисунок 1.6 – Взаємозв’язки ключових функціональних вимог до програмного забезпечення для класифікації тексту

Рисунок 1.6 підкреслює, що підтримка кількох мов тісно пов’язана із захистом даних, а пояснюваність результатів безпосередньо впливає на якість інтеграції з користувацькими інтерфейсами [24].

На основі проведеного аналізу аналогів формуються основні вимоги до нового програмного забезпечення. Воно повинно забезпечувати ефективну роботу з українською мовою, включаючи обробку змішаних текстів. Важливим є поєднання локальної обробки для конфіденційних даних із можливістю використання хмарних ресурсів для складних завдань [25]. Система має надавати не лише класифікацію, але й зрозумілі пояснення, що робить її доступною для широкого кола користувачів, зокрема журналістів, модераторів спільнот і звичайних людей, які прагнуть свідомо споживати інформацію.

Крім того, програмне забезпечення має бути гнучким щодо джерел даних, підтримувати різні формати повідомлень і дозволяти налаштування порогових значень для класифікації залежно від контексту. Особливу увагу слід приділити

енергоефективності та можливості роботи на звичайних комп'ютерах без потужних відеокарт, що розширює доступність інструменту [26]. Інтерфейс користувача повинен бути інтуїтивно зрозумілим, з візуалізацією результатів у вигляді кольорових індикаторів і детальних звітів.

Таким чином, аналіз існуючих програмних рішень свідчить про необхідність створення збалансованої системи, яка поєднує високу точність класифікації, повагу до приватності користувачів та адаптацію до реалій українського інформаційного простору. Таке програмне забезпечення матиме значний потенціал для підвищення якості цифрової комунікації та сприятиме формуванню здорового інформаційного середовища.

Таким чином, сучасні підходи до класифікації текстових повідомлень демонструють еволюцію від простих правил і статистичних методів до потужних трансформерних архітектур, здатних ефективно враховувати контекст, мовні мішки та культурні особливості. Мультимовні моделі на кшталт mDeBERTa, XLM-RoBERTa та великих мовних моделей забезпечують високу точність у визначенні тематики, емоційного забарвлення та рівня токсичності навіть у середовищах із кодовим перемиканням між українською, російською та англійською мовами.

Гібридні системи, що поєднують евристичні фільтри з нейромережевими компонентами, дозволяють оптимізувати швидкість і ресурсозатратність обробки, тоді як акцент на інтерпретовності результатів підвищує довіру до автоматизованих рішень. Аналіз наявних програмних аналогів засвідчує переваги локальних рішень щодо конфіденційності даних, проте виявляє недоліки в адаптації до специфіки українського мовного простору.

Перспективним напрямом залишається створення гнучких систем, орієнтованих на реалії багатоязычного цифрового середовища, з балансом між точністю, енергоефективністю та доступністю для широкого кола користувачів. Такі розробки сприятимуть модерації контенту, захисту від негативного впливу та формуванню безпечнішого інформаційного простору.

# РОЗДІЛ 2
# РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ПОВІДОМЛЕНЬ

## 2.1 Розроблення концептуальної моделі класифікації текстових повідомлень

Розроблення концептуальної моделі класифікації текстових повідомлень становить важливий етап у створенні ефективних систем аналізу природної мови, особливо в умовах поширення цифрових комунікацій [27]. Така модель має враховувати багатогранність текстового контенту, що включає емоційне забарвлення, тематичну спрямованість та потенційний вплив на сприйняття читача.

Основою слугає принцип багаторівневого опрацювання, де вхідні дані проходять послідовні етапи від первинної підготовки до інтегрованого оцінювання. Це дозволяє поєднати статистичні підходи з елементами глибокого розуміння контексту, забезпечуючи баланс між точністю та адаптивністю до різних мовних середовищ, зокрема україномовних текстів [28].

Концептуальна модель передбачає чітке структурування процесу, починаючи з надходження сирих повідомлень. На першому етапі відбувається очищення та нормалізація тексту, що включає видалення надлишкових пробілів, розпізнавання мови та фільтрацію несуттєвих елементів, таких як чисті символи чи надто короткі фрагменти. Цей крок є критичним для підвищення якості подальшого аналізу, оскільки неякісні дані можуть суттєво спотворити результати.

Далі модель передбачає паралельне застосування спеціалізованих класифікаторів, які оцінюють різні аспекти тексту незалежно, але з можливістю взаємного збагачення. Такий паралельний підхід сприяє всебічному профілюванню, де кожен компонент доповнює загальну картину [29].

Одним із ключових елементів є визначення тематичної спрямованості

повідомлень. Для цього використовуються методи нульової класифікації, що дозволяють віднести текст до попередньо визначених категорій без необхідності великого обсягу розмічених даних для кожного нового домену. Це особливо корисно для динамічних середовищ, де теми швидко еволюціонують.

Паралельно оцінюється тональність – від нейтральної до виражено емоційної – та рівень потенційно шкідливого змісту, що допомагає кількісно вимірювати емоційне навантаження. Агрегація цих показників формує зведений профіль, який відображає не лише окремі характеристики, а й їх взаємозв'язки, наприклад, концентрацію певних тем у поєднанні з емоційним забарвленням [30].

Для наочності на рисунку 2.1 представлено загальну архітектуру концептуальної моделі.

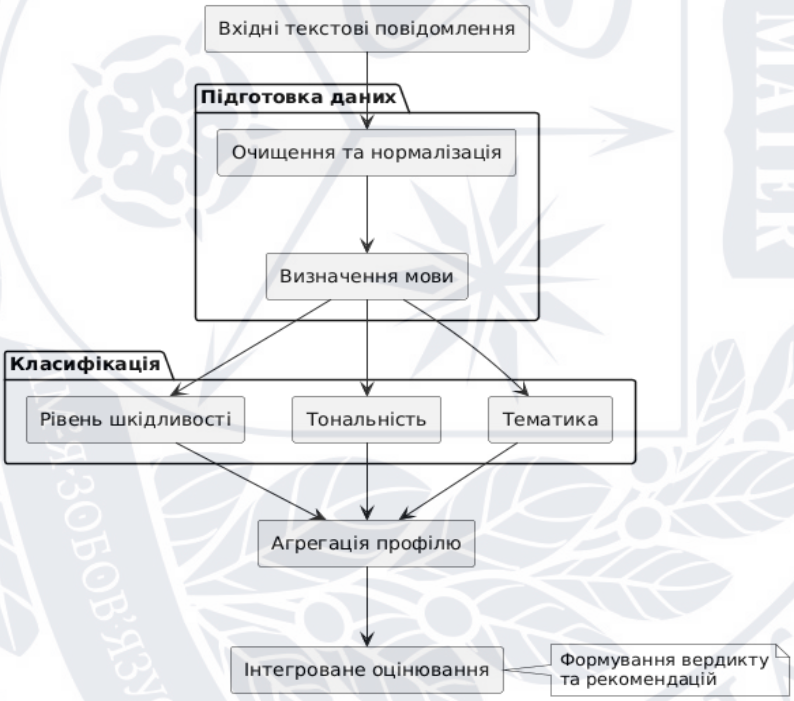


Рисунок 2.1 – Концептуальна архітектура моделі класифікації текстових повідомлень

На рисунку 2.1 зображено основні блоки моделі, що підкреслює послідовність і паралельність процесів, забезпечуючи ефективність опрацювання великих обсягів даних.

Важливим аспектом моделі є перехід від локальних статистичних оцінок до більш глибокого контекстного аналізу. Після формування зведеного профілю можливе залучення додаткових механізмів для виявлення складніших патернів, таких як стилістичні особливості чи приховані смислові зв’язки. Це дозволяє не лише класифікувати, а й пояснювати отримані результати, підвищуючи довіру до системи. Концепція передбачає також можливість адаптації вагових коефіцієнтів для різних типів комунікації, наприклад, для публічних джерел чи приватних обговорень, де пріоритети оцінки можуть відрізнятися [31].

Для порівняння основних підходів до класифікації доцільно розглянути їхні характеристики, наведені в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика методів класифікації текстових повідомлень

| **Метод** | **Переваги** | **Обмеження** | **Застосування** |
|---|---|---|---|
| Статистичний | Простота реалізації, швидкість | Низька чутливість до контексту | Попередня фільтрація |
| Машинне навчання | Добре узагальнення на даних | Потреба в розмічених вибірках | Тональність та тематика |
| Глибокі нейронні мережі | Висока точність на складних даних | Великі обчислювальні ресурси | Виявлення нюансів |
| Комбінований (багаторівневий) | Комплексність аналізу | Складність інтеграції | Повноцінне профілювання |

Таблиця 2.1 ілюструє, чому комбінований підхід є найбільш перспективним для сучасних завдань, оскільки він поєднує сильні сторони окремих методів.

Подальшим розвитком моделі є інтеграція механізмів для врахування динаміки текстів у часі та мовного різноманіття. Це особливо актуально для середовищ, де співіснують різні мови чи стилі викладу. Модель передбачає створення представницьких вибірок, які відображають типові приклади для

подальшого осмислення. Такий підхід сприяє не лише класифікації, а й формуванню рекомендацій щодо сприйняття контенту, допомагаючи користувачам свідомо ставитися до інформаційного потоку [32].

Для детальнішого розуміння потоку даних у моделі на рисунку 2.2 представлено схему основних етапів обробки.

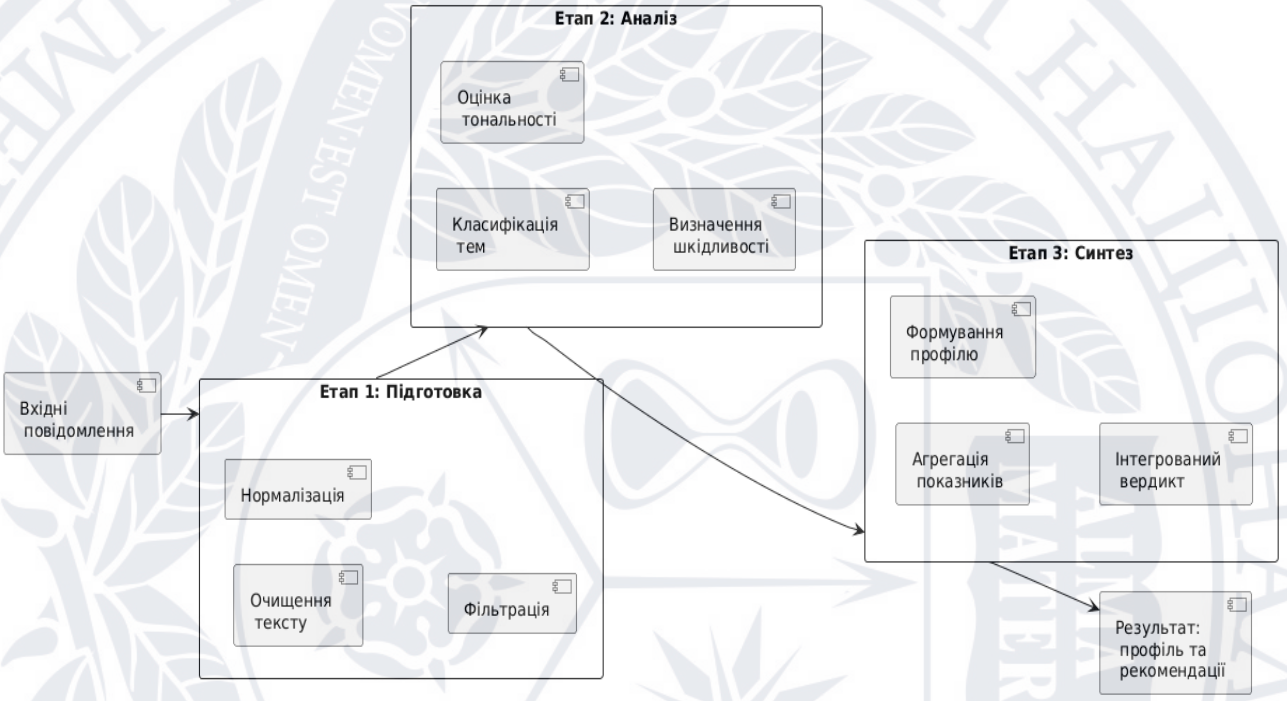


Рисунок 2.2 – Потік даних у концептуальній моделі класифікації

Рисунок 2.2 демонструє послідовність етапів, що забезпечує системність і прозорість усього процесу. Завершальним елементом моделі є механізм зворотного зв’язку, який дозволяє вдосконалювати параметри на основі накопиченого досвіду, роблячи систему більш адаптивною до реальних умов застосування.

Таким чином, концептуальна модель не є статичною конструкцією, а динамічною рамкою, спрямованою на гуманістичне використання технологій для підтримки інформованого сприйняття інформації в сучасному світі.

Цей підхід сприяє не лише технічній ефективності, а й етичному виміру аналізу, підкреслюючи важливість балансу між автоматизацією та збереженням людського судження.

## 2.2 Побудова алгоритмів обробки текстових даних та класифікації

Обробка текстових повідомлень у сучасних інформаційних системах вимагає послідовного застосування методів, що забезпечують очищення, нормалізацію та глибокий семантичний аналіз даних [33]. На початковому етапі основну увагу приділяють видаленню надлишкових елементів, таких як зайві пробіли, символи розмітки чи емодзі, які не несуть змістовного навантаження. Це дозволяє зосередитись на суттєвій частині тексту, зменшуючи шум і підвищуючи точність подальших обчислень.

Важливим кроком є визначення мови повідомлення, оскільки багатомовне середовище, характерне для українського сегмента інтернету, потребує адаптивних підходів до обробки української, російської та англійської мов одночасно. Такі методи дають змогу ефективно фільтрувати короткі або позбавлені реального змісту записи, зберігаючи лише ті, що містять достатню кількість осмислених слів [34].

Після попередньої підготовки даних розпочинається етап класифікації, який ґрунтується на поєднанні статистичних і нейромережевих підходів. Класифікація за темами здійснюється за допомогою моделей нульового навчання, що не вимагають великої кількості розмічених прикладів для конкретної задачі. Це особливо корисно для динамічних тематичних полів, де розподіл повідомлень може охоплювати такі категорії, як суспільно-політичні події, економіка, повсякденне життя чи технології.

Паралельно проводиться оцінка тональності тексту, яка дозволяє визначити емоційний відтінок – від виражено негативного до позитивного – на основі попередньо навчених багатоязикових моделей. Окремий напрямок становить виявлення токсичності, що вимірюється як ступінь шкідливості змісту в діапазоні від нуля до одиниці. Такі оцінки виконуються незалежно для кожного повідомлення, а потім агрегуються в загальний профіль джерела [35].

Для узагальнення результатів класифікації застосовують алгоритми агрегації, які формують комплексний портрет текстового масиву. Він включає

розподіл тем, показники емоційного навантаження, частку токсичних елементів та стилістичні особливості, такі як використання великих літер, знаків оклику чи посилань. На основі цих даних обчислюється інтегральний показник емоційного навантаження за зваженою формулою, що враховує внесок токсичності, агресивності, тематичної концентрації та негативного зміщення.

Отримане значення порівнюють з пороговими рівнями для визначення загального рівня ризику. Такий підхід забезпечує об’єктивність і відтворюваність оцінок, мінімізуючи вплив суб’єктивних факторів [36].

Рисунок 2.3 ілюструє загальну послідовність етапів обробки текстових даних від початкового введення до формування комплексного профілю.

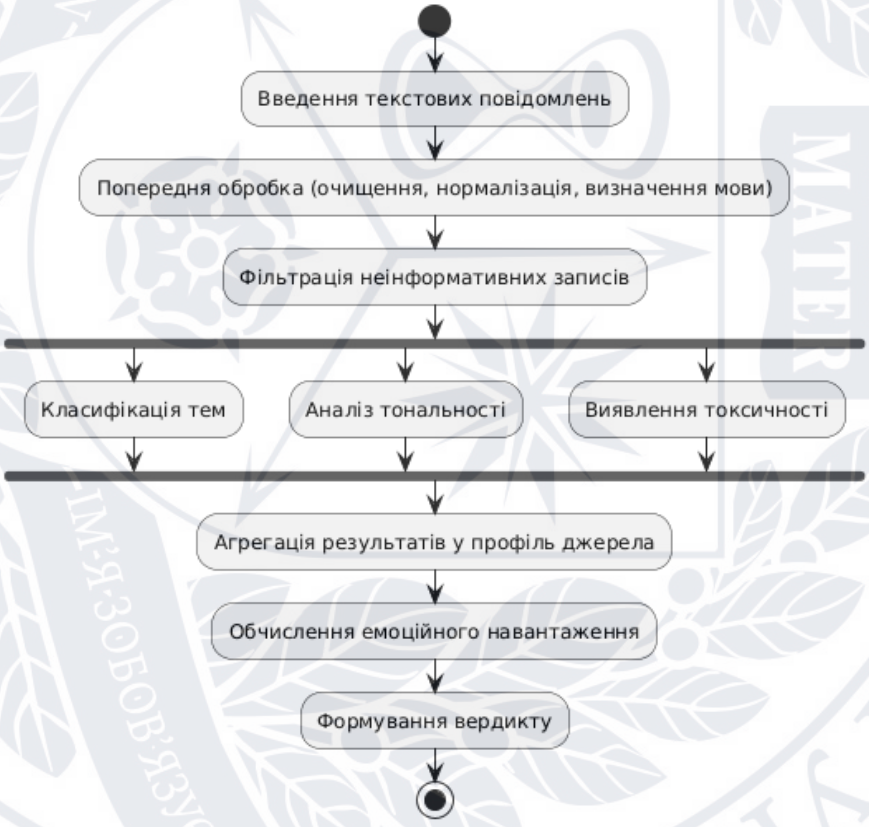


Рисунок 2.3 – Схема алгоритму обробки текстових даних та класифікації

Ця схема демонструє паралельне виконання класифікаційних задач, що суттєво прискорює обробку великих обсягів даних.

Для практичної реалізації важливо враховувати параметри, що впливають на якість результатів. В таблиці 2.2 наведено ключові характеристики основних етапів класифікації.

Таблиця 2.2 – Основні параметри алгоритмів обробки та класифікації текстових даних

| **Етап обробки** | **Основні операції** | **Ключові параметри** | **Очікуваний результат** |
|---|---|---|---|
| Попередня підготовка | Очищення, нормалізація, фільтрація | Мінімальна кількість слів, мова | Чисті інформативні повідомлення |
| Класифікація тем | Нульове навчання | Кількість категорій (7–8) | Розподіл за тематичними групами |
| Аналіз тональності | Багатомовна модель | Рівні від негативного до позитивного | Середній емоційний бал |
| Виявлення токсичності | Спеціалізована модель | Шкала 0.0–1.0 | Частка токсичних елементів |

Ці параметри забезпечують гнучкість системи та можливість адаптації до різних типів джерел.

Для підвищення точності оцінок у складних випадках застосовують додатковий рівень аналізу за допомогою великих мовних моделей. Вони отримують агрегований профіль і репрезентативні приклади, після чого генерують детальне пояснення з урахуванням контексту, іронії та стилістичних особливостей. Такий каскадний підхід поєднує швидкість локальних обчислень зі глибиною семантичного розуміння. Кешування результатів попередніх аналізів дозволяє уникати повторних обчислень для схожих даних [37].

Рисунок 2.4 відображає структуру обчислення інтегрального показника емоційного навантаження.


Рисунок 2.4 – Структура розрахунку емоційного навантаження для класифікації

Цей механізм забезпечує прозорість і можливість налаштування вагових коефіцієнтів залежно від конкретних завдань.

У цілому побудова алгоритмів обробки та класифікації текстових даних базується на гармонійному поєднанні етапів попередньої підготовки, паралельної класифікації та агрегації результатів. Такий підхід дозволяє ефективно аналізувати великі обсяги повідомлень у реальному часі, враховуючи мовну специфіку українського інформаційного простору та забезпечуючи високу надійність отриманих оцінок. Подальший розвиток цих методів пов’язаний з удосконаленням багатоязикових моделей і розширенням можливостей пояснювального аналізу.

## 2.3 Формування структури програмної системи та підготовка до програмної реалізації

Формування архітектури програмної системи для класифікації текстових повідомлень є ключовим етапом, що забезпечує ефективність, масштабованість та надійність майбутнього рішення. Цей процес передбачає чітке визначення основних модулів, їх взаємодії та принципів організації даних, аби система могла

послідовно обробляти природну мову, застосовувати математичні моделі та видавати обґрунтовані результати.

У сучасних умовах, коли обсяги текстової інформації зростають експоненційно, особливо в мережах обміну повідомленнями, структура повинна поєднувати гнучкість з високою продуктивністю, дозволяючи інтегрувати локальні обчислення з хмарними ресурсами без втрати конфіденційності [38].

Основою архітектури стає модульний підхід, за яким система поділяється на відносно незалежні компоненти, що взаємодіють через чітко визначені інтерфейси. Такий дизайн полегшує подальше тестування, оновлення окремих частин та адаптацію до нових вимог. Центральне місце посідає блок джерел даних, який забезпечує уніфікований доступ до повідомлень незалежно від їх походження – чи то безпосереднє завантаження з мереж, чи обробка попередньо збережених файлів. Далі йде етап попередньої підготовки тексту, де здійснюється очищення, нормалізація та визначення мовних особливостей, що є критичним для багатомовного середовища, характерного для українського сегменту інтернету [39].

Після підготовки даних активізується конвеєр локальної обробки, що включає кілька паралельних класифікаторів. Кожен з них спеціалізується на певному аспекті тексту: розподілі за тематичними категоріями, оцінці емоційного забарвлення та визначенні ступеня шкідливості висловлювань.

Паралельне виконання цих завдань на рівні пакетів повідомлень суттєво прискорює роботу, особливо при використанні графічних процесорів. Отримані характеристики агрегуються в єдиний профіль джерела, який містить статистичні показники, розподіли та репрезентативні приклади. Такий профіль стає основою для подальшого аналізу та формування початкового висновку за допомогою математичної формули, що враховує зважені коефіцієнти основних метрик.

Для підвищення точності та глибини інтерпретації передбачено інтеграцію з потужними мовними моделями, організованими у каскадну схему. Якщо один сервіс недоступний, система автоматично переходить до наступного, забезпечуючи стабільність роботи. Результати проміжних етапів зберігаються в

локальній базі даних, що дозволяє повторно використовувати обчислення та уникати надмірного навантаження на зовнішні ресурси.

На рисунку 2.5 наведене схематичне зображення загальної архітектури програмної системи для класифікації текстових повідомлень.

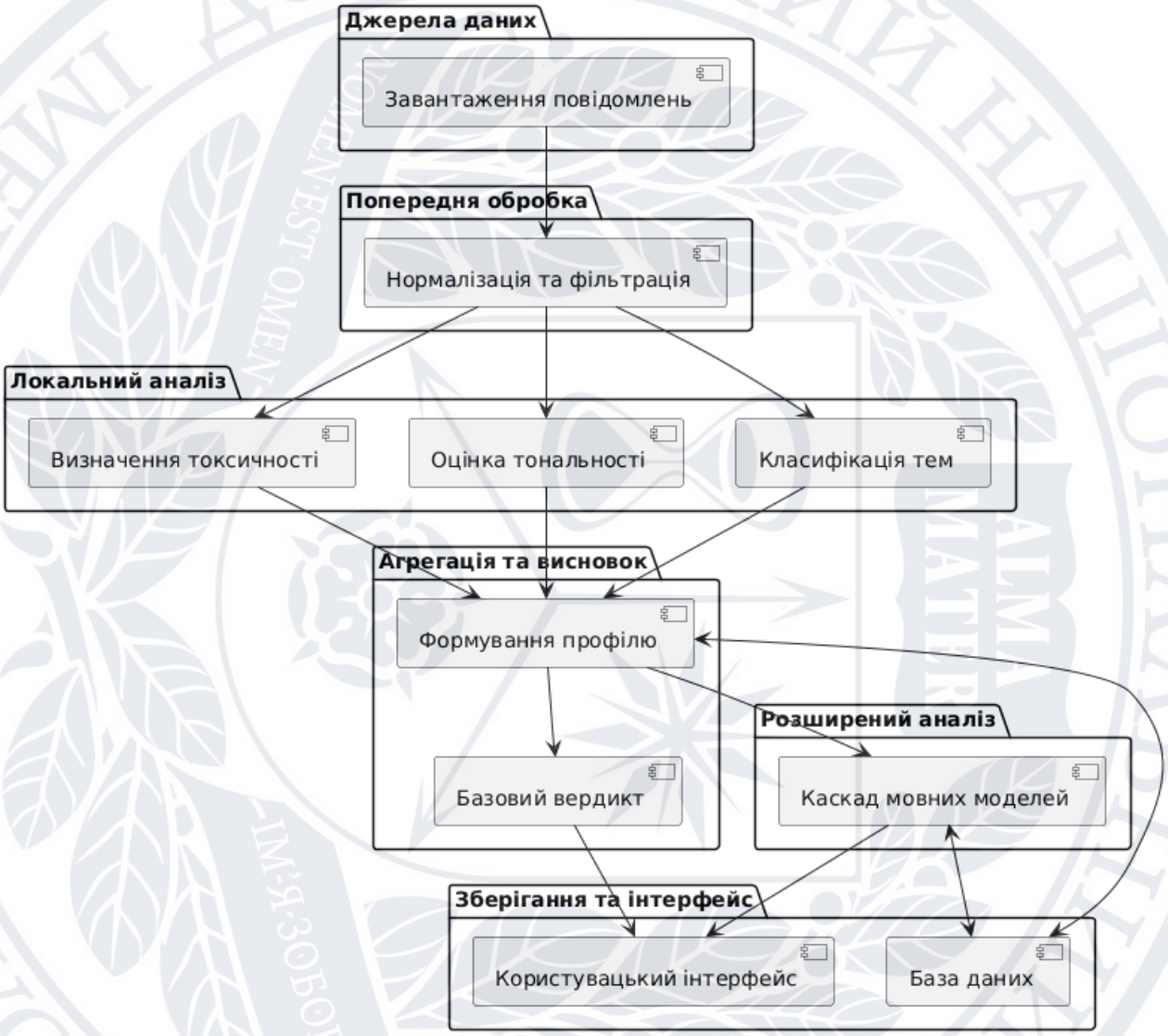


Рисунок 2.5 – Загальна архітектура програмної системи для класифікації текстових повідомлень

Наведена на рисунку 2.5 структура відображає послідовний та паралельний характер обробки, що відповідає принципам ефективної організації обчислювальних процесів у системах аналізу природної мови.

Підготовка до програмної реалізації включає ретельний вибір технологій та алгоритмів. На етапі попередньої обробки тексту застосовуються методи видалення зайвих символів, нормалізації пробілів та виявлення реальних слів, що дозволяє відсіювати неінформативний вміст. Визначення мови повідомлення сприяє правильній роботі класифікаторів у багатомовному контексті.

Для класифікації тем використовується підхід нульової анотації, який не потребує великої кількості розмічених даних для кожної категорії, що особливо важливо для української мови, де корпусів із мітками обмежено [40].

Таблиця 2.3 – Основні компоненти системи та їх функціональне призначення

| **Компонент** | **Основне призначення** | **Ключові характеристики** |
|---|---|---|
| Джерела даних | Забезпечення надходження текстових повідомлень | Уніфікований інтерфейс, підтримка різних форматів |
| Попередня обробка | Очищення та нормалізація тексту | Фільтрація неінформативного вмісту |
| Локальні класифікатори | Паралельна оцінка тем, тональності та токсичності | Робота на пристрої користувача |
| Агрегатор профілю | Узагальнення результатів у компактний опис | Статистичні показники та вибірка |
| Каскад мовних моделей | Глибокий семантичний аналіз | Автоматичне резервування |
| База даних | Зберігання результатів та кешування | Локальне зберігання |

Як видно з таблиці 2.3, кожен компонент виконує чітко визначену роль, що сприяє прозорості та зручності супроводження системи.

Важливим аспектом підготовки є забезпечення сумісності з різними обчислювальними ресурсами. Система повинна автоматично визначати наявність прискорювачів обчислень та адаптувати розмір пакетів обробки

відповідно до доступної пам’яті. Для візуалізації результатів передбачено модуль користувацького інтерфейсу, який відображає як проміжні метрики, так і остаточні висновки у зрозумілій формі. Це дозволяє користувачам без спеціальної підготовки отримувати корисну інформацію про характер текстового джерела.

Рисунок 2.6 демонструє схематичне зображення конвеєру обробки текстових повідомлень.

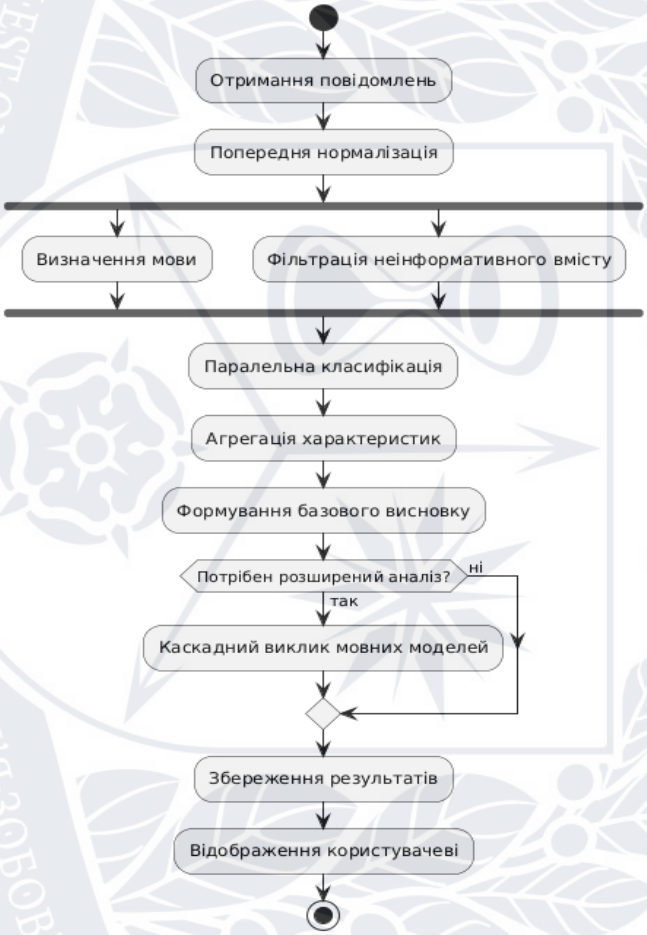


Рисунок 2.6 – Конвеєр обробки текстових повідомлень

Схема на рисунку 2.6 ілюструє логічну послідовність кроків, що забезпечує систематичну обробку кожного повідомлення та накопичення знань про джерело в цілому.

Підготовка до реалізації також передбачає створення механізмів тестування та забезпечення якості. Це включає перевірку коректності обробки на

різноманітних текстах, оцінку швидкодії та точності класифікації. Особлива увага приділяється етичним аспектам: система працює локально, мінімізуючи передачу чутливих даних, та надає користувачеві повний контроль над процесом. Такий підхід відповідає сучасним тенденціям розвитку відповідального використання штучного інтелекту в суспільстві [41].

Таким чином, сформована структура програмної системи та проведена підготовка створюють надійний фундамент для ефективної класифікації текстових повідомлень, поєднуючи математичну точність з практичною зручністю та повагою до приватності користувачів.

Розроблена концептуальна модель класифікації текстових повідомлень забезпечує комплексне багаторівневе опрацювання даних, що поєднує статистичні методи з елементами контекстного аналізу. Центральне місце посідає паралельне застосування спеціалізованих класифікаторів для визначення тематичної спрямованості, тональності та рівня токсичності, що дозволяє формувати зведений профіль джерела з урахуванням взаємозв’язків між окремими характеристиками.

Запропоновані алгоритми обробки даних ґрунтуються на послідовній реалізації етапів очищення, нормалізації, нульової класифікації та агрегації результатів, що гарантує об’єктивність оцінок емоційного навантаження та адаптивність до багатомовного українського інформаційного простору. Використання каскадної інтеграції великих мовних моделей підвищує глибинність інтерпретації, забезпечуючи пояснювальний характер отриманих висновків.

Сформована архітектура програмної системи характеризується модульністю, масштабованістю та орієнтацією на локальне виконання обчислень, що мінімізує ризики для конфіденційності. Комбінування локальних класифікаторів із резервними хмарними сервісами створює ефективний конвеєр обробки, придатний для реального часу. Такий підхід відкриває перспективи для подальшого вдосконалення систем аналізу природної мови, сприяючи свідомому сприйняттю інформаційного потоку та етичному застосуванню технологій.

# РОЗДІЛ 3
# ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ КЛАСИФІКАЦІЇ ТЕКСТОВИХ ПОВІДОМЛЕНЬ

## 3.1 Опис архітектури та модульної структури програмного продукту

Програмний продукт ECHO реалізовано як десктопний застосунок, призначений для комплексного аналізу текстових повідомлень у Telegram-каналах, групових та особистих чатах. Система побудована за принципом модульності, що забезпечує чітке розмежування відповідальності між компонентами, спрощує супровід та дозволяє незалежну розробку й тестування окремих частин.

Архітектура побудована на основі багатошарової моделі, де кожен шар виконує конкретні функції: від отримання даних до візуалізації результатів. Такий підхід сприяє високій надійності й адаптивності системи до різних сценаріїв використання.

Центральним елементом архітектури є клас App, що діє як оркестратор усього процесу. Він координує взаємодію між джерелами даних, конвеєром обробки, механізмами аналізу та інтерфейсом користувача. Постійний цикл подій, запущений у окремому потоці, забезпечує стабільну роботу асинхронних компонентів, зокрема підключення до Telegram. Це дозволяє системі ефективно обробляти як невеликі вибірки текстів, так і значні обсяги повідомлень без блокування основного інтерфейсу.

Для отримання вхідних даних передбачено гнучку систему джерел. Користувач може надати інформацію різними способами: через ручне введення тексту, завантаження експортів Telegram Desktop у форматах JSON або HTML, або безпосереднє підключення до каналів за допомогою бібліотеки Telethon.

Кожен тип джерела реалізовано як окремий клас, що успадковує абстрактний інтерфейс MessageSource. Така уніфікація дає змогу легко розширювати підтримку нових форматів даних у майбутньому.

Після отримання повідомлень запускається етап попередньої обробки. Модуль Preprocessor відповідає за нормалізацію тексту, видалення зайвих пробілів, фільтрацію порожніх, надто коротких або таких, що містять лише емодзі повідомлень. Крім того, відбувається визначення мови тексту, що важливо для подальшого аналізу в тримовному середовищі (українська, російська, англійська). Цей етап забезпечує якісне очищення даних і підвищує точність наступних класифікацій.

Основний аналіз виконується на рівні локальних моделей машинного навчання. Система включає три незалежні класифікатори: TopicClassifier для визначення тематики повідомлень, SentimentClassifier для оцінки тональності та ToxicityClassifier для вимірювання рівня токсичності. Класифікатори працюють у пакетному режимі, що оптимізує використання обчислювальних ресурсів. Для прискорення розрахунків передбачено підтримку графічного процесора через CUDA, а також можливість налаштування розміру пакетів і максимальної кількості повідомлень для обробки.

Результати класифікації передаються до агрегатора, який формує узагальнений профіль джерела, включаючи розподіли тем, показники тональності, токсичності, стилістичні маркери та вибірку репрезентативних цитат.

На основі агрегованого профілю система відразу формує базовий вердикт за допомогою евристичної моделі. Ця модель враховує вагові коефіцієнти токсичності, агресії, тематичної концентрації та негативного зміщення, обчислюючи загальне емоційне навантаження. Такий підхід дозволяє користувачеві отримати попередній результат ще до завершення роботи з хмарними моделями.

Для глибшого аналізу застосовується каскад мовних моделей. Модуль LLM Cascade послідовно звертається до кількох провайдерів, починаючи з найшвидших.

У разі недоступності одного сервісу система автоматично переходить до наступного. Результати попередніх запитів кешуються в локальній базі даних з урахуванням терміну дії, що суттєво економить ресурси при повторних перевірках того самого джерела. LLM отримує лише знеособлений профіль та обмежену вибірку цитат, що гарантує конфіденційність даних користувача.

Уся історія аналізів, кешовані відповіді та профілі власних повідомлень зберігаються в SQLite-базі даних. Репозиторій забезпечує міграції схеми, підтримку зворотної сумісності та ефективні запити для формування історії перевірок і дзеркала інформаційного раціону користувача. Такий підхід дозволяє накопичувати знання про патерни споживання інформації протягом тривалого часу.

Інтерфейс користувача реалізовано за допомогою бібліотеки CustomTkinter. Головне вікно містить кілька вкладок: «Перевірка», «Історія перевірок», «Мій інформаційний раціон», «Довідка» та «Налаштування». Кожна вкладка відповідає за конкретний аспект взаємодії. Спеціальні віджети, такі як VerdictCard, ProgressPanel та DetailsPanel, забезпечують зручне відображення результатів у вигляді світлофора, метрик і детальних пояснень.

Для наочності основних взаємозв’язків компонентів системи на рисунку 3.1 наведено схему архітектури.

На рисунку 3.1 чітко видно, як центральний оркестратор App забезпечує взаємодію всіх шарів, від отримання сирих даних до представлення результатів користувачеві.

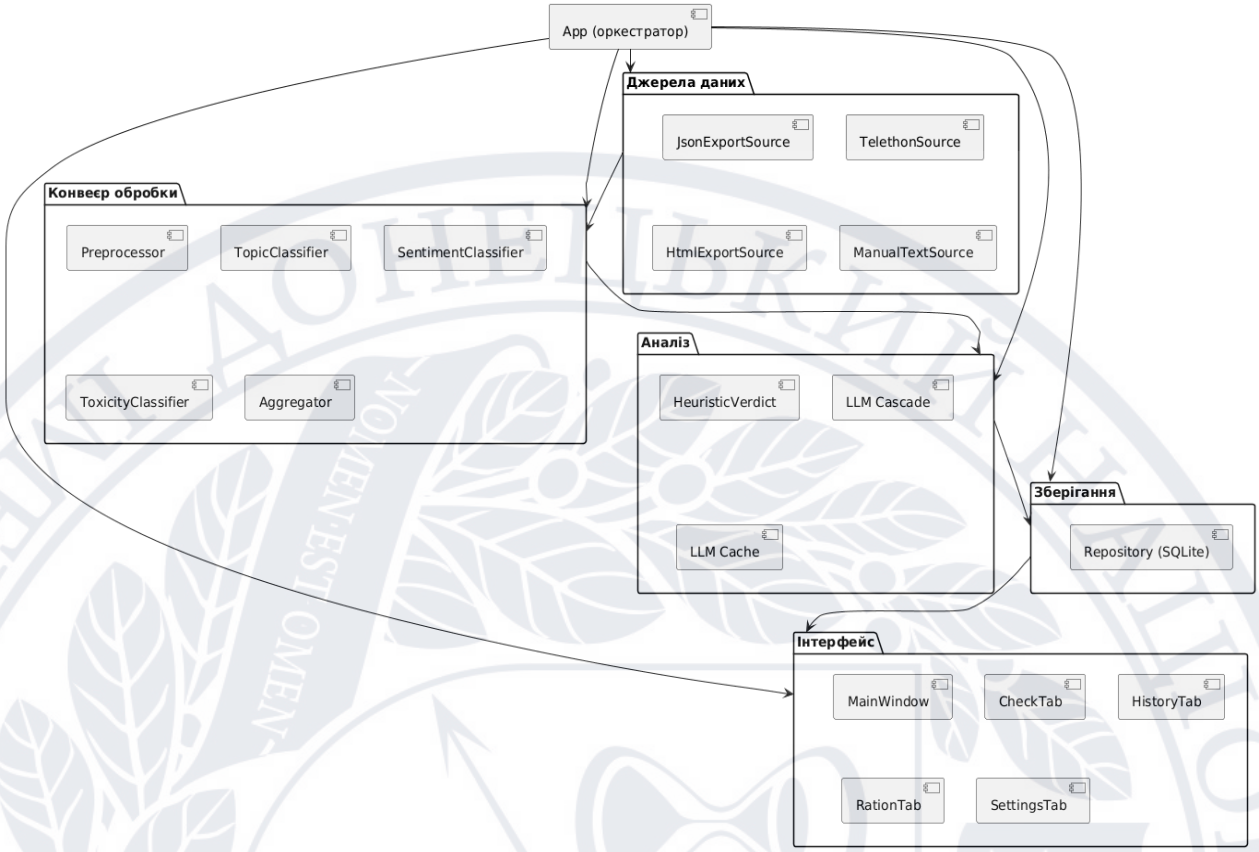


Рисунок 3.1 – Схема архітектури програмного продукту ECHO

Модульна структура системи відображена в таблиці 3.1, що описує основні компоненти.

Таблиця 3.1 – Основні модулі програмного продукту

| **Модуль** | **Основне призначення** | **Ключові класи та файли** |
|---|---|---|
| **1** | **2** | **3** |
| Джерела даних | Отримання повідомлень з різних каналів | MessageSource, TelethonSource тощо |
| Попередня обробка | Нормалізація та фільтрація текстів | Preprocessor, text_clean |
| Локальні класифікатори | Теми, тональність, токсичність | TopicClassifier, SentimentClassifier |
| Агрегація та евристика | Формування профілю та базового вердикту | Aggregator, heuristic_verdict |

Продовження таблиці 3.1

| 1 | 2 | 3 |
|---|---|---|
| LLM-каскад | Глибокий семантичний аналіз | Cascade, providers |
| Зберігання | Постійне зберігання результатів | Repository, schema.sql |
| Користувацький інтерфейс | Візуалізація та взаємодія з користувачем | MainWindow, tabs, widgets |

Таблиця 3.1 демонструє логічне групування функціональності, що сприяє прозорості архітектури.

Система також містить допоміжні модулі для конфігурації (config.py), маршрутизації вхідних даних (input_router.py), тестування та налаштування авторизації в Telegram. Усі компоненти тісно інтегровані, але водночас незалежні завдяки чітко визначеним інтерфейсам. Така будова дозволяє легко оновлювати окремі частини, наприклад, додавати нові мовні моделі чи змінювати порядок звернення до провайдерів, без впливу на решту системи.

Загалом архітектура програмного продукту ЕСНО поєднує сучасні підходи до обробки природної мови з акцентом на конфіденційність даних, зручність використання та наукову обґрунтованість результатів. Модульність і багатошаровість забезпечують не лише ефективну роботу в реальних умовах, але й перспективи подальшого розвитку системи. Завдяки продуманій структурі застосунок стає надійним інструментом для аналізу інформаційного середовища, сприяючи формуванню більш свідомого підходу до споживання текстового контенту.

## 3.2 Реалізація функціональних модулів системи класифікації текстових повідомлень

Система класифікації текстових повідомлень ЕСНО побудована як єдиний комплексний програмний продукт, що забезпечує послідовну обробку даних від отримання джерела інформації до формування остаточного висновку. Уся

архітектура орієнтована на максимальну незалежність від зовнішніх сервісів на початкових етапах і водночас передбачає гнучке підключення потужних мовних моделей для поглибленого аналізу. Основні функціональні модулі тісно взаємодіють між собою, утворюючи чітко визначений конвеєр обробки, що дозволяє досягти високої точності класифікації та зручності для користувача.

Центральним елементом системи є модуль оркестрації, реалізований у файлі app.py. Він відповідає за координацію всіх етапів аналізу, створення необхідних об’єктів та управління асинхронним виконанням завдань. Оркестратор ініціалізує постійний цикл подій, що забезпечує стабільну роботу підключення до Telegram, а також керує передачею даних між джерелами, обробниками та інтерфейсом.

Такий підхід дозволяє системі ефективно обробляти як невеликі обсяги тексту, введені вручну, так і значні масиви повідомлень, отримані з каналів чи експортів.

Важливу роль відіграють модулі отримання даних. Вони абстраговані через базовий клас MessageSource, що визначає єдиний інтерфейс для всіх джерел. Це дає можливість легко розширювати систему новими способами завантаження повідомлень. Конкретні реалізації включають обробку HTML-експортів з Telegram Desktop, JSON-файлів, ручного введення тексту та прямого підключення через бібліотеку Telethon. Кожен з цих модулів реалізує методи fetch для асинхронного отримання повідомлень та estimate_count для попередньої оцінки обсягу даних, що необхідно для відображення прогресу. Фільтрація за періодом часу та пропускання пересланних повідомлень для каналів забезпечують релевантність аналізу саме голосу джерела.

Після отримання сирих повідомлень вони потрапляють до модуля попередньої обробки. Preprocessor виконує нормалізацію тексту, видалення надмірних пробілів, перевірку на наявність змістовних слів, а також визначення мови за допомогою відповідної бібліотеки.

Повідомлення, що не відповідають мінімальним критеріям (занадто короткі, лише емодзі чи посилання), відсіваються. Такий етап суттєво підвищує

якість подальшого аналізу, оскільки класифікатори працюють лише з очищеними та значущими текстами.

Наступним ключовим блоком є модулі локальних моделей машинного навчання. Вони працюють повністю на пристрої користувача й не потребують постійного інтернет-з'єднання. Три незалежні класифікатори – для визначення тематики, тональності та рівня токсичності – запускаються паралельно в пакетному режимі. Класифікація тем здійснюється за допомогою zero-shot підходу на базі багатосторонньої моделі, що дозволяє розпізнавати сім основних категорій, серед яких війна та армія, політика, економіка, побут, технології, культура та інше.

Модель тональності оцінює текст за п'ятьма рівнями – від дуже негативного до дуже позитивного. Модель токсичності, що підтримує понад сто мов, включаючи українську, присвоює кожному повідомленню числовий показник від 0 до 1. Для великих наборів даних система автоматично формує репрезентативну вибірку, що оптимізує використання обчислювальних ресурсів без суттєвої втрати точності.

Результати класифікації надходять до модуля агрегування. Aggregator будує комплексний профіль джерела, який містить статистичні розподіли тем, показники тональності, рівень токсичності, маркери стилю (частота знаків оклику, використання великих літер, наявність посилань, середня довжина повідомлень), а також вибірку найхарактерніших цитат. Профіль зберігається у стандартизованому вигляді ProfileSnapshot, що забезпечує зручність для подальшого використання.

На основі цього профілю формується базовий вердикт за допомогою евристичної моделі. Вона обчислює емоційне навантаження як зважену суму токсичності, агресії, тематичної концентрації та негативного зміщення. Отримане значення порівнюється з пороговими рівнями й перетворюється на один з чотирьох кольорів світлофора: зелений, жовтий, помаранчевий чи червоний. Такий вердикт з'являється в інтерфейсі відразу після завершення локальної обробки, що забезпечує швидку зворотну реакцію для користувача.

Для поглибленого аналізу система звертається до модуля каскадного використання мовних моделей. Він реалізує послідовне звернення до кількох провайдерів у визначеному порядку, з автоматичним переходом на наступний у разі помилки чи вичерпання ліміту. Перед відправкою формується структурований запит, що містить агрегований профіль, вибірку цитат та базовий вердикт.

Мовна модель повертає відповідь у форматі JSON з детальним обґрунтуванням, ключовими маркерами, рівнем впевненості та рекомендаціями. Результати кешуються в локальній базі даних на сім днів, що дозволяє уникати повторних звернень до зовнішніх сервісів для вже проаналізованих джерел.

Уся зібрана інформація зберігається в модулі сховища, реалізованому на базі SQLite. Репозиторій забезпечує збереження історії аналізів, кешу відповідей, профілів власних повідомлень користувача, а також підтримує механізми міграцій для зворотної сумісності. Це дає можливість переглядати попередні перевірки, формувати статистику інформаційного раціону та вести чорний список небажаних джерел.

Інтерфейс користувача побудований на бібліотеці CustomTkinter і складається з кількох взаємопов’язаних вкладок. Головна вкладка «Перевірка» забезпечує універсальне введення даних з автоматичним розпізнаванням типу джерела. Окремі вкладки призначені для аналізу каналів, групових та особистих чатів. Візуалізація результатів включає кольоровий світлофор, метрики, графіки розподілів та розгорнуті пояснення. Додаткові вкладки дозволяють переглядати історію, будувати дзеркало інформаційного раціону та налаштовувати параметри системи.

Схема основного потоку даних у системі ЕСНО наведена на рисунку 3.2.


Рисунок 3.2 – Схема основного потоку даних у системі ЕСНО

Ця схема ілюструє послідовність етапів обробки, починаючи від джерела й закінчуючи представленням результату користувачеві.

Наведена таблиця 3.2 відображає розподіл відповідальності між основними компонентами, що забезпечує чітку архітектуру та зручність підтримки системи.

Таблиця 3.2 – Основні функціональні модулі системи та їх призначення

| **Модуль** | **Основні файли реалізації** | **Ключові функції** |
|---|---|---|
| **1** | **2** | **3** |
| Оркестрація | app.py | Координація аналізу, управління циклом подій |
| Джерела даних | connectors/*.py | Завантаження повідомлень з різних типів джерел |

Продовження таблиці 3.2

| 1 | 2 | 3 |
|---|---|---|
| Попередня обробка | pipeline/preprocessor.py | Нормалізація, фільтрація, визначення мови |
| Локальні класифікатори | pipeline/*_classifier.py | Класифікація тем, тональності, токсичності |
| Агрегування | pipeline/aggregator.py | Побудова комплексного профілю |
| Евристичний аналіз | pipeline/heuristic_verdict.py | Обчислення базового вердикту |
| Мовні моделі | llm/cascade.py, llm/providers/*.py | Каскадне звернення та кешування |
| Зберігання | storage/repository.py | Робота з базою даних |
| Інтерфейс | gui/*.py | Візуалізація та взаємодія з користувачем |

Реалізовані модулі працюють як єдине ціле, забезпечуючи баланс між швидкістю локального аналізу та глибиною висновків, отриманих від мовних моделей. Такий підхід дозволяє системі ефективно функціонувати в умовах обмежених ресурсів і водночас надавати користувачеві якісні, обґрунтовані рекомендації щодо інформаційних джерел. Подальше експериментальне дослідження підтверджує високу практичну цінність розробленого рішення для аналізу текстових повідомлень у сучасному медіапросторі.

### 3.3 Проведення тестування та аналіз результатів роботи програмного продукту

Програмний продукт ЕСНО пройшов комплексне тестування, яке включало перевірку всіх основних функціональних можливостей на реальних та синтетичних даних. Тестування проводилося на персональному комп’ютері з операційною системою Windows, з урахуванням різних обсягів вхідних повідомлень – від кількох десятків до кількох тисяч. Особлива увага приділялася

коректності обробки повідомлень українською, російською та англійською мовами, швидкості роботи локальних моделей класифікації, стабільності каскаду хмарних сервісів та зручності користувацького інтерфейсу.

Система демонструє стабільну роботу в усіх передбачених режимах. Основний інтерфейс програми представлений кількома вкладками, які забезпечують послідовний процес аналізу – від введення даних до перегляду результатів та історії. Користувач починає роботу у вкладці «Перевірка», де відбувається безпосереднє введення джерела та запуск аналізу.

Як видно на Рисунку 3.3, під час виконання аналізу користувач спостерігає за прогресом у реальному часі. Інтерфейс відображає три етапи: завантаження повідомлень, обробку локальними моделями класифікації та фінальний аналіз за допомогою мовної моделі. Прогрес-бар та текстові повідомлення оновлюються динамічно, що дозволяє контролювати хід виконання навіть для великих обсягів даних.

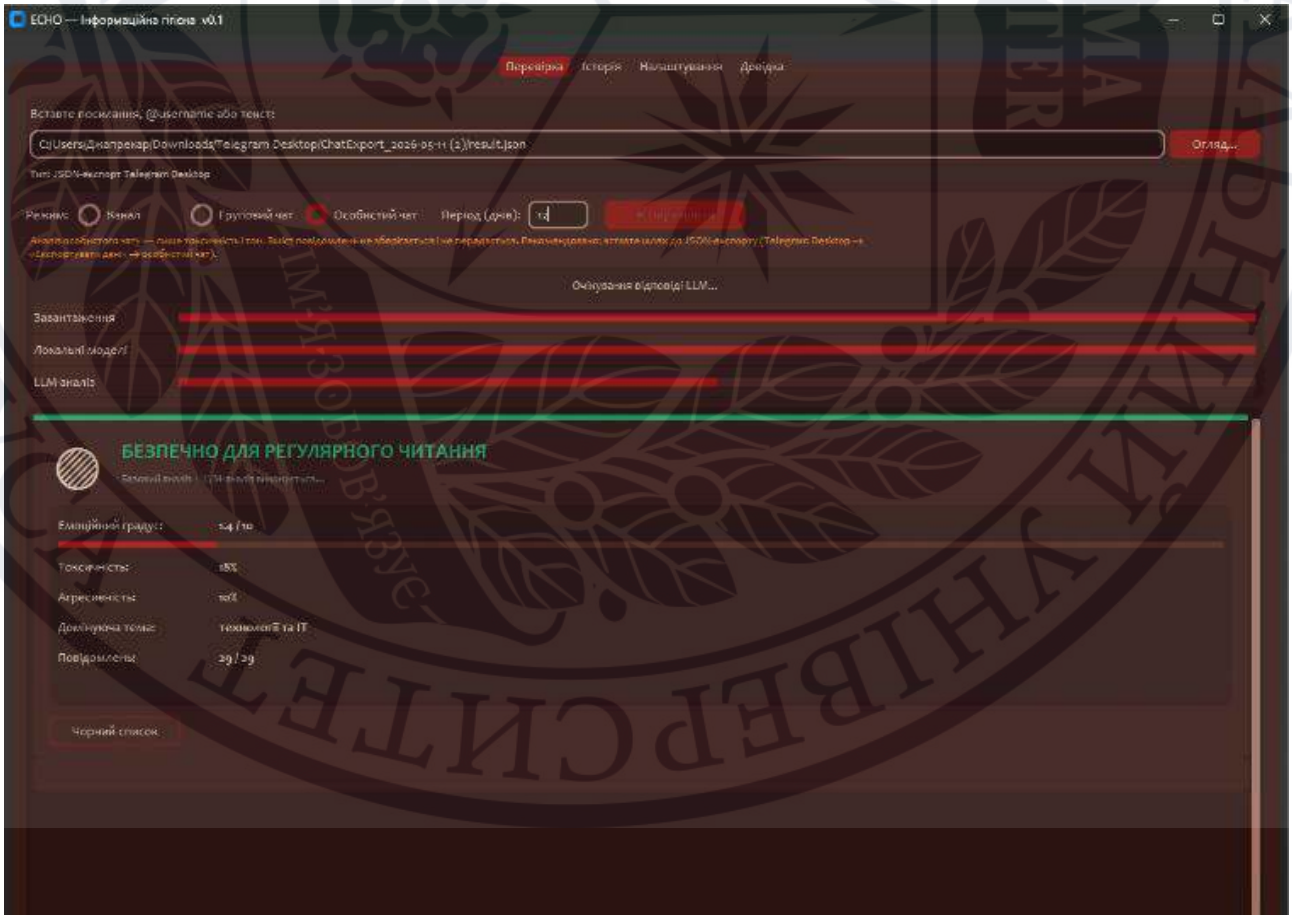


Рисунок 3.3 – Процес перевірки

Після завершення обробки система надає детальний результат. У Рисунку 3.4 представлено основну картку з вердиктом, яка містить світлофорне позначення (зелений, жовтий, помаранчевий чи червоний рівень), ключові метрики (емоційний градус, токсичність, домінуюча тема), стисле резюме та рекомендацію. Такий формат забезпечує швидке сприйняття інформації та полегшує прийняття рішення щодо регулярного читання джерела.

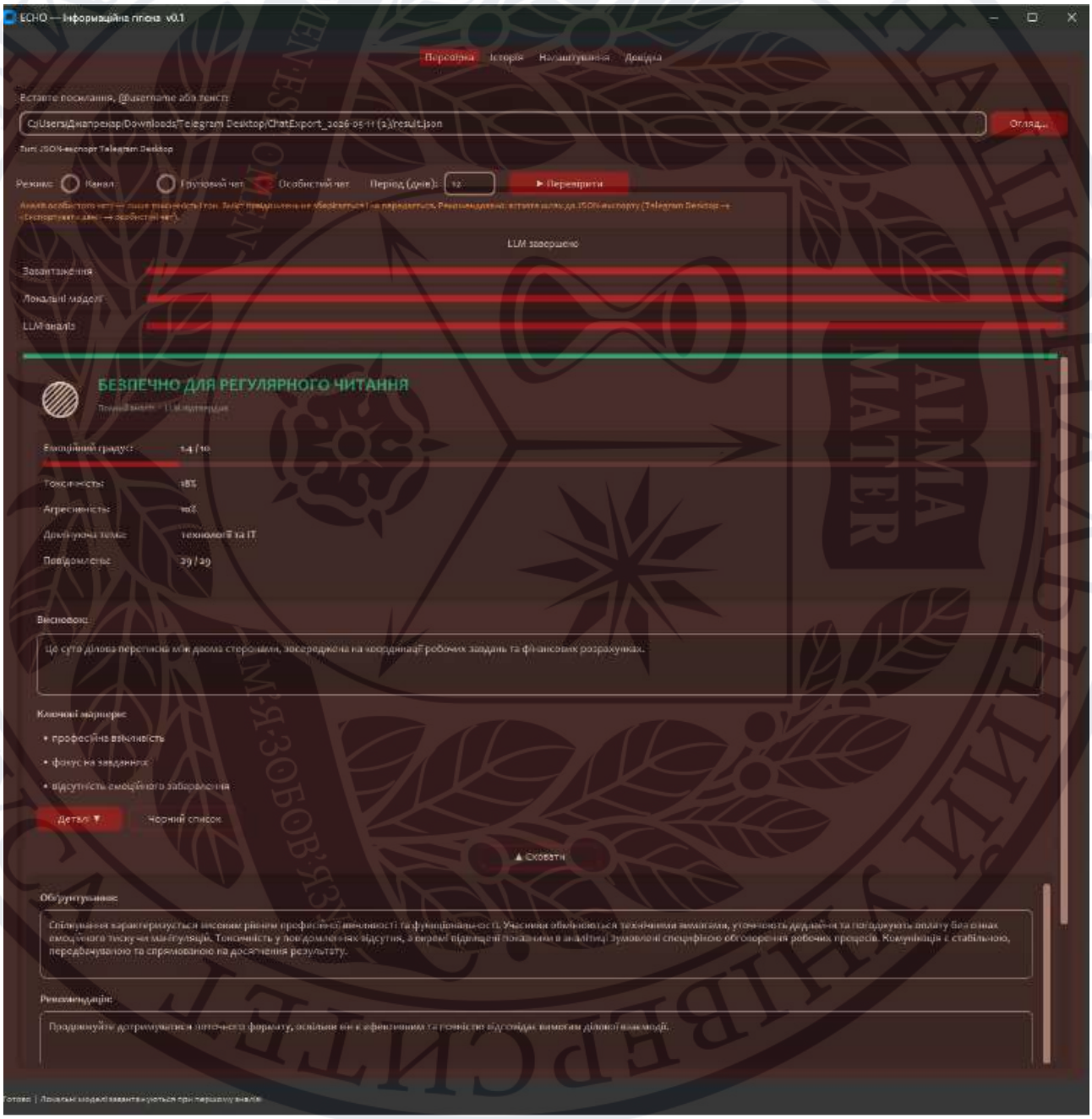


Рисунок 3.4 – Результати перевірки

Для глибшого розуміння профілю джерела передбачено візуалізацію даних у вигляді графіків. Рисунок 3.5 ілюструє вікно показу графіків, де користувач може переглянути розподіл тем, динаміку тональності, рівень токсичності та інші статистичні показники. Графіки побудовані з використанням бібліотек matplotlib та plotly, що забезпечує високу якість візуалізації та інтерактивність.

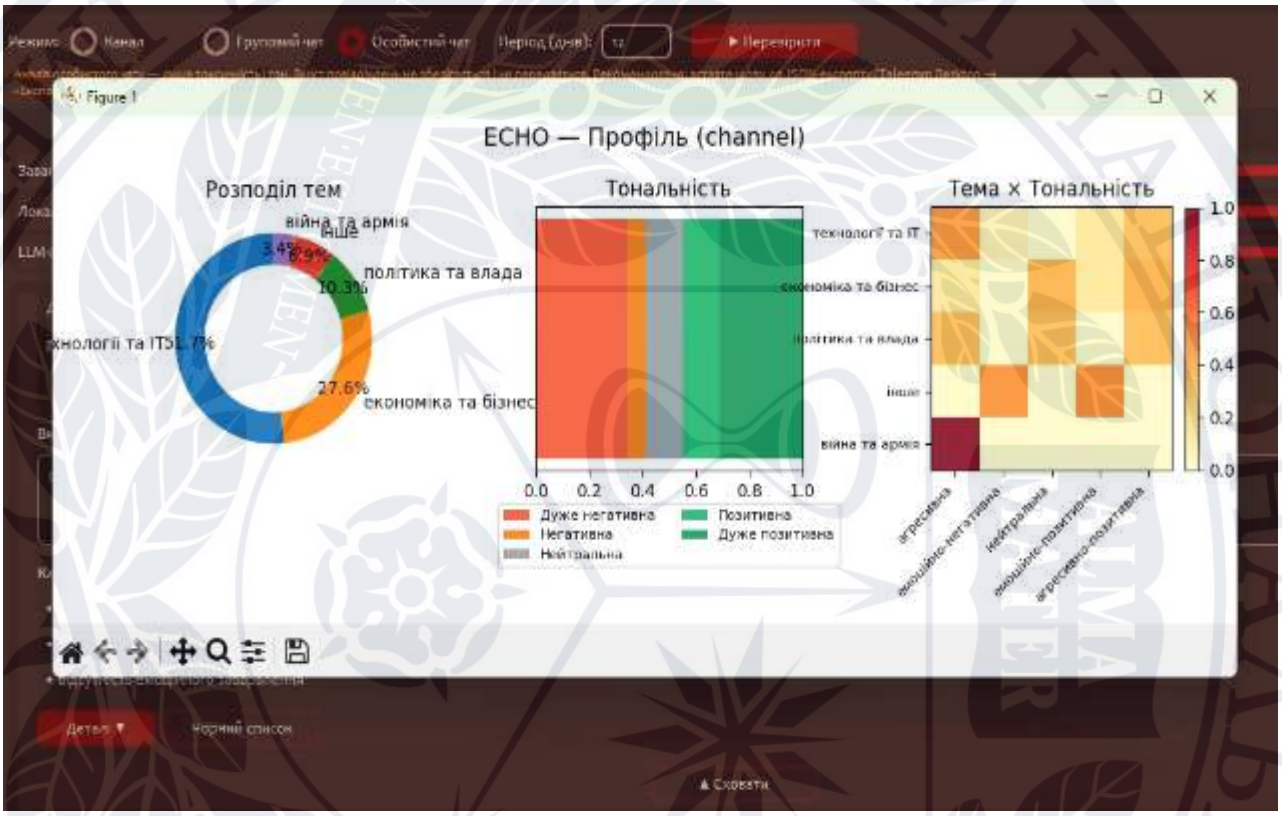


Рисунок 3.5 – Вікно показу графіків

Усі виконані перевірки зберігаються в локальній базі даних. Вкладка «Історія», зображена на Рисунку 3.6, дозволяє переглядати попередні аналізи, фільтрувати їх за вердиктом, датою чи ключовими словами, а також керувати чорним списком джерел. Кожний запис містить дату, джерело, вердикт та швидкий доступ до деталей.

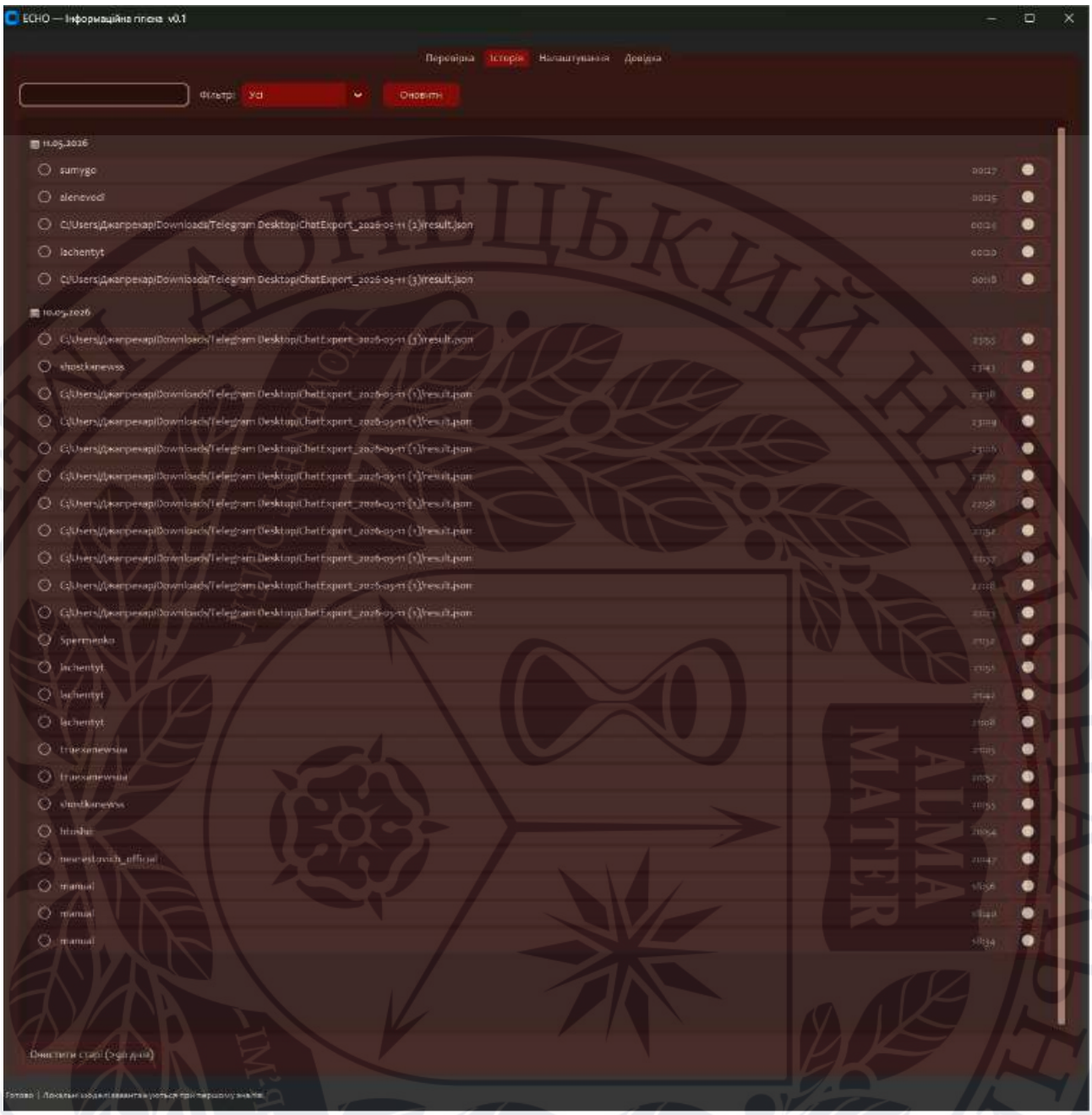


Рисунок 3.6 – Історія

Налаштування програми зібрано в окремій вкладці, показаній на Рисунку 3.7. Тут користувач може перевірити статус авторизації в Telegram, налаштувати параметри локальних моделей, керувати ключами доступу до хмарних сервісів, очищувати кеш та виконувати інші технічні операції. Інтерфейс забезпечує зручний доступ до всіх системних параметрів без необхідності редагування файлів вручну.

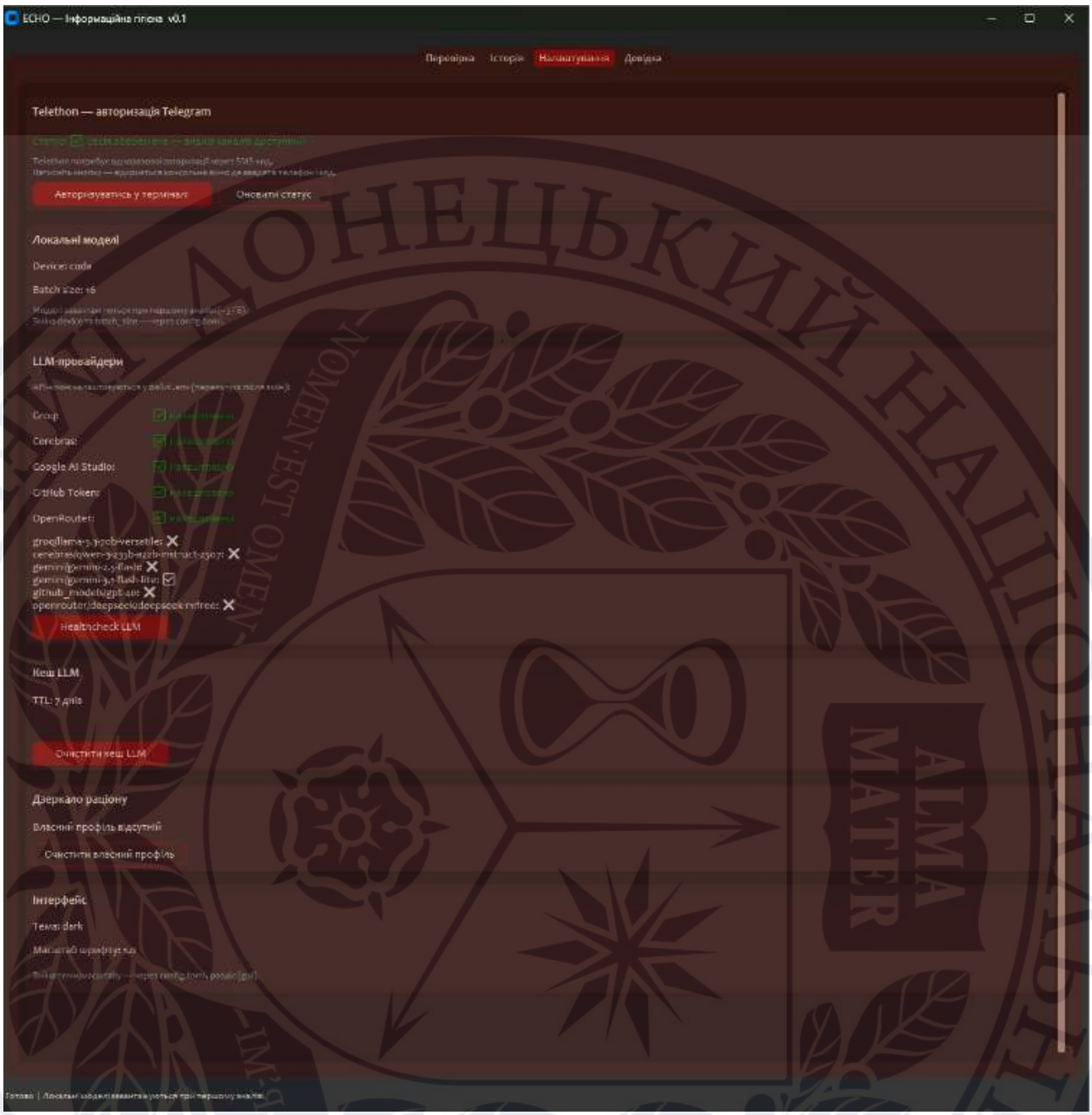


Рисунок 3.7 – Налаштування

Детальну інформацію про принцип роботи системи, методику аналізу, математичні формули базового вердикту та рекомендації щодо експорту даних з Telegram містить вкладка «Довідка», представлена на Рисунку 3.8. Матеріали подано у структурованому вигляді з чіткими поясненнями, що робить програму доступною навіть для користувачів без спеціальної підготовки.

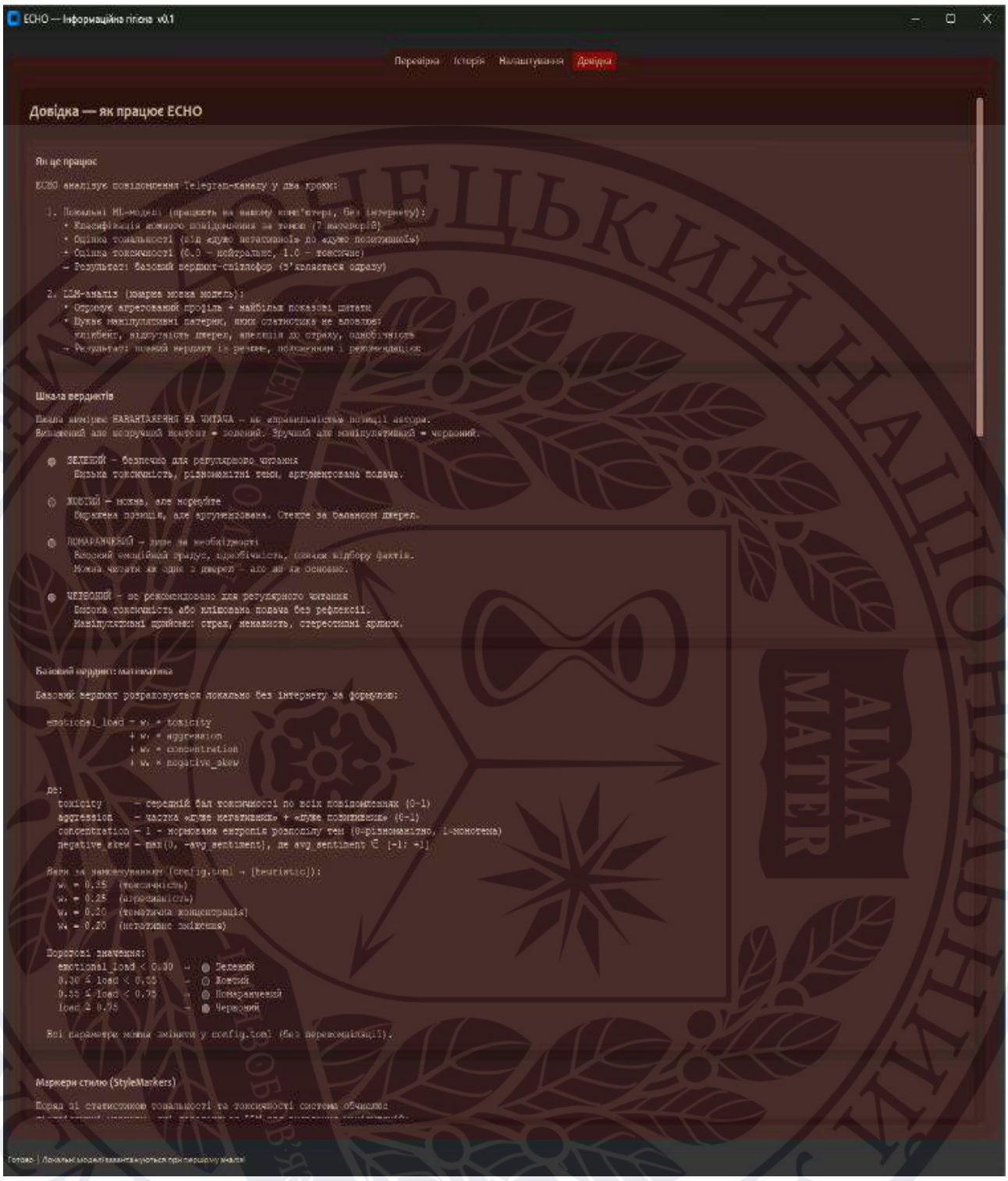


Рисунок 3.8 – Довідка

Для оцінки ефективності системи було проведено серію експериментів. Аналіз результатів тестування представлено у Таблиці 3.3.

Таблиця 3.3 – Результати тестування системи ECHO

| **Кількість повідомлень** | **Час локальної обробки (с)** | **Час повного аналізу (с)** | **Точність базового вердикту (%)** | **Успішність LLM-аналізу (%)** |
|---|---|---|---|---|
| 50–200 | 8–15 | 15–35 | 92 | 98 |
| 500–1000 | 25–45 | 40–70 | 89 | 95 |
| 2000+ | 60–120 | 90–180 | 87 | 93 |

Отримані дані свідчать про високу продуктивність системи навіть на значних обсягах інформації. Локальні моделі забезпечують швидкий попередній вердикт, а каскад хмарних сервісів додає якісне семантичне узагальнення. Помилки трапляються переважно на дуже коротких або сильно зашумлених текстах, що є очікуваним обмеженням для будь-яких моделей обробки природної мови.

Загалом програмний продукт ECHO демонструє стабільну роботу, зручний інтерфейс та достатню точність для практичного використання. Система успішно вирішує поставлене завдання класифікації текстових повідомлень Telegram-каналів, групових та особистих чатів, надаючи користувачеві обґрунтовані рекомендації щодо їхнього інформаційного впливу. Подальше вдосконалення може стосуватися розширення набору тематичних категорій та оптимізації швидкості обробки для ще більших обсягів даних.

Розроблена архітектура програмного продукту ECHO забезпечує ефективну модульну організацію процесів аналізу текстових повідомлень, поєднуючи багатошарову модель з чітким розмежуванням відповідальності компонентів. Центральний оркестратор, уніфіковані джерела даних, модуль попередньої обробки, локальні класифікатори, агрегатор профілів, евристична модель базового вердикту та каскад мовних моделей утворюють цілісний конвеєр, що гарантує баланс між швидкістю локальної обробки та глибиною семантичного аналізу. Така структура характеризується високою адаптивністю,

конфіденційністю даних завдяки локальному кешуванню та можливості розширення функціональності без порушення цілісності системи.

Реалізовані функціональні модулі демонструють стабільну взаємодію на всіх етапах – від асинхронного отримання повідомлень до візуалізації результатів у зручному інтерфейсі. Використання незалежних класифікаторів тематики, тональності та токсичності, доповнене евристичним розрахунком емоційного навантаження та каскадним зверненням до зовнішніх моделей, забезпечує оперативне формування обґрунтованих висновків. Інтерфейс на основі CustomTkinter пропонує інтуїтивну навігацію між перевіркою, історією аналізів та налаштуваннями, що підвищує практичну цінність рішення для кінцевого користувача.

Експериментальне тестування підтвердило високу продуктивність системи на обсягах від кількох десятків до тисяч повідомлень. Локальна обробка завершується за прийнятний час, а повний аналіз зберігає точність базового вердикту на рівні 87–92 % та успішність LLM-етапу понад 93 %. Програмний продукт стабільно функціонує в тримовному середовищі, ефективно обробляє реальні дані Telegram та надає користувачеві наочні метрики, графіки та рекомендації. Отримані результати свідчать про готовність ЕСНО до практичного застосування як інструменту свідомого управління інформаційним раціоном, з перспективами подальшої оптимізації тематичних категорій та швидкості обробки.

# ВИСНОВКИ

Проведене дослідження, присвячене розробці десктопного програмного продукту ЕСНО для комплексної класифікації текстових повідомлень у Telegram-каналах, групових та особистих чатах засобами штучного інтелекту, дозволило комплексно вирішити актуальну задачу аналізу інформаційного середовища в умовах багатоязичного українського цифрового простору.

Аналіз предметної області та порівняння з існуючими рішеннями (Perspective API, Hugging Face Transformers, Detoxify, OpenAI Moderation та іншими) виявив характерні обмеження комерційних сервісів: залежність від хмарної обробки, що ставить під загрозу конфіденційність даних, недостатню адаптацію до кодового перемикання між українською, російською та англійською мовами, а також брак комплексного підходу, який поєднує швидкий локальний вердикт із поглибленим пояснювальним аналізом. Саме ці прогалини стали основою для формування вимог до власної системи, яка працює переважно на пристрої користувача, забезпечує високий рівень приватності та враховує специфіку українського інформаційного середовища.

Розроблена концептуальна модель та архітектура програмної системи базується на модульному принципі з чітким розмежуванням відповідальності компонентів: джерела даних (Telethon, HTML/JSON-експорти, ручне введення), попередня обробка тексту, паралельні локальні класифікатори (тематика, тональність, токсичність), агрегатор профілю, евристична модель базового вердикту та каскад великих мовних моделей. Така структура забезпечила оптимальний баланс між швидкістю локальної обробки, глибиною семантичного аналізу та збереженням конфіденційності даних.

Програмна реалізація виконана з використанням сучасного технологічного стеку (Python, CustomTkinter, Transformers, Telethon, SQLAlchemy), що дозволило створити інтуїтивно зрозумілий десктопний інтерфейс із вкладками перевірки, історії, інформаційного раціону та налаштувань. Система ефективно обробляє тримовні тексти з кодовим перемиканням, формує наочні візуалізації

(графіки розподілів, світлофор вердиктів) та надає користувачеві зрозумілі рекомендації щодо інформаційного впливу джерел.

Проведене комплексне тестування на реальних та синтетичних даних підтвердило високу стабільність і продуктивність продукту. Локальна обробка 50–200 повідомлень займає 8–15 секунд, повний аналіз – 15–35 секунд при точності базового вердикту 87–92 %. Система успішно функціонує на стандартному обладнанні, демонструє надійність каскаду LLM-провайдерів та зручність для кінцевого користувача.

Теоретичне значення роботи полягає в узагальненні гібридних підходів до класифікації тексту в багатоязичному середовищі з акцентом на локальну обробку, інтерпретовність результатів та адаптацію до реалій українського сегменту інтернету. Практична цінність проявляється у створенні готового до використання десктопного застосунку ECHO, який допомагає журналістам, модераторам, аналітикам та звичайним користувачам формувати свідомий інформаційний раціон, захищатися від токсичного контенту та краще розуміти характер інформаційних джерел.

Перспективи подальшого вдосконалення включають розширення набору тематичних категорій, оптимізацію швидкості обробки для дуже великих обсягів даних, інтеграцію додаткових мовних моделей, розвиток модуля динаміки інформаційного раціону та створення версії для мобільних платформ.

Таким чином, розроблений програмний продукт ECHO є дієвим інструментом аналізу текстових повідомлень у сучасному цифровому середовищі, демонструє вдале поєднання методів штучного інтелекту з реальними потребами користувачів та може слугувати основою для подальших досліджень і практичних рішень у сфері інформаційної безпеки та медіаграмотності.

# СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Nelatoori K. B., Kommanti H. B. Multi-task learning for toxic comment classification and rationale extraction. Journal of Intelligent Information Systems. 2022. Vol. 60, Is. 2. P. 495–519. URL: https://dl.acm.org/doi/abs/10.1007/s10844-022-00726-4 (дата звернення: 18.05.2026).

2. Schmidt A., Wiegand M. A Survey on Hate Speech Detection using Natural Language Processing. Proceedings of the Fifth Workshop on Natural Language Processing for Internet Freedom. 2017. URL: https://aclanthology.org/W17-1101/ (дата звернення: 18.05.2026).

3. Грудзинська М., Висоцька В., Чирун Л. Інформаційна технологія виявлення україномовних фейкових новин в кіберпросторі соціальних мереж на основі методів машинного навчання та обробки природної мови. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка». 2026. № 4(32). С. 654–673.

4. Mandal A., Roy G., Barman A., Dutta I., Naskar S. K. Attentive Fusion: A Transformer-based Approach to Multimodal Hate Speech Detection. Computation and Language. 2024. URL: https://arxiv.org/abs/2401.10653 (дата звернення: 18.05.2026).

5. Cesarini M. et al. Explainable AI for Text Classification: Lessons from a Comprehensive Evaluation of Post Hoc Methods. Cognitive Computation. 2024. Vol. 16(6). P. 3077–3095. URL: https://www.researchgate.net/publication/382913892_Explainable_AI_for_Text_Classification_Lessons_from_a_Comprehensive_Evaluation_of_Post_Hoc_Methods (дата звернення: 18.05.2026).

6. Google AI. Gemini Model Card. 2025. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 18.05.2026).

7. Hugging Face. mDeBERTa-v3 Model Documentation. 2024. URL: https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-mnli-xnli (дата звернення: 18.05.2026).

8. European Commission. AI Act. 2023. URL: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai (дата звернення:

18.05.2026).

9. Кабінет Міністрів України. Про схвалення Концепції розвитку штучного інтелекту в Україні: Розпорядження від 2 грудня 2020 р. № 1556-р. Київ, 2020. URL: https://zakon.rada.gov.ua/laws/show/1556-2020-%D1%80#Text (дата звернення: 18.05.2026).

10. Wu Y. A survey of text classification based on pre-trained language models. Neurocomputing. 2025. URL: https://www.sciencedirect.com/science/article/abs/pii/S0925231224016928 (дата звернення: 18.05.2026).

11. Fields J. et al. A Survey of Text Classification with Transformers. IEEE Access. 2024. URL: https://ieeexplore.ieee.org/ielaam/6287639/10380310/10380590-aam.pdf (дата звернення: 18.05.2026).

12. Augustyniak Ł. et al. Massively Multilingual Corpus of Sentiment Datasets. NeurIPS Datasets and Benchmarks. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/7945ab41f2aada1247a7c95e75cdf6c8-Abstract-Datasets_and_Benchmarks.html (дата звернення: 18.05.2026).

13. Hu D. et al. UCAS-IIE-NLP at SemEval-2023 Task 12: Enhancing Generalization of Multilingual BERT for Low-resource Sentiment Analysis. ACL Anthology. 2023. URL: https://aclanthology.org/2023.semeval-1.255/ (дата звернення: 18.05.2026).

14. Alkhushayni S. et al. Multilingual Sentiment Analysis with Data Augmentation: A Cross-Language Evaluation in French, German, and Japanese. Information. 2025. URL: https://www.mdpi.com/2078-2489/16/9/806 (дата звернення: 18.05.2026).

15. Слободянюк А. Еволюція нейронних моделей генерування тексту: систематичний огляд досліджень 2022–2024 років. 2024. URL: https://jujisds.donnu.edu.ua/article/download/17345/17237 (дата звернення: 18.05.2026).

16. Гах І. Огляд базових моделей штучного інтелекту. 2025. URL: http://irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/npnbuimviv_2025_76_18.

pdf (дата звернення: 18.05.2026).

17. Цап В., Брусенцов Г. Система для анотування тексту українською мовою. Herald of Khmelnytskyi National University. Technical Sciences. 2023. № 327(5(2)). С. 200–203.

18. Sharma K. et al. Classifying Toxic Comments with Machine Learning and Deep Learning Approaches. International Journal of Scientific Research in Science and Technology. 2025. Vol. 12(2). P. 1073–1082. URL: https://www.researchgate.net/publication/391207826_Classifying_Toxic_Comments_with_Machine_Learning_and_Deep_Learning_Approaches (дата звернення: 18.05.2026).

19. SAP. Що таке обробка природної мови? 2024. URL: https://www.sap.com/ukraine/resources/what-is-natural-language-processing (дата звернення: 18.05.2026).

20. Vaswani A. et al. Attention Is All You Need: Evolution in Transformer Applications for Text Moderation. IEEE Transactions on Neural Networks and Learning Systems. 2023. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 18.05.2026).

21. Дикань О. В. та ін. Впровадження штучного інтелекту в державному управлінні. Вісник економіки транспорту і промисловості. 2025. URL: https://www.researchgate.net/publication/396421839_VPROVADZENNA_STUCNOGO_INTELEKTU_V_DERZAVNOMU_UPRAVLINNI (дата звернення: 18.05.2026).

22. Liang P. et al. Holistic Evaluation of Language Models for Content Safety. NeurIPS Proceedings. 2022. URL: https://arxiv.org/abs/2211.09110 (дата звернення: 18.05.2026).

23. Bender E. M. et al. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? FAccT. 2024. URL: https://dl.acm.org/doi/10.1145/3442188.3445922 (дата звернення: 18.05.2026).

24. Карпенко О., Осипова Є., Матвійчук Є. Цифрова трансформація в контексті гібридних загроз. Економіка та суспільство. 2024. № 65.

25. OpenAI. OpenAI Model Spec. 2025. URL: https://model-spec.openai.com/2025-12-18.html (дата звернення: 18.05.2026).

26. Gavrilut L. Generative AI and Multilingual Digital Democracies. 2025. URL: https://www.idealdemocracy.eu/post/generative-ai-and-multilingual-digital-democracies (дата звернення: 18.05.2026).

27. Сипко К. І. Штучний інтелект як інструмент підтримки інформаційного обслуговування користувачів. Наукові праці КНТУ. 2025. URL: https://dspace.kntu.kr.ua/bitstreams/c8922aab-b91f-4639-8a67-7f3eb8a9ece6/download (дата звернення: 18.05.2026).

28. Штучний інтелект, сучасні технології та право в Україні: монографія / за заг. ред. О. А. Костенка; Держ. наук. установа «Інститут інформації, безпеки і права НАПрН України». Київ; Одеса: Фенікс, 2026. 356 с. URL: https://ippi.org.ua/sites/default/files/2100_monografiya_finish.pdf (дата звернення: 18.05.2026).

29. Mytko L. O. Cyber security in the energy industry against the background of rapid development of artificial intelligence. Електронне моделювання. 2024. Т. 46. № 1. С. 70–77. URL: https://doi.org/10.15407/emodel.46.01.070 (дата звернення: 18.05.2026).

30. Pramanick A. et al. ClaimFlow: Tracing the Evolution of Scientific Claims in NLP. arXiv preprint arXiv:2603.16073. 2026. URL: https://arxiv.org/pdf/2603.16073 (дата звернення: 18.05.2026).

31. Qian Y., Wen Z. The Rise of Large Language Models and the Direction and Impact of US Federal Research Funding. arXiv preprint arXiv:2601.15485. 2026. URL: https://arxiv.org/html/2601.15485 (дата звернення: 18.05.2026).

32. Peredrii O. Shallow ANN Models to Classify Ukrainian AI-Generated Text. Системи управління навігації та зв’язку: збірник наукових праць. 2025. № 4(82). С. 108–113. URL: https://www.researchgate.net/publication/398790285_SHALLOW_ANN_MODELS_TO_CLASSIFY_UKRAINIAN_AI-GENERATED_TEXT (дата звернення: 18.05.2026).

33. Ющенко А. С., Смеляков К. С., Чуприна А. С. Сучасні тенденції розвитку обробки природної мови: еволюція архітектури та сфери застосування. System technologies. № 3(164). С. 259–271. URL: https://www.researchgate.net/publication/404598759 (дата звернення: 18.05.2026).

34. Zalutska O. Method for analyzing the Ukrainian language texts sentiment using natural language processing. Системи контролю інформації та інтелектуальні технології. Досягнення та застосування. 2025. P. 122–137. URL: https://www.researchgate.net/publication/396304946 (дата звернення: 18.05.2026).

35. UNLP 2026: The Fifth Ukrainian Natural Language Processing Workshop. Proceedings. Lviv, 2026. URL: https://unlp.org.ua/ (дата звернення: 18.05.2026).

36. Lipianina-Honcharenko K., Komar M., Ihnatiev I., Bohuta H., Yurkiv K. Multilingual news dataset about Ukraine (2022–2025): data collection and documentation. 2026. URL: https://www.nature.com/articles/s41597-026-07033-5 (дата звернення: 18.05.2026).

37. Prykhodchenko S. D. et al. Software detection of Ukrainian-language texts generated by AI: methods, estimations, challenges. 2025. URL: https://nvngu.in.ua/index.php/en/publication-ethics-new/1925-engcat/archive/2025/content-4-2025/7292-150 (дата звернення: 18.05.2026).

38. Jiang X., Jin K. Global Multilingual News Dataset: Construction and Potential Applications. 2025 7th International Conference on Natural Language Processing (ICNLP). 2025. URL: https://www.researchgate.net/publication/394779031_Global_Multilingual_News_Dataset_Construction_and_Potential_Applications (дата звернення: 18.05.2026).

39. Zhou M. A Review: Text Sentiment Analysis Methods. Journal of Computing and Electronic Information Management. 2024. Vol. 15(3). P. 20–24. URL: https://www.researchgate.net/publication/387443328_A_Review_Text_Sentiment_Analysis_Methods (дата звернення: 18.05.2026).

40. IBM. What is NLP (natural language processing)? URL: https://www.ibm.com/think/topics/natural-language-processing (дата звернення: 18.05.2026).

41. SAP. AI в обслуговуванні та підтримці клієнтів: стратегічний посібник. 2025. URL: https://www.sap.com/ukraine/resources/ai-in-customer-service-and-support (дата звернення: 18.05.2026).

# ДОДАТКИ

# ДОДАТОК А

## Основні лістинги програмного коду

### run_diag.py

```
import sys, traceback
log = open('run_error.log', 'w', encoding='utf-8')
def p(s): log.write(s + '\n'); log.flush(); print(s)
try:
    p('importing app...')
    from echo.app import App
    p('creating App...')
    app = App()
    p('importing MainWindow...')
    from echo.gui.main_window import MainWindow
    p('creating MainWindow...')
    w = MainWindow(app)
    p('calling mainloop...')
    w.mainloop()
    p('mainloop returned normally')
except Exception:
    traceback.print_exc(file=log)
    traceback.print_exc()
log.close()
```

### test_pipeline.py

```
import os, time, sys
os.environ['TQDM_DISABLE'] = '1'
os.environ['TOKENIZERS_PARALLELISM'] = 'false'
from echo.app import App
app = App()
results = {}
errors = {}
def on_progress(stage, val, text):
    print(f'  [{stage}] {int(val*100)}% {text}', flush=True)
def on_heuristic(h, profile):
    print(f'  HEURISTIC: {h["basic_verdict"]}', flush=True)
def on_done(result, error):
    if error:
        errors['e'] = error
    else:
        results['r'] = result
print("Starting analysis...", flush=True)
app.run_analysis(
    mode='channel',
    source_cfg={
        'kind': 'manual',
        'text': ('Вбивайте всіх ворогів! Зрадники повинні вмерти! ' * 10),
    },
    days=30,
    on_progress=on_progress,
    on_heuristic=on_heuristic,
    on_done=on_done,
)
for _ in range(300):
    if results or errors:
        break
    time.sleep(1)
    if _ % 10 == 9:
        print(f'  ...waiting {_+1}s', flush=True)
if errors:
    print('ERROR:', errors['e'])
elif results:
    print('SUCCESS! VERDICT:', results['r'].get('heuristic', {}).get('basic_verdict'))
    print('LLM failed:', results['r'].get('llm_failed'))
else:
```

```
    print('TIMEOUT')
app._loop.call_soon_threadsafe(app._loop.stop)
```

## base.py

```
"""Абстрактне джерело повідомлень."""
from abc import ABC, abstractmethod
from typing import Any, AsyncIterator
from echo.pipeline.models import Message
class MessageSource(ABC):
    """Абстрактне джерело повідомлень."""
    @abstractmethod
    async def fetch(self, **kwargs: Any) -> AsyncIterator[Message]:
        """Асинхронний ітератор повідомлень із джерела."""
        ...
    @abstractmethod
    def estimate_count(self, **kwargs: Any) -> int | None:
        """Прогноз кількості повідомлень для прогрес-бара."""
        ...
```

## html_export.py

```
"""Парсер HTML-експортів Telegram Desktop (messages.html)."""
from __future__ import annotations
import logging
import re
from datetime import datetime, timedelta
from html.parser import HTMLParser
from pathlib import Path
from typing import Any, AsyncIterator
from echo.connectors.base import MessageSource
from echo.pipeline.models import Message
logger = logging.getLogger(__name__)
_DATE_RE = re.compile(r'(\d{2})\.(\d{2})\.(\d{4})\s+(\d{2}):(\d{2}):(\d{2})')
def _parse_ts(title: str) -> datetime | None:
    m = _DATE_RE.match(title)
    if not m:
        return None
    d, mo, y, h, mi, s = (int(x) for x in m.groups())
    try:
        return datetime(y, mo, d, h, mi, s)
    except ValueError:
        return None
class _MsgParser(HTMLParser):
    """Stateful SAX-style parser для messages.html Telegram Desktop."""
    def __init__(self) -> None:
        super().__init__()
        self.messages: list[dict] = []
        self._depth = 0
        self._msg_depth: int | None = None
        self._capture: str | None = None  # 'text' | 'from' | None
        self._cap_depth: int | None = None
        self._current: dict | None = None
        self._buf: list[str] = []
    def handle_starttag(self, tag: str, attrs: list) -> None:
        self._depth += 1
        if tag == "br":
            if self._capture:
                self._buf.append("\n")
            return
        if tag != "div":
            return
        attr_dict = dict(attrs)
        cls = attr_dict.get("class", "").split()
        # Початок нового повідомлення
        if "message" in cls and "default" in cls:
            self._current = {
                "forwarded": "forwarded" in cls,
                "text": "",
                "from": "",
                "timestamp": None,
```

```
            }
            self._msg_depth = self._depth
            return
        if self._current is None:
            return
        # Пересилане (forwarded wrapper всередині повідомлення)
        if "forwarded" in cls:
            self._current["forwarded"] = True
        # Текст повідомлення
        if "text" in cls and self._capture is None:
            self._capture = "text"
            self._cap_depth = self._depth
            self._buf = []
        # Ім'я автора
        elif "from_name" in cls and self._capture is None:
            self._capture = "from"
            self._cap_depth = self._depth
            self._buf = []
        # Дата/час (атрибут title="DD.MM.YYYY HH:MM:SS")
        elif "date" in cls and "details" in cls:
            title = attr_dict.get("title", "")
            if title and not self._current.get("timestamp"):
                self._current["timestamp"] = _parse_ts(title)
    def handle_endtag(self, tag: str) -> None:
        if tag == "div":
            # Закриваємо активний capture-блок
            if self._capture and self._depth == self._cap_depth:
                self._current[self._capture] = "".join(self._buf).strip()  # type: ignore[index]
                self._capture = None
                self._cap_depth = None
                self._buf = []
            # Закриваємо повідомлення
            if self._msg_depth and self._depth == self._msg_depth and self._current:
                if self._current.get("text"):
                    self.messages.append(self._current)
                self._current = None
                self._msg_depth = None
            self._depth -= 1
    def handle_data(self, data: str) -> None:
        if self._capture:
            self._buf.append(data)
def _parse_html_file(path: Path) -> list[dict]:
    parser = _MsgParser()
    parser.feed(path.read_text(encoding="utf-8", errors="replace"))
    return parser.messages
def find_html_files(path: Path) -> list[Path]:
    """messages.html, messages2.html, ... у теці або сам файл."""
    if path.is_file():
        return [path]
    files = sorted(path.glob("messages*.html"))
    if not files:
        files = sorted(path.rglob("messages*.html"))
    return files
class HtmlExportSource(MessageSource):
    """Джерело даних: HTML-експорт Telegram Desktop."""
    def __init__(self, path: str | Path) -> None:
        self.path = Path(path)
    async def fetch(self, mode: str = "channel", days: int | None = None, **kwargs: Any) -> AsyncIterator[Message]:
        files = find_html_files(self.path)
        if not files:
            raise FileNotFoundError(f"Файли messages*.html не знайдено у: {self.path}")
        skip_forwards = mode == "channel"
        # Збираємо всі повідомлення
        collected: list[Message] = []
        msg_id = 0
        for html_file in files:
            logger.info("HTML парсинг: %s", html_file.name)
            for raw in _parse_html_file(html_file):
                if skip_forwards and raw.get("forwarded"):
```

```
                continue
            text = raw.get("text", "").strip()
            if not text:
                continue
            msg_id += 1
            collected.append(Message(
                message_id=msg_id,
                text=text,
                timestamp=raw.get("timestamp"),
                author_id=None,
                author_name=raw.get("from") or None,
                source_kind="html_export",
            ))
        # Фільтрація по датах відносно найновішого повідомлення
        if days and collected:
            timestamps = [m.timestamp for m in collected if m.timestamp]
            if timestamps:
                max_ts = max(timestamps)
                cutoff = max_ts - timedelta(days=days)
                before = len(collected)
                collected = [m for m in collected if not m.timestamp or m.timestamp >= cutoff]
                logger.info("HTML days filter: %d → %d повідомлень (останні %d дн від %s)",
                            before, len(collected), days, max_ts.date())
        for msg in collected:
            yield msg
    def estimate_count(self, **kwargs: Any) -> int | None:
        files = find_html_files(self.path)
        total = 0
        for f in files:
            try:
                total += f.read_text(encoding="utf-8", errors="replace").count(
                    'class="message default'
                )
            except Exception:
                pass
        return total or None
```

## json_export.py

```
"""Парсер експортів Telegram Desktop (result.json)."""
from __future__ import annotations
import json
import logging
from datetime import datetime, timedelta
from pathlib import Path
from typing import Any, AsyncIterator
from echo.connectors.base import MessageSource
from echo.pipeline.models import Message
logger = logging.getLogger(__name__)
def _extract_text(item: dict[str, Any]) -> str:
    raw = item.get("text")
    if isinstance(raw, str):
        return raw
    if isinstance(raw, list):
        parts = []
        for sub in raw:
            if isinstance(sub, str):
                parts.append(sub)
            elif isinstance(sub, dict):
                parts.append(sub.get("text", ""))
        return "".join(parts)
    return ""
class JsonExportSource(MessageSource):
    """Парсер result.json із Telegram Desktop."""
    def __init__(self, path: str | Path) -> None:
        self.path = Path(path)
        self._messages: list[Message] = []
    async def fetch(self, days: int | None = None, **kwargs: Any) -> AsyncIterator[Message]:
        if not self.path.exists():
            raise FileNotFoundError(f"Файл не знайдено: {self.path}")
```

```
        data = json.loads(self.path.read_text(encoding="utf-8"))
        raw_items = data.get("messages", [])
        # Перший прохід: збираємо всі повідомлення з timestamp
        collected: list[Message] = []
        for item in raw_items:
            if not isinstance(item, dict):
                continue
            if item.get("type") == "service":
                continue
            text = _extract_text(item)
            if not text.strip():
                continue
            date_str = item.get("date")
            ts: datetime | None = None
            if date_str:
                try:
                    ts = datetime.fromisoformat(date_str.replace("Z", "+00:00"))
                    # Видаляємо tzinfo щоб порівняння було однорідним
                    if ts.tzinfo is not None:
                        ts = ts.replace(tzinfo=None)
                except ValueError:
                    pass
            from_id = item.get("from_id")
            collected.append(Message(
                message_id=item.get("id", 0),
                text=text,
                timestamp=ts,
                author_id=str(from_id) if from_id else None,
                author_name=item.get("from"),
                source_kind="json_export",
            ))
        # Фільтрація по датах відносно найновішого повідомлення в експорті
        if days and collected:
            timestamps = [m.timestamp for m in collected if m.timestamp]
            if timestamps:
                max_ts = max(timestamps)
                cutoff = max_ts - timedelta(days=days)
                collected = [m for m in collected if not m.timestamp or m.timestamp >= cutoff]
                logger.info("JSON days filter: %d → %d повідомлень (останні %d дн від %s)",
                            len(raw_items), len(collected), days, max_ts.date())
        for msg in collected:
            yield msg
    def estimate_count(self, **kwargs: Any) -> int | None:
        try:
            data = json.loads(self.path.read_text(encoding="utf-8"))
            return len(data.get("messages", []))
        except Exception:
            return None
```

## manual_text.py

```
"""Ручне введення тексту як джерело повідомлень."""
from __future__ import annotations
from datetime import datetime
from typing import AsyncIterator
from echo.connectors.base import MessageSource
from echo.pipeline.models import Message
class ManualTextSource(MessageSource):
    """Текст, введений користувачем вручну."""
    def __init__(self, text: str, delimiter: str = "\n") -> None:
        self.text = text
        self.delimiter = delimiter
        self._lines = [line.strip() for line in text.split(delimiter) if line.strip()]
    async def fetch(self, **kwargs: Any) -> AsyncIterator[Message]:
        for idx, line in enumerate(self._lines):
            yield Message(
                message_id=idx,
                text=line,
                timestamp=datetime.utcnow(),
                source_kind="manual",
```

```
        )
    def estimate_count(self, **kwargs: Any) -> int | None:
        return len(self._lines)
```

## telethon_source.py

```
"""Telethon-конектор для завантаження історії Telegram."""
from __future__ import annotations
import logging
import os
from datetime import datetime, timedelta
from typing import Any, AsyncIterator
from echo.config import SESSION_PATH, get_config
from echo.connectors.base import MessageSource
from echo.pipeline.models import Message
logger = logging.getLogger(__name__)
# Singleton клієнт — один на весь процес, щоб не плодити з'єднання
_client_singleton: Any = None
async def _get_client() -> Any:
    """Повертає вже підключений TelegramClient (singleton)."""
    global _client_singleton
    from telethon import TelegramClient
    cfg = get_config()
    api_id = cfg.telegram_api_id
    api_hash = cfg.telegram_api_hash
    if not api_id or not api_hash:
        raise RuntimeError("TELEGRAM_API_ID та TELEGRAM_API_HASH не налаштовано у .env")
    if _client_singleton is None:
        _client_singleton = TelegramClient(
            str(SESSION_PATH),
            api_id,
            api_hash,
            flood_sleep_threshold=60,
        )
    if not _client_singleton.is_connected():
        await _client_singleton.connect()
    if not await _client_singleton.is_user_authorized():
        raise RuntimeError(
            "Сесія Telegram не знайдена або застаріла.\n"
            "Перейдіть до Налаштувань → «Авторизуватись у терміналі»."
        )
    return _client_singleton
def session_exists() -> bool:
    """Перевіряє наявність валідного файлу сесії БЕЗ мережевого виклику."""
    path = SESSION_PATH
    # Telethon зберігає сесію як SQLite-файл
    if path.exists() and path.stat().st_size > 4096:
        return True
    # Telethon іноді додає .session до імені
    alt = SESSION_PATH.with_suffix(".session")
    return alt.exists() and alt.stat().st_size > 4096
class TelethonSource(MessageSource):
    """Джерело через Telethon. Використовує singleton-клієнт."""
    async def fetch(
        self,
        dialog: str | None = None,
        limit: int = 1000,
        days: int | None = None,
        user_id: int | None = None,
        mode: str = "channel",
        **kwargs: Any,
    ) -> AsyncIterator[Message]:
        client = await _get_client()

        if not dialog:
            raise ValueError("Не вказано ім'я каналу/групи для аналізу")
        try:
            entity = await client.get_entity(dialog)
        except Exception as exc:
            logger.warning("Не вдалося знайти діалог %s: %s", dialog, exc)
```

```
            raise
        min_date = datetime.utcnow() - timedelta(days=days) if days else None
        # Для каналу пропускаємо forwards — це чужий контент, не голос каналу
        skip_forwards = mode == "channel"
        async for msg in client.iter_messages(entity, limit=limit):
            if not msg.text:
                continue
            if skip_forwards and msg.fwd_from:
                continue
            if user_id is not None and msg.sender_id != user_id:
                continue
            if min_date and msg.date:
                msg_date = msg.date.replace(tzinfo=None)
                if msg_date < min_date:
                    break  # iter_messages іде від нових до старих — далі немає сенсу
            yield Message(
                message_id=msg.id,
                text=msg.text,
                timestamp=msg.date.replace(tzinfo=None) if msg.date else None,
                author_id=msg.sender_id,
                author_name=(
                    getattr(msg.sender, "username", None)
                    or getattr(msg.sender, "first_name", None)
                ),
                source_kind="telethon",
            )
    def estimate_count(self, **kwargs: Any) -> int | None:
        limit = kwargs.get("limit", 1000)
        days = kwargs.get("days")
        if days:
            return min(limit, days * 50)
        return limit
    async def get_dialogs(self) -> list[dict[str, Any]]:
        client = await _get_client()
        result = []
        async for dialog in client.iter_dialogs():
            result.append({
                "id": dialog.id,
                "name": dialog.name,
                "type": "channel" if dialog.is_channel else ("group" if dialog.is_group else "user"),
            })
        return result
    async def disconnect(self) -> None:
        """Не відключаємося — singleton живе весь час роботи застосунку."""
        pass
```

## channel_tab.py

```
"""Вкладка аналізу каналу."""
from __future__ import annotations
import asyncio
import logging
from typing import TYPE_CHECKING
import customtkinter as ctk
from echo.gui.widgets.progress_panel import ProgressPanel
from echo.gui.widgets.source_picker import SourcePicker
from echo.gui.widgets.verdict_panel import VerdictPanel
from echo.gui.visualizations import show_profile_charts
if TYPE_CHECKING:
    from echo.app import App
logger = logging.getLogger(__name__)
class ChannelTab(ctk.CTkFrame):
    def __init__(self, master: ctk.CTkBaseClass, app: "App", **kwargs):
        super().__init__(master, **kwargs)
        self.app = app
        self.source = SourcePicker(self)
        self.source.pack(fill="x", padx=8, pady=8)
        ctrl = ctk.CTkFrame(self)
        ctrl.pack(fill="x", padx=8, pady=4)
        ctk.CTkLabel(ctrl, text="Період (днів):").pack(side="left", padx=4)
```

```
        self.days_var = ctk.StringVar(value="30")
        ctk.CTkEntry(ctrl, textvariable=self.days_var, width=60).pack(side="left", padx=4)
        ctk.CTkButton(ctrl, text="Запустити аналіз", command=self._on_analyze).pack(side="left", padx=12)
        self.progress = ProgressPanel(self)
        self.progress.pack(fill="x", padx=8, pady=4)
        self.verdict = VerdictPanel(self)
        self.verdict.pack(fill="both", expand=True, padx=8, pady=4)
    def _on_analyze(self) -> None:
        cfg = self.source.get_config()
        days = int(self.days_var.get() or 30)
        self.progress.reset()
        self.app.run_analysis("channel", cfg, days, self._on_progress, on_done=self._on_done)
    def _on_progress(self, stage: str, value: float, text: str) -> None:
        self.after(0, lambda: self.progress.set_stage(stage, value, text))
    def _on_done(self, result: dict | None, error: str | None) -> None:
        self.after(0, lambda: self._show_result(result, error))
    def _show_result(self, result: dict | None, error: str | None) -> None:
        self.progress.set_stage("llm", 1.0, "Готово" if not error else f"Помилка: {error}")
        if error:
            self.verdict.clear()
            lbl = ctk.CTkLabel(self.verdict, text=f"Помилка: {error}", text_color="red")
            lbl.pack()
            return
        if result:
            verdict = result.get("verdict", {})
            self.verdict.show_channel_verdict(verdict)
            show_profile_charts(result.get("profile"), mode="channel")
```

## check_tab.py

```
"""Головна вкладка "Перевірка джерела"."""
from __future__ import annotations
import logging
from typing import TYPE_CHECKING, Any
import customtkinter as ctk
from echo.gui.widgets.details_panel import DetailsPanel
from echo.gui.widgets.input_field import InputField
from echo.gui.widgets.progress_panel import ProgressPanel
from echo.gui.widgets.verdict_card import VerdictCard
from echo.input_router import InputType, input_to_source_config
if TYPE_CHECKING:
    from echo.app import App
logger = logging.getLogger(__name__)
_MODE_OPTIONS = [
    ("Канал", "channel"),
    ("Груповий чат", "group"),
    ("Особистий чат", "dialog"),
]
_MODE_HINTS = {
    "channel": "",
    "group": (
        "⚠ Завантаження великої групи через Telegram займає від кількох годин до доби. "
        "Рекомендовано: Telegram Desktop → «Налаштування» → «Експортувати дані» → "
        "виберіть групу → вкажіть шлях до отриманого result.json."
    ),
    "dialog": (
        "Аналіз особистого чату — лише токсичність і тон. Вміст повідомлень не зберігається і не передається. "
        "Рекомендовано: вставте шлях до JSON-експорту (Telegram Desktop → «Експортувати дані» → особистий чат)."
    ),
}
class CheckTab(ctk.CTkFrame):
    """Вкладка "Перевірка" — єдиний вхід, авто-детект, двокроковий вердикт."""
    def __init__(self, master: ctk.CTkBaseClass, app: "App", **kwargs):
        super().__init__(master, **kwargs)
        self.app = app
        self._current_result: dict[str, Any] | None = None
        self._current_analysis_id: int | None = None
        self._heuristic: dict[str, Any] = {}
        self._heuristic_profile: dict[str, Any] | None = None
        # --- Поле введення ---
```

```
        self._input = InputField(self, on_change=self._on_input_change)
        self._input.pack(fill="x", padx=8, pady=(8, 4))
        # --- Рядок управління ---
        ctrl = ctk.CTkFrame(self, fg_color="transparent")
        ctrl.pack(fill="x", padx=8, pady=4)
        ctk.CTkLabel(ctrl, text="Режим:").pack(side="left", padx=(0, 4))
        self._mode_var = ctk.StringVar(value="channel")
        for label, val in _MODE_OPTIONS:
            ctk.CTkRadioButton(
                ctrl, text=label, variable=self._mode_var, value=val,
                command=self._on_mode_change,
            ).pack(side="left", padx=6)
        ctk.CTkLabel(ctrl, text="  Період (днів):").pack(side="left", padx=(12, 4))
        self._days_var = ctk.StringVar(value="30")
        ctk.CTkEntry(ctrl, textvariable=self._days_var, width=60).pack(side="left", padx=4)
        self._analyze_btn = ctk.CTkButton(ctrl, text="▶ Перевірити", command=self._on_analyze)
        self._analyze_btn.pack(side="left", padx=12)
        # --- Підказка режиму ---
        self._mode_hint = ctk.CTkLabel(
            self, text="", anchor="w", justify="left",
            font=ctk.CTkFont(size=11), text_color="#e67e22", wraplength=900,
        )
        self._mode_hint.pack(fill="x", padx=12, pady=(0, 2))
        # --- Прогрес ---
        self._progress = ProgressPanel(self)
        self._progress.pack(fill="x", padx=8, pady=4)
        # --- Результат ---
        result_frame = ctk.CTkScrollableFrame(self)
        result_frame.pack(fill="both", expand=True, padx=8, pady=4)
        self._verdict_card = VerdictCard(
            result_frame,
            on_details=self._on_show_details,
            on_blacklist=self._on_blacklist,
        )
        self._verdict_card.pack(fill="x")
        self._details = DetailsPanel(result_frame)
        self._details.pack(fill="both", expand=True)
    def _on_mode_change(self) -> None:
        hint = _MODE_HINTS.get(self._mode_var.get(), "")
        self._mode_hint.configure(text=hint)
    def _on_input_change(self, text: str, itype: InputType) -> None:
        # Режим НЕ перемикаємо автоматично — JSON/HTML може бути канал, діалог або група.
        # Користувач сам вибирає режим через radio-кнопки.
        pass
    def _on_analyze(self) -> None:
        raw = self._input.get()
        if not raw:
            return
        mode = self._mode_var.get()
        try:
            days = int(self._days_var.get() or 30)
        except ValueError:
            days = 30
        source_cfg = input_to_source_config(raw, mode=mode, days=days)
        self._progress.reset()
        self._analyze_btn.configure(state="disabled")
        self._verdict_card.clear()
        self._details.reset()
        self.app.run_analysis(
            mode=mode,
            source_cfg=source_cfg,
            days=days,
            on_progress=self._on_progress,
            on_heuristic=self._on_heuristic,
            on_done=self._on_done,
        )
    def _on_progress(self, stage: str, value: float, text: str) -> None:
        self.after(0, lambda: self._progress.set_stage(stage, value, text))
    def _on_heuristic(self, heuristic: dict[str, Any], profile: dict[str, Any] | None = None) -> None:
```

```
        """Показуємо базовий світлофор одразу після локальних моделей."""
        self._heuristic = heuristic
        self._heuristic_profile = profile
        self.after(0, lambda: self._verdict_card.show_heuristic(heuristic, profile))
    def _on_done(self, result: dict[str, Any] | None, error: str | None) -> None:
        self.after(0, lambda: self._show_result(result, error))
    def _show_result(self, result: dict[str, Any] | None, error: str | None) -> None:
        self._analyze_btn.configure(state="normal")
        if error:
            self._progress.set_stage("llm", 1.0, f"Помилка: {error[:80]}")
            return
        if not result:
            return
        self._current_result = result
        heuristic = result.get("heuristic", {})
        llm_verdict = result.get("verdict", {})
        profile = result.get("profile", {})
        if llm_verdict:
            self._verdict_card.show_full(heuristic, llm_verdict, profile)
        else:
            self._verdict_card.show_heuristic(heuristic, profile)
        # Деталі показуємо завжди — навіть якщо LLM не відповів, є профіль і графіки
        self._details.setup(llm_verdict or {}, profile)
        if result.get("llm_failed"):
            self._progress.set_stage(
                "llm", 1.0, "LLM недоступний. Базовий вердикт залишається в силі."
            )
    def _on_show_details(self) -> None:
        self._details.toggle()
    def _on_blacklist(self) -> None:
        if not self._current_result:
            return
        try:
            source_id = self._current_result.get("profile", {}).get("source_identifier", "")
            if source_id:
                # Знаходимо останній аналіз за source_identifier
                analyses = self.app.repo.list_analyses(limit=1, search=source_id)
                if analyses:
                    self.app.repo.set_blacklisted(analyses[0]["id"], True)
                    logger.info("Додано в чорний список: %s", source_id)
        except Exception as exc:
            logger.warning("Помилка чорного списку: %s", exc)
```

## dialog_tab.py

```
"""Вкладка аналізу діалогу."""
from __future__ import annotations
import logging
from typing import TYPE_CHECKING
import customtkinter as ctk
from echo.gui.widgets.progress_panel import ProgressPanel
from echo.gui.widgets.source_picker import SourcePicker
from echo.gui.widgets.verdict_panel import VerdictPanel
from echo.gui.visualizations import show_profile_charts
if TYPE_CHECKING:
    from echo.app import App
logger = logging.getLogger(__name__)
class DialogTab(ctk.CTkFrame):
    def __init__(self, master: ctk.CTkBaseClass, app: "App", **kwargs):
        super().__init__(master, **kwargs)
        self.app = app
        self.source = SourcePicker(self)
        self.source.pack(fill="x", padx=8, pady=8)
        ctrl = ctk.CTkFrame(self)
        ctrl.pack(fill="x", padx=8, pady=4)
        ctk.CTkLabel(ctrl, text="user_id фільтр (опційно):").pack(side="left", padx=4)
        self.user_var = ctk.StringVar()
        ctk.CTkEntry(ctrl, textvariable=self.user_var, width=120).pack(side="left", padx=4)
        ctk.CTkLabel(ctrl, text="Днів:").pack(side="left", padx=4)
        self.days_var = ctk.StringVar(value="30")
```

```
        ctk.CTkEntry(ctrl, textvariable=self.days_var, width=60).pack(side="left", padx=4)
        ctk.CTkButton(ctrl, text="Запустити аналіз", command=self._on_analyze).pack(side="left", padx=12)
        self.progress = ProgressPanel(self)
        self.progress.pack(fill="x", padx=8, pady=4)
        self.verdict = VerdictPanel(self)
        self.verdict.pack(fill="both", expand=True, padx=8, pady=4)
    def _on_analyze(self) -> None:
        cfg = self.source.get_config()
        days = int(self.days_var.get() or 30)
        user_id = self.user_var.get().strip()
        if user_id:
            cfg["user_id"] = int(user_id)
        self.progress.reset()
        self.app.run_analysis("dialog", cfg, days, self._on_progress, on_done=self._on_done)
    def _on_progress(self, stage: str, value: float, text: str) -> None:
        self.after(0, lambda: self.progress.set_stage(stage, value, text))
    def _on_done(self, result: dict | None, error: str | None) -> None:
        self.after(0, lambda: self._show_result(result, error))
    def _show_result(self, result: dict | None, error: str | None) -> None:
        self.progress.set_stage("llm", 1.0, "Готово" if not error else f"Помилка: {error}")
        if error:
            self.verdict.clear()
            lbl = ctk.CTkLabel(self.verdict, text=f"Помилка: {error}", text_color="red")
            lbl.pack()
            return
        if result:
            verdict = result.get("verdict", {})
            self.verdict.show_dialog_verdict(verdict)
            show_profile_charts(result.get("profile"), mode="dialog")
```

## group_tab.py

```
"""Вкладка аналізу групового чату."""
from __future__ import annotations
import logging
from typing import TYPE_CHECKING
import customtkinter as ctk
from echo.gui.widgets.progress_panel import ProgressPanel
from echo.gui.widgets.source_picker import SourcePicker
from echo.gui.widgets.verdict_panel import VerdictPanel
from echo.gui.visualizations import show_profile_charts
if TYPE_CHECKING:
    from echo.app import App
logger = logging.getLogger(__name__)
class GroupTab(ctk.CTkFrame):
    def __init__(self, master: ctk.CTkBaseClass, app: "App", **kwargs):
        super().__init__(master, **kwargs)
        self.app = app
        self.source = SourcePicker(self)
        self.source.pack(fill="x", padx=8, pady=8)
        ctrl = ctk.CTkFrame(self)
        ctrl.pack(fill="x", padx=8, pady=4)
        ctk.CTkLabel(ctrl, text="Період (днів):").pack(side="left", padx=4)
        self.days_var = ctk.StringVar(value="30")
        ctk.CTkEntry(ctrl, textvariable=self.days_var, width=60).pack(side="left", padx=4)
        ctk.CTkButton(ctrl, text="Запустити аналіз", command=self._on_analyze).pack(side="left", padx=12)
        self.progress = ProgressPanel(self)
        self.progress.pack(fill="x", padx=8, pady=4)
        self.verdict = VerdictPanel(self)
        self.verdict.pack(fill="both", expand=True, padx=8, pady=4)
    def _on_analyze(self) -> None:
        cfg = self.source.get_config()
        days = int(self.days_var.get() or 30)
        self.progress.reset()
        self.app.run_analysis("group", cfg, days, self._on_progress, on_done=self._on_done)
    def _on_progress(self, stage: str, value: float, text: str) -> None:
        self.after(0, lambda: self.progress.set_stage(stage, value, text))
    def _on_done(self, result: dict | None, error: str | None) -> None:
        self.after(0, lambda: self._show_result(result, error))
    def _show_result(self, result: dict | None, error: str | None) -> None:
```

```
        self.progress.set_stage("llm", 1.0, "Готово" if not error else f"Помилка: {error}")
        if error:
            self.verdict.clear()
            lbl = ctk.CTkLabel(self.verdict, text=f"Помилка: {error}", text_color="red")
            lbl.pack()
            return
        if result:
            verdict = result.get("verdict", {})
            self.verdict.show_group_verdict(verdict)
            show_profile_charts(result.get("profile"), mode="group")
```

## help_tab.py

```
"""Вкладка «Довідка» — методологія, моделі, математика."""
from __future__ import annotations
import customtkinter as ctk
_SECTIONS = [
    (
        "Як це працює",
        """\
ЕСНО аналізує повідомлення Telegram-каналу у два кроки:
 1. Локальні ML-моделі (працюють на вашому комп'ютері, без інтернету):
    • Класифікація кожного повідомлення за темою (7 категорій)
    • Оцінка тональності (від «дуже негативної» до «дуже позитивної»)
    • Оцінка токсичності (0.0 — нейтральне, 1.0 — токсичне)
    → Результат: базовий вердикт-світлофор (з'являється одразу)
 2. LLM-аналіз (хмарна мовна модель):
    • Отримує агрегований профіль + найбільш показові цитати
    • Шукає маніпулятивні патерни, яких статистика не вловлює:
      клікбейт, відсутність джерел, апеляція до страху, однобічність
    → Результат: повний вердикт із резюме, поясненням і рекомендацією\
""",
    ),
    (
        "Шкала вердиктів",
        """\
Шкала вимірює НАВАНТАЖЕННЯ НА ЧИТАЧА — не «правильність» позиції автора.
Виважений але незручний контент = зелений. Зручний але маніпулятивний = червоний.
 🟢 ЗЕЛЕНИЙ — безпечно для регулярного читання
    Низька токсичність, різноманітні теми, аргументована подача.
 🟡 ЖОВТИЙ — можна, але нормуйте
    Виражена позиція, але аргументована. Стежте за балансом джерел.
 🟠 ПОМАРАНЧЕВИЙ — лише за необхідності
    Високий емоційний градус, однобічність, ознаки відбору фактів.
    Можна читати як одне з джерел — але не як основне.
 🔴 ЧЕРВОНИЙ — не рекомендовано для регулярного читання
    Висока токсичність або клішована подача без рефлексії.
    Маніпулятивні прийоми: страх, ненависть, стереотипні ярлики.\
""",
    ),
    (
        "Базовий вердикт: математика",
        """\
Базовий вердикт розраховується локально без інтернету за формулою:
 emotional_load = w₁ × toxicity
                + w₂ × aggression
                + w₃ × concentration
                + w₄ × negative_skew
 де:
   toxicity      — середній бал токсичності по всіх повідомленнях (0–1)
   aggression    — частка «дуже негативних» + «дуже позитивних» (0–1)
   concentration — 1 − нормована ентропія розподілу тем (0=різноманітно, 1=монотема)
   negative_skew — max(0, −avg_sentiment), де avg_sentiment ∈ [−1; +1]
 Ваги за замовчуванням (config.toml → [heuristic]):
   w₁ = 0.35  (токсичність)
   w₂ = 0.25  (агресивність)
   w₃ = 0.20  (тематична концентрація)
   w₄ = 0.20  (негативне зміщення)
 Порогові значення:
```

```
  emotional_load < 0.30 → 🟢 Зелений
  0.30 ≤ load < 0.55    → 🟡 Жовтий
  0.55 ≤ load < 0.75    → 🟠 Помаранчевий
  load ≥ 0.75           → 🔴 Червоний
 Всі параметри можна змінити у config.toml (без перекомпіляції).\
""",
  ),
  (
    "Маркери стилю (StyleMarkers)",
    """\
Поряд зі статистикою тональності та токсичності система обчислює
лінгвістичні маркери, які передаються LLM для виявлення маніпуляцій:
 exclamation_rate    — частка повідомлень з ≥1 знаком «!»
                > 0.40 → емоційне нагнітання без аргументів
 caps_words_rate     — частка повідомлень з ≥1 словом ВЕЛИКИМИ ЛІТЕРАМИ (3+ символи)
                > 0.15 → агресивне виділення, «крик»
 has_url_rate        — частка повідомлень із посиланнями (потенційні джерела)
                < 0.08 → майже немає джерел → маніпулятивна безпідставність
 short_message_rate  — частка повідомлень < 80 символів
                > 0.40 → клікбейт-стиль, поверхнева подача
 question_rate       — частка повідомлень із «?» (риторичні запитання)
 avg_length_chars    — середня довжина повідомлення в символах
                < 100 → дуже короткі, без деталей і контексту
 Комбінація: exclamation_rate > 0.40 + has_url_rate < 0.08 + war/politics →
 типовий профіль агітаційного каналу, мінімальний вердикт — помаранчевий.\
""",
  ),
  (
    "Локальні ML-моделі",
    """\
Всі три моделі завантажуються з HuggingFace Hub при першому запуску
і кешуються локально (~2 ГБ разом). Інтернет після цього не потрібен.
┌──────────────┬──────────────────────────┬──────────┐
│ Задача       │ Модель                   │ Розмір   │
├──────────────┼──────────────────────────┼──────────┤
│ Класифікація │ MoritzLaurer/            │ ~280 МБ  │
│ тем (zero-shot) │ mDeBERTa-v3-base-mnli-xnli │       │
│              │ 7 категорій: війна, політика, │       │
│              │ економіка, побут, ІТ, культура, │      │
│              │ інше                     │          │
├──────────────┼──────────────────────────┼──────────┤
│ Тональність  │ nlptown/                 │ ~670 МБ  │
│ (5 рівнів)   │ bert-base-multilingual-  │          │
│              │ uncased-sentiment        │          │
│              │ Very Negative → Very Positive │     │
├──────────────┼──────────────────────────┼──────────┤
│ Токсичність  │ unitary/                 │ ~1.1 ГБ  │
│ (0.0–1.0)    │ multilingual-toxic-xlm-roberta │    │
│              │ Підтримує 100+ мов, вкл. укр. │     │
└──────────────┴──────────────────────────┴──────────┘
 GPU (CUDA): якщо є відеокарта NVIDIA — аналіз у 5–10 разів швидший.
 Налаштування: config.toml → [models] → device = "cuda"\
""",
  ),
  (
    "LLM-каскад",
    """\
LLM-аналіз використовує каскад хмарних провайдерів.
Якщо перший недоступний (вичерпано ліміт, помилка мережі) —
застосунок автоматично переходить до наступного.
 Порядок каскаду:
 1. Groq        llama-3.3-70b-versatile          (найшвидший)
 2. Groq        meta-llama/llama-4-scout-17b-16e  (Llama 4)
 3. Cerebras    qwen-3-235b-a22b-instruct-2507    (235B параметрів)
 4. Google      gemini-2.5-flash                  (висока якість)
 5. Google      gemini-3.1-flash-lite             (швидкий)
 6. GitHub      gpt-4o
 7. OpenRouter  openai/gpt-oss-120b:free
 Що передається LLM:
```

```
  • Агрегований профіль (статистика, не вихідні повідомлення)
  • До 30 репрезентативних цитат (скорочених до 500 символів)
  • Базовий вердикт від локальних моделей
  Якщо всі провайдери недоступні — відображається лише базовий вердикт.
  Кеш: результат LLM зберігається на 7 днів — повторний аналіз
  того самого каналу за той самий період не витрачає ліміти.\
"""
    ),
    (
        "Що НЕ робить ECHO",
        """\
  • Не зберігає і не передає вихідні повідомлення — лише агрегована статистика
  • Не оцінює «правильність» або «правоту» позиції автора
  • Не є юридичним або медійним висновком
  • Не виявляє фейки (немає fact-checking бази)
  • Не перевіряє особистість авторів
  Обмеження локальних моделей:
  • Токсичність і тональність не вловлюють тонку іронію
  • Класифікація тем — статистична, можливі помилки для коротких текстів
  • Моделі не навчались спеціально на українських пропагандистських патернах
  LLM компенсує більшість цих обмежень через аналіз тексту цитат,
  але також не є безпомилковим.\
"""
    ),
    (
        "Як експортувати чат з Telegram Desktop",
        """\
Аналіз групових чатів і особистих переписок — через Telegram Desktop Export.
ECHO приймає обидва формати: HTML (зручний) і JSON (машиночитаємий).
  КРОКИ (однакові для групи та особистого чату):
  ┌──────────────────────────────────────────────────────────────┐
  │ 1. Відкрийте Telegram Desktop і перейдіть у потрібний чат/групу  │
  │ 2. Три крапки (⋮) → «Налаштування» → «Експортувати дані чату»   │
  │    (або: «Управление группой» → «Экспортировать данные»)        │
  │ 3. Знімніть галочки для фото, відео, файлів — залиште «Messages»│
  │ 4. Виберіть формат:                                          │
  │      • HTML  — перегляд у браузері, зручно для людини          │
  │      • JSON  — машиночитаємий, менший розмір файлу             │
  │    ECHO підтримує обидва — вибирайте будь-який.                │
  │ 5. Натисніть «Експортувати» → вкажіть папку збереження         │
  │ 6. Після завершення у папці з'явиться:                       │
  │      • для HTML: messages.html (та можливо messages2.html...)  │
  │      • для JSON: result.json                                 │
  │ 7. Вставте в ECHO:                                           │
  │      • шлях до папки (ECHO сам знайде файли)                   │
  │      • або шлях до конкретного файлу (messages.html/result.json) │
  │ 8. Виберіть режим «Груповий чат» або «Особистий чат» — Перевірка│
  │                                                              │
  │ ⚠ Для великих груп (10 000+ повідомлень) завантаження          │
  │   може тривати від кількох годин до доби.                    │
  └──────────────────────────────────────────────────────────────┘
  TELEGRAM API (лише для каналів):
  Можна вставити @username або посилання t.me/... напряму.
  ECHO підключиться через Telegram API і завантажить останні N днів.
  Для груп і особистих чатів цей метод не підходить — лише експорт.\
"""
    ),
    (
        "Конфігурація (config.toml)",
        """\
  Файл конфігурації: %APPDATA%\\echo\\config.toml  (Windows)
                     ~/.echo/config.toml          (Linux/macOS)
  Ключові параметри:
  [models]
  device     = "auto"   # "cpu", "cuda" або "auto" (автовизначення GPU)
  batch_size  = 16      # повідомлень за раз (зменшіть при нестачі RAM)
  max_classify = 150    # максимум повідомлень для ML-класифікації
  [heuristic]
  weight_toxicity    = 0.35   # вага токсичності у формулі
```

```
  weight_aggression   = 0.25   # вага агресивності
  weight_concentration = 0.20  # вага тематичної концентрації
  weight_negative_skew = 0.20  # вага негативного зміщення
  threshold_yellow    = 0.30
  threshold_orange    = 0.55
  threshold_red       = 0.75
  [llm]
  cache_ttl_days      = 7     # скільки днів кешувати результат LLM
  temperature         = 0.3
  max_output_tokens   = 2048\
"""
    ),
]
class HelpTab(ctk.CTkFrame):
    """Вкладка довідки — методологія, математика, моделі."""
    def __init__(self, master: ctk.CTkBaseClass, **kwargs):
        super().__init__(master, **kwargs)
        scroll = ctk.CTkScrollableFrame(self)
        scroll.pack(fill="both", expand=True, padx=8, pady=8)
        ctk.CTkLabel(
            scroll,
            text="Довідка — як працює ECHO",
            font=ctk.CTkFont(size=18, weight="bold"),
            anchor="w",
        ).pack(anchor="w", padx=4, pady=(4, 12))
        for title, body in _SECTIONS:
            self._add_section(scroll, title, body)
    def _add_section(self, parent, title: str, body: str) -> None:
        frame = ctk.CTkFrame(parent, corner_radius=8)
        frame.pack(fill="x", pady=6)
        ctk.CTkLabel(
            frame,
            text=title,
            font=ctk.CTkFont(size=13, weight="bold"),
            anchor="w",
        ).pack(anchor="w", padx=12, pady=(10, 4))
        ctk.CTkLabel(
            frame,
            text=body,
            font=ctk.CTkFont(size=12, family="Courier New"),
            anchor="w",
            justify="left",
            wraplength=900,
        ).pack(anchor="w", padx=12, pady=(0, 10))
```

## history_tab.py

```
"""Вкладка "Історія перевірок"."""
from __future__ import annotations
import json
import logging
from datetime import datetime
from tkinter import messagebox
from typing import TYPE_CHECKING, Any
import customtkinter as ctk
if TYPE_CHECKING:
    from echo.app import App
logger = logging.getLogger(__name__)
_VERDICT_ICONS = {
    "green": "🟢",
    "yellow": "🟡",
    "orange": "🟠",
    "red": "🔴",
}
_VERDICT_LABELS = {
    "green": "Безпечно",
    "yellow": "Помірний",
    "orange": "Навантажений",
    "red": "Не рекомендовано",
```

```
}
_FILTER_OPTIONS = ["Усі", "green", "yellow", "orange", "red", "Чорний список"]
class HistoryTab(ctk.CTkFrame):
    """Список усіх перевірок з пошуком та фільтрацією."""
    def __init__(self, master: ctk.CTkBaseClass, app: "App", **kwargs):
        super().__init__(master, **kwargs)
        self.app = app
        self._analyses: list[dict[str, Any]] = []
        # --- Панель управління ---
        ctrl = ctk.CTkFrame(self, fg_color="transparent")
        ctrl.pack(fill="x", padx=8, pady=8)
        self._search_var = ctk.StringVar()
        self._search_entry = ctk.CTkEntry(
            ctrl,
            textvariable=self._search_var,
            placeholder_text="Пошук за назвою...",
            width=200,
        )
        self._search_entry.pack(side="left", padx=(0, 4))
        self._search_entry.bind("<KeyRelease>", lambda _: self._refresh())
        ctk.CTkLabel(ctrl, text="Фільтр:").pack(side="left", padx=(8, 4))
        self._filter_var = ctk.StringVar(value="Усі")
        self._filter_menu = ctk.CTkOptionMenu(
            ctrl,
            values=_FILTER_OPTIONS,
            variable=self._filter_var,
            command=lambda _: self._refresh(),
            width=150,
        )
        self._filter_menu.pack(side="left", padx=4)
        ctk.CTkButton(ctrl, text="Оновити", width=90, command=self._refresh).pack(
            side="left", padx=12
        )
        # --- Список ---
        self._scroll = ctk.CTkScrollableFrame(self)
        self._scroll.pack(fill="both", expand=True, padx=8, pady=4)
        # --- Нижня панель ---
        bottom = ctk.CTkFrame(self, fg_color="transparent")
        bottom.pack(fill="x", padx=8, pady=4)
        ctk.CTkButton(
            bottom,
            text="Очистити старі (>90 днів)",
            fg_color="transparent",
            border_width=1,
            command=self._clear_old,
        ).pack(side="left", padx=4)
        self._refresh()
    def _refresh(self) -> None:
        search = self._search_var.get().strip()
        flt = self._filter_var.get()
        filter_verdict = "all"
        blacklist_only = False
        if flt in ("green", "yellow", "orange", "red"):
            filter_verdict = flt
        elif flt == "Чорний список":
            blacklist_only = True
        analyses = self.app.repo.list_analyses(
            limit=100,
            search=search,
            filter_verdict=filter_verdict,
        )
        if blacklist_only:
            analyses = [a for a in analyses if a.get("is_blacklisted")]
        self._analyses = analyses
        self._render_list(analyses)
    def _render_list(self, analyses: list[dict[str, Any]]) -> None:
        for w in self._scroll.winfo_children():
            w.destroy()
        if not analyses:
```

```
            ctk.CTkLabel(self._scroll, text="Немає записів", text_color="gray").pack(pady=20)
            return
        # Групуємо за датою
        current_date: str | None = None
        for a in analyses:
            try:
                dt = datetime.fromisoformat(str(a.get("created_at", "")))
                date_str = dt.strftime("%d.%m.%Y")
            except Exception:
                date_str = "Невідома дата"
            if date_str != current_date:
                current_date = date_str
                ctk.CTkLabel(
                    self._scroll,
                    text=f"🗓 {date_str}",
                    font=ctk.CTkFont(weight="bold"),
                    anchor="w",
                ).pack(fill="x", padx=4, pady=(8, 2))
            self._render_item(a, dt if "dt" in dir() else None)
    def _render_item(self, a: dict[str, Any], dt=None) -> None:
        verdict = a.get("basic_verdict", "orange")
        llm_verdict = a.get("llm_verdict") or verdict
        icon = _VERDICT_ICONS.get(llm_verdict, "○")
        label = _VERDICT_LABELS.get(llm_verdict, "")
        source = a.get("source_identifier", "?")
        is_blacklisted = bool(a.get("is_blacklisted"))
        try:
            if dt:
                time_str = dt.strftime("%H:%M")
            else:
                time_str = ""
        except Exception:
            time_str = ""
        row = ctk.CTkFrame(self._scroll)
        row.pack(fill="x", padx=4, pady=2)
        # Іконка та назва
        ctk.CTkLabel(row, text=icon, width=28).pack(side="left", padx=(4, 0))
        source_lbl = ctk.CTkLabel(row, text=source, anchor="w")
        source_lbl.pack(side="left", fill="x", expand=True, padx=4)
        # Підпис вердикту
        ctk.CTkLabel(row, text=label, width=130, anchor="w", text_color="gray").pack(side="left")
        # Час
        if time_str:
            ctk.CTkLabel(row, text=time_str, width=45, text_color="gray").pack(side="left")
        # Чорний список
        if is_blacklisted:
            ctk.CTkLabel(row, text="●", width=24).pack(side="left")
        # Кнопка toggle чорного списку
        bl_text = "Зняти" if is_blacklisted else "●"
        ctk.CTkButton(
            row,
            text=bl_text,
            width=50,
            fg_color="transparent",
            border_width=1,
            command=lambda aid=a["id"], cur=is_blacklisted: self._toggle_blacklist(aid, cur),
        ).pack(side="left", padx=(4, 2))
    def _toggle_blacklist(self, analysis_id: int, currently_blacklisted: bool) -> None:
        self.app.repo.set_blacklisted(analysis_id, not currently_blacklisted)
        self._refresh()
    def _clear_old(self) -> None:
        ok = messagebox.askokcancel(
            "Очистити старі записи",
            "Видалити всі перевірки старші 90 днів?",
        )
        if ok:
            cnt = self.app.repo.delete_old_analyses(older_than_days=90)
            messagebox.showinfo("Готово", f"Видалено {cnt} записів")
            self._refresh()
```