ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Вдосконалення методу обробки неструктурованих даних на основі трансформерів»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Вадим БОГУТА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Вадим БОГУТА

Керівник: Юрій ЗАДОНЦЕВ
канд. техн. наук

Рецензент:
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Богуті Вадиму Сергійовичу

1. Тема кваліфікаційної роботи: «Вдосконалення методу обробки неструктурованих даних на основі трансформерів»
керівник кваліфікаційної роботи Юрій ЗАДОНЦЕВ, канд. техн. Наук,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.
2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, праці з архітектури трансформерів, матеріали з вилучення інформації та розпізнавання іменованих сутностей, методики обробки природної мови.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд існуючих підходів та методів добування інформації та структурної розмітки неструктурованих даних.
2. Аналіз архітектури трансформерів та їх застосування для задач обробки природної мови та вилучення сутностей/відносин.
3. Розробка та імплементація алгоритму попередньої обробки неструктурованих даних.

5. Перелік ілюстративного матеріалу: *презентація*

1. Схема методу обробки неструктурованих даних.
2. Математична модель перетворення даних.
3. Діаграма послідовності дій.
4. Алгоритм формування та валідації промта.
5. Практичний результат (Екранні форми).
6. Демонстрація екрану.
7. Порівняльний аналіз.

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-05.11.25	
2	Вивчення архітектур трансформерів (BERT, GPT, T5) та їх застосування в NLP	06.11-12.11.25	
3	Дослідження методів вилучення інформації та розпізнавання сутностей	13.11-19.11.25	
4	Аналіз особливостей впливу моделей трансформерів на якість структурної розмітки неструктурованих даних	20.11-26.11.25	
5	Розробка вдосконаленого методу	27.11-03.12.25	
6	Імплементація, навчання моделі та проведення експериментальних досліджень	04.12-08.12.25	
7	Оформлення роботи: вступ, висновки, реферат	09.12-15.12.25	
8	Розробка демонстраційних матеріалів	16.12-19.12.25	

Здобувач вищої освіти _____ Вадим БОГУТА
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 91 стор., 6 табл., 18 рис., 30 джерел.

Мета роботи – підвищення якості перетворення неструктурованих даних на людиночитану форму шляхом вдосконалення методу генерації текстових звітів на основі трансформерів.

Об'єкт дослідження – процес перетворення неструктурованих даних у текстові звіти.

Предмет дослідження – метод перетворення неструктурованих даних на людиночитану форму на основі моделей типу трансформер.

У роботі використано методи глибокого навчання, математичного моделювання, статистичного аналізу, а також принципи об'єктно-орієнтованого програмування. Основний акцент зроблено на використанні архітектур трансформерів та великих мовних моделей для задачі узагальнення і структуризації неструктурованих даних.

Проведено аналіз сучасних підходів до формування звітів з неструктурованих джерел, включаючи класичні rule-based системи, шаблонну генерацію та сучасні нейронні методи. Розглянуто особливості використання моделей трансформерів для задач екстракції сутностей, контекстного узагальнення та генерації людиночитаних звітів.

Розроблено вдосконалений метод перетворення неструктурованих даних, який включає попередню нормалізацію і структуризацію вхідних даних, побудову промптів (контекстів) для LLM та постобробку згенерованих відповідей з метою забезпечення точності та повноти відображення вихідних даних (наприклад, звітів за квотами ресурсів чи фінансових показників).

Реалізовано програмну систему, що приймає на вхід неструктуровані текстові дані (логи, вільні коментарі, raw-звіти, напівструктуровані файли) та генерує формалізовані текстові звіти із заданою структурою. Система інтегрує

модуль підготовки даних, модуль взаємодії з моделлю-трансформером та модуль валідації вихідних текстів.

Проведено експериментальні дослідження на тестових наборах даних, які містять приклади реальних неструктурованих записів і референсні (еталонні) звіти. Оцінено якість роботи системи за автоматичними метриками (BLEU, ROUGE, BERTScore) та за результатами експертної оцінки зрозумілості, повноти та коректності інформації.

Результати дослідження підтверджують, що використання вдосконаленого методу на основі трансформерів дозволяє підвищити якість генерованих звітів у порівнянні з базовими нейронними та rule-based підходами, а також зменшити час підготовки аналітичних матеріалів з неструктурованих джерел.

КЛЮЧОВІ СЛОВА: ТРАНСФОРМЕРИ, НЕСТРУКТУРОВАНІ ДАНІ, ОБРОБКА ПРИРОДНОЇ МОВИ, ВИЛУЧЕННЯ ІНФОРМАЦІЇ, РОЗПІЗНАВАННЯ ІМЕНОВАНИХ СУТНОСТЕЙ, ГЛИБОКЕ НАВЧАННЯ, СТРУКТУРНА РОЗМІТКА.

ABSTRACT

Text part of the master's qualification work: 91 pages, 18 pictures, 6 table, 30 sources.

The purpose of the work is to improve the quality of transforming unstructured data into human-readable form by enhancing a transformer-based method for generating textual reports.

Object of research – is the process of transforming unstructured data into textual reports.

Subject of research – is a transformer-based method for converting unstructured data into human-readable text.

Summary of the work:

The thesis investigates state-of-the-art approaches to generating reports from unstructured sources, including classical rule-based and template-based systems as well as modern neural methods. Special attention is paid to transformer architectures and large language models (LLMs) used for information extraction, summarization and controlled text generation.

An improved method is proposed that combines preprocessing and normalization of raw input data, construction of prompts for LLMs, and post-processing of model outputs to ensure that all important facts and numerical values (e.g. quotas, financial indicators, resource usage) are preserved and clearly presented.

A software system is implemented which accepts unstructured textual data (logs, free-form comments, raw reports, semi-structured files) and generates structured human-readable reports with a predefined layout. The system integrates a data preparation module, a transformer-based generation module and an output validation module.

Experimental evaluation is carried out on test datasets containing real-world unstructured records and corresponding reference reports. The quality of generated texts is assessed using automatic metrics (BLEU, ROUGE, BERTScore) and expert human evaluation of fluency, correctness and completeness.

The results show that the proposed transformer-based method improves report quality compared to baseline neural and rule-based approaches, and significantly reduces the time needed to prepare analytical reports from unstructured data sources

KEYWORDS: TRANSFORMERS, UNSTRUCTURED DATA, NATURAL LANGUAGE PROCESSING, INFORMATION EXTRACTION, NAMED ENTITY RECOGNITION, DEEP LEARNING, STRUCTURED MARKING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, програмний інтерфейс прикладного програмування.

BLEU (Bilingual Evaluation Understudy) – метрика автоматичної оцінки якості машинно згенерованого тексту шляхом порівняння з еталонним текстом.

BERTScore – метрика оцінки подібності текстів на основі контекстних векторних представлень (ембеддингів).

GPT (Generative Pre-trained Transformer) – сімейство генеративних попередньо натренованих мовних моделей на основі трансформерів.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, який відноситься до напівструктурованих форматів.

LLM (Large Language Model) – велика мовна модель, як правило, нейронна модель з мільярдами параметрів, здатна виконувати широкий спектр задач обробки природної мови.

NLP (Natural Language Processing) – галузь штучного інтелекту, що вивчає методи автоматичної обробки природної мови.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) – набір метрик для оцінки якості автоматичного резюмування та генерації тексту.

CSV (Comma-Separated Values) – текстовий формат зберігання табличних даних, де значення розділяються комами чи іншими роздільниками.

XML (eXtensible Markup Language) – розширювана мова розмітки для подання напівструктурованих даних.

Промпт (prompt) – текстова інструкція/запит до мовної моделі, що описує задачу, формат та контекст відповіді.

Квота – встановлений ліміт (обсяг) використання ресурсу, показник, який не повинен бути перевищений.

Трансформер – архітектура нейронних мереж, що використовує механізм self-attention для обробки послідовностей даних.

Data-to-text – клас задач, у яких за структурованими даними (таблицями, фактами) автоматично генерується текстовий опис.

GPU (Graphics Processing Unit) – Графічний процесор, використовуваний для прискореного обчислення нейронних мереж

TPU (Tensor Processing Unit) – Тензорний процесор, спеціалізований на прискоренні обчислень для машинного навчання

PaLM (Pathways Language Model) – Велика мовна модель, розроблена компанією Google для задач NLP

Falcon – Серія великих мовних моделей із відкритим кодом

RNN (Recurrent Neural Network) – Рекурентна нейронна мережа, призначена для обробки послідовностей даних

LSTM (Long Short-Term Memory) – Модифікація RNN з механізмом довготривалої пам’яті для запам’ятовування довгих залежностей

ВСТУП

У зв'язку з прогресом у сфері цифрових технологій, обсяги даних, що не мають чіткої структури, зростають експоненційно. Сюди входять різноманітні текстові звіти, записи подій (логи), кореспонденція електронної пошти, відгуки споживачів, а також файли, що мають лише часткову структуру, але містять цінну інформацію без чітко окресленої схеми. Щоб ефективно підтримувати керівництво у прийнятті рішень, стежити за дотриманням встановлених лімітів ресурсів, аналізувати операційні процеси чи моніторити стан обладнання, ці масиви даних мусять бути трансформовані у стислі, зрозумілі та чітко структуровані звіти. Класичні методи створення звітних матеріалів, які спираються на фіксовані шаблони та потребують ручної обробки, є надто ресурсомісткими, погано піддаються розширенню та часто не беруть до уваги ситуаційний контекст.

Поява архітектур трансформерів та потужних мовних моделей (LLM) відкрила широкий простір для автоматичного формування текстів, зрозумілих для людини, безпосередньо з комплексних і нерівномірних джерел інформації. Використання LLM на неструктурованих даних, без застосування належних методик попередньої обробки, точного конструювання запитів та верифікації кінцевого результату, несе ризик спотворення фактичного змісту, пропуску критично важливих відомостей або генерування моделлю хибних даних.

Тому виникає актуальна науково-практична задача вдосконалення методу перетворення неструктурованих даних на людиночитану форму з використанням трансформерів таким чином, щоб забезпечити:

– збереження ключових фактів та числових показників (наприклад, квот і лімітів),

– достатню повноту та структурованість звіту,

– високу зрозумілість тексту для кінцевого користувача,

– можливість адаптації методу до різних предметних областей.

Мета роботи - підвищення якості перетворення неструктурованих даних на людиночитану форму шляхом розробки та вдосконалення методу генерації

текстових звітів на основі архітектури трансформерів.

Об’єкт дослідження - процес перетворення неструктурованих даних у текстові звіти.

Предмет дослідження - метод перетворення неструктурованих даних на людиночитану форму, що використовує моделі типу трансформер (LLM) і включає етапи підготовки, генерації та постобробки тексту.

Досягнення вказаної мети забезпечується розв’язуванням наступних задач:

1. Провести огляд і класифікацію неструктурованих даних, що використовуються для побудови звітів.
2. Проаналізувати сучасні підходи до генерації текстів: rule-based, шаблонні, нейронні та моделі на основі трансформерів.
3. Формалізувати задачу перетворення неструктурованих даних у звіти (у тому числі звіти за квотами, показниками тощо).
4. Розробити вдосконалений метод перетворення даних, який включає попередню структуризацію, побудову контексту для LLM та механізм контролю фактичної коректності згенерованих текстів.
5. Спроєктувати та реалізувати програмний прототип системи, що реалізує запропонований метод.
6. Провести експериментальні дослідження на тестових наборах неструктурованих даних, порівняти результати роботи системи з існуючими підходами, оцінити якість текстів за обраними метриками.
7. Зробити висновки щодо ефективності запропонованого методу та можливостей його подальшого розвитку.

Новизна наукового доробку криється не лише у формальному висуненні об’єднаного методу для опрацювання неструктурованих відомостей, але й у всебічному обґрунтуванні концепції, котра синергетично поєднує передові досягнення глибокого навчання з виваженими стадіями попереднього опрацювання та фінального доопрацювання вхідних відомостей. Використання архітектур, подібних до «трансформерів», тут трактується не як ізольований засіб, а як

центральний елемент цілісного механізму, націленого на всеосяжне збереження семантичних кореляцій, фактичної суті та логічної послідовності первинного звіту.

Практична вага здобутих висновків виявляється у значному діапазоні потенційних сфер застосування розробленого методичного підходу та супутнього програмного забезпечення. Зокрема, ці напрацювання мають змогу бути інкорпоровані у рішення для автоматизації створення регуляторної та службової документації, що ґрунтується на об’ємних наборах неструктурованих логів подій, відомостей фінансових операцій, текстових звернень, технічних звітів (логів) та безлічі інших сховищ інформації. Впровадження запропонованого інструментарію потенційно здатне докорінно знизити операційне навантаження на експертів-аналітиків, прискорити терміни підготовки фінальних матеріалів і, що не менш значуще, мінімізувати імовірність помилок, притаманних ручному введенню та аналізу даних

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У нинішньому інформаційному полі, потужність накопичуваних даних збільшується шаленими темпами. Щоденно по планеті продукується більше трьох сотень ексабайтів відомостей — від простих текстових по dato слів у соцмережах до візуальних файлів, звукових доріжок, а також потоків даних із сенсорів у пристроях "Інтернету речей". Значна частка цих даних не має виразної внутрішньої організації, що автоматично відносить їх до класу неструктурованої інформації.

Під неструктурованими даними слід розуміти відомості, котрі неможливо втиснути у заздалегідь окреслений шаблон чи спосіб їхнього зберігання. До цього відносять як живу мову (документи, коментарі, репортажі), так і медіафайли (картинки, звук, кінозаписи), чи навіть формати з частково налагодженою логікою (наприклад, JSON, XML, HTML), де сама структура є в наявності лише частково, а сенсовий зміст залишається змінним і нерівномірним.

1.1 Поняття та класифікація неструктурованих даних

У нинішньому інформаційному середовищі, де цифрові обсяги накопичуються шаленими темпами, простежується тенденція: значний відсоток даних набуває форми неструктурованої або лише частково впорядкованої. Корінь цього явища полягає в тому, що вагома частина інформаційних потоків генерується не за допомогою стандартизованих табличних структур чи заздалегідь визначених реєстрів, а як невіддільний наслідок діяльності користувачів, програмних комплексів, різноманітних датчиків чи сервісів комунікації. Як результат, виникає надзвичайна варіативність даних: вони не підкоряються суворим схемам, не мають чітко визначених полів чи типів даних і часто постають у формі вільного тексту чи довільного мультимедійного контенту.

Під неструктурованими даними зазвичай мають на увазі інформацію, котра не зберігається у вигляді записів із фіксованими параметрами, що виходять із

попередньо затвердженої схеми (як це прийнято у реляційних сховищах), а існує у відкритому, нерегламентованому форматі. До класичних прикладів таких даних відносять текстові матеріали різної довжини та призначення, електронні листи, коментарі відвідувачів соціальних мереж чи бізнес-платформ, системні логи, протоколи зустрічей, конфігураційні файли, повідомлення у сервісах миттєвих розмов тощо. Увесь цей масив даних може суттєво розрізнятися ступенем впорядкованості, манерою викладу, внутрішньою будовою та контекстом, що ускладнює їхню пряму машинну інтерпретацію.

У професійній сфері та під час проведення аналізу даних нерідко послуговуються і терміном «напівструктуровані дані». На противагу абсолютно неорганізованій інформації, ці дані демонструють певну внутрішню структуру, оскільки використовують відповідні мітки або теги, які дають змогу частково описати їхню організацію. До цієї категорії належать поширені формати, такі як JSON, XML, YAML, HTML, а також файли CSV, які можуть мати змінні набори полів чи вбудовані структури. Подібні типи даних можна позиціонувати як перехідну стадію між традиційними впорядкованими таблицями та абсолютно неформалізованим текстом: вони не володіють жорсткими схемами та типами, проте водночас забезпечують мінімальний рівень формалізації, цілком достатній для початкового автоматичного розбору.. Ключові характеристики неструктурованих даних:

Відсутність схеми (Schema-less): Дані не відповідають жорсткій структурі. Кожен документ або об'єкт може містити свій унікальний набір інформації.

Велика різноманітність форматів: Включають текст, медіафайли, бінарні дані.

Контекстуальна залежність: Значення даних часто залежить від контексту, що вимагає складних алгоритмів для вилучення смислу (наприклад, NLP).

Неоднорідність: Дані можуть бути неповними, суперечливими або мати різний ступінь деталізації.

Типові приклади неструктурованих даних включають:

Текстові документи: юридичні контракти, дослідницькі звіти, технічні мануали, книги, дисертації.

Комунікації: електронні листи, повідомлення в месенджерах, транскрипти дзвінків, коментарі в соціальних мережах.

Системна інформація: журнали подій (логи), файли конфігурацій, дампи пам'яті.

У реальності, здебільшого, виділяють таку групу, як напівструктуровані відомості (Semi-structured data), котра займає позицію посередині між суворо табличними форматами та абсолютно неорганізованим текстовим масивом.

Ці напівструктуровані дані володіють певною внутрішньою структурою і містять мітки або дескриптори (тобто метадані), що сприяють розмежуванню та розпізнаванню окремих складових. Водночас, їм не властива фіксована, обов'язкова схема, що дає можливість зручно вводити додаткові атрибути або ж ігнорувати вже наявні. Приклади напівструктурованих даних:

Формати обміну даними: JSON (*JavaScript Object Notation*), XML (*Extensible Markup Language*). Ці формати використовують пари ключ-значення або теги для організації ієрархічної структури.

Веб-документи: HTML-сторінки.

Спеціалізовані формати: Файли CSV (*Comma Separated Values*) або YAML, де кількість та порядок колонок може змінюватися від файлу до файлу, або лог-файли з власним синтаксисом.

Обробка напівструктурованих даних є дещо простішою, ніж неструктурованих, оскільки маркери вже надають початковий рівень організації.

За типом носія та способом представлення неструктуровані дані можна поділити на такі основні групи:

1. Текстові дані - це найпоширеніший тип неструктурованих даних. Їх обробка є критично важливою для бізнес-аналітики, оскільки вони містять безпосередню думку клієнтів, деталі транзакцій та системні збої.

Вільний текст (Free Text): Довгі, зв'язні документи (статті, звіти, політики компанії, описи продуктів).

Короткі повідомлення: Чати, повідомлення від служби підтримки (тікети), коментарі користувачів, пости в блогах.

Журнали подій (Logs): Системні логи (*system logs*), логи додатків (*application logs*), аудиторські сліди (*audit trails*). Ці дані є послідовними записами подій, часто з мітками часу, але їхнє внутрішнє наповнення (текст помилки, опис дії) є неструктурованим.

2. Мультимедійні дані - це інформація, яка потребує спеціалізованих алгоритмів для розпізнавання, аналізу та вилучення метаданих.

Зображення та графіка: Фотографії, скани документів (для OCR), діаграми, інфографіка.

Аудіо- та відеозаписи: Записи розмов кол-центрів (для аналізу тональності), відео нарад, записи з камер спостереження.

Геопросторові дані: Неструктуровані карти або зображення із супутників.

3. Напівструктуровані формати - це формати, які поєднують текст та мінімальну структуру.

Дані веб-API: Відповіді у форматі JSON або XML, які можуть мати різну глибину вкладеності.

Конфігураційні файли**:** Файли з налаштуваннями (*INI, CFG*), де структура визначається доменною логікою, а не жорсткою схемою бази даних.

Усі ці розмаїття даних об’єднані однією визначальною рисою: їх упорядкування та опрацювання є суттєво складнішим завданням, аніж робота з даними, що мають чітко визначену табличну структуру. Щоб виокремити сутності, числові величини, часові залежності чи певні події автоматичним шляхом, мусимо задіяти цілий арсенал методологій — від систем обробки природної мови (НЛП) до аналітичних моделей часових послідовностей, методів класифікації графічних даних, сегментації сигналів та інших інтелектуальних алгоритмів.

Дивовижно, але саме ці "неорганізовані" масиви відомостей, незважаючи на їхню відсутність фіксованої форми, нерідко містять найцінніші відомості для

проведення ґрунтовного аналізу, формування бізнес-прогнозів, здійснення аудиторських перевірок, ідентифікації відхилень чи допомоги у формуванні керівних постанов. У контексті цього дослідження левова частка уваги зосереджена саме на текстових даних, які є або повністю неструктурованими, або мають часткову структуру, оскільки вони найчастіше фігурують у практичних системах і водночас володіють найвищим аналітичним потенціалом.

Підвищену значущість надано таким видам джерел, як службові журнали (лог-файли), довільні текстові звіти, записи щодо квот та обсягів спожитих ресурсів, а також комбіновані формати, що зводять докупи текст і числові складові. Перетворення та глибинне опрацювання саме цих типів інформації становить центральне завдання, яке усувається у межах запропонованого методу.

1.2 Людиночитана форма даних та вимоги до текстових звітів

У сучасних обчислювальних комплексах, призначених для оперування даними та їхнього дослідження, де щоденно обробляються колосальні обсяги різнорідної інформації, з'являється нагальна потреба не лише акумулювати та зберігати відомості, але й належним чином їх візуалізувати (подавати). Користувачі цих платформ, незалежно від рівня їхньої технічної підкованості, потребують доступу до інформації таким чином, щоб мати змогу за мить схопити суть поточного стану речей, виділити найсуттєвіші аспекти та прийняти обґрунтовані рішення. З огляду на цю вимогу, процес формування представлення даних, що буде інтуїтивно зрозумілим для сприйняття людиною, набуває першочергового значення.

Під формою, що легко піддається людському розумінню, ми розуміємо таке впорядкування відомостей, де інформація подається у спосіб, який є прозорим, логічно структурованим і змістовним, що унеможливлює необхідність проведення додаткових технічних маніпуляцій чи ретельного вивчення сирих, неконсолідованих записів. По суті, це такий формат, який дозволяє особі, що застосовує ці дані — чи то керівник вищої ланки, спеціаліст із аналітики,

виконавець робіт чи інший професіонал — засвоїти зміст, не занурюючись у лабіринти технічних деталей або особливості внутрішньої архітектури самої системи.

Створення формату подачі даних, оптимізованого для людського сприйняття, стає критично важливим, особливо тоді, коли вихідні відомості характеризуються відсутністю чіткої структури, є лише частково впорядкованими, подані у фрагментарному вигляді або переобтяжені великою кількістю низькорівневих технічних приміток. У подібних сценаріях автоматична генерація текстових резюме перетворюється на ключовий інструмент, здатний перетворити щільні інформаційні потоки на упорядкований, цінний для подальшого аналізу документ. Більшість програмних рішень встановлюють певний перелік базових критеріїв до цих текстових звітів, які визначають їхню реальну корисність, якість виконання та придатність до практичного використання.Основні характеристики людиночитаної форми текстових звітів:

1. Наявність чіткої логічної структури.

Рядовому користувачеві, як правило, нема потреби вникати у всі події, що відбуваються на низькому рівні, чи заглиблюватися у другорядні технічні нюанси. Натомість, значно більшої ваги набуває донесення головних векторів розвитку, загальних підсумків та тих висновків, які мають прямий вплив на сприйняття загальної картини. Отже, завдання тексту полягає не у простому фіксуванні наявних даних, а в їх осмисленні та подачі через призму аналізу.

2. Узагальнення та висновки.

Користувач зазвичай не потребує ознайомлення з усіма низькорівневими подіями або дрібними технічними деталями. Значно важливішим є подання ключових тенденцій, узагальнень та висновків, які безпосередньо впливають на розуміння ситуації. Таким чином, текст має не просто відтворювати дані, а інтерпретувати їх.

3. Виділення важливих показників і ключових фактів.

Ось приклади показників, що можуть бути використані: ліміти, квоти, виявлені расходження, тенденції, рівні використання ресурсів, а також часові

зв'язки між подіями. Відокремлення цієї інформації здатне відбуватися або через текстові методи (як-от виділення чи коментарі), або за допомогою структурного форматування (приміром, таблиць чи переліків).

4.Зрозумілість мови та термінології.

Написання тексту варто адаптувати відповідно до рівня знань цільової аудиторії. Якщо звіт розрахований на людей, які не є фахівцями у технічній сфері, варто або зменшити використання вузькоспеціалізованої термінології, або ж долучати до неї лаконічні роз’яснення. Такий підхід дасть змогу запобігти двозначності та гарантує правильне розуміння викладеного матеріалу.

5. Відсутність надмірних технічних деталей.

Технічні логи, діагностичні сповіщення, тимчасові збої чи службова довідка можуть мати значення для профільних експертів, проте для загального огляду вони, як правило, не є на часі. Документ має фокусуватися на фінальному доробку, а не на нюансах процесу його досягнення.Щодо критеріїв якості звітів, що створюються машиною, їхнє значення є першочерговим, адже в рамках цього дослідження ми сфокусовані саме на автоматичному формуванні текстових матеріалів, базованих на неформалізованих даних.

1. Точність.

Усі ключові числові величини, визначені моменти часу, унікальні номери, наявні активи та значущі факти мусять бути переказані абсолютно точно, без жодного викривлення. Завдання моделі полягає в тому, щоб уникнути фальсифікації даних, не прирівнювати події, які не мають спільного, та не замінювати показники хибними. Будь-яка невідповідність встановленим фактам здатна призвести до хибних рішень і, як наслідок, до невірних узагальнень..

2. Повнота.

Звіт мусить охопити увесь спектр ключових моментів, що стосуються функціонування системи чи аналізованої сфери: будь-які відступи від встановлених стандартів, випадки виходу за межі виділених лімітів, недотримання обумовлених рівнів обслуговування, а також події, що є або нетиповими, або

схильні до рецидиву. Якщо звіт буде невсеосяжним, це може призвести до некоректного розуміння фактичного стану речей.

3. Зрозумілість і читабельність

Мова тексту повинна відповідати нормам української граматики, бути витримана у відповідному стилі й не містити жодних суперечностей. Плавно мають бути забезпечені логічні переходи між усіма розділами документа, а його внутрішня побудова — легкою для розуміння. Навіть якщо вихідна інформація була високотехнічною, звіт мусить бути доступним для сприйняття читачем, який володіє лише базовими знаннями.

4. Структурованість і послідовність викладу.

Подання відомостей має відбуватися за принципом "від загального до окремого": с спершу варто надати стисле резюме чи загальний огляд, а вже потім занурюватися у деталізацію по окремих секціях (таких як інциденти, використані ресурси, часові проміжки). Ця методологія узгоджується з загальноприйнятими нормами, що стосуються створення технічної документації та проведення бізнес-аналізу.

5. Уніфікованість та стандартизованість.

Усі вихідні дані, генеровані системою, мусять дотримуватися уніфікованої логіки представлення відомостей, а також мати подібну структуру та манеру оформлення. Такий підхід дає змогу спростити їх подальшу машину обробку, покращує можливість зіставлення даних та гарантує послідовність у їх інтерпретації.

6. Контрольованість процесу генерації.

Ключовим моментом є те, щоб система надавала можливість встановлювати параметри для генерації звітності: приміром, ступінь деталізації, для кого він призначений, якою мовою, спосіб відображення чисел, чи навіть певні уривки, котрі мають бути присутніми або ж пропущені. Подібна адаптивність значно розширює обсяг використання цього інструментарію.

Отже, справа автоматичного створення звітів, зрозумілих для людини, на базі неструктурованих даних – це завдання, що охоплює декілька рівнів і є доволі

заплутаним. Воно передбачає не лише здатність точно вихоплювати потрібні відомості та вміло їх підсумовувати, а й конструювати текст, який би логічно вибудовувався, легко читався та відповідав саме тим, що треба користувачеві. З огляду на це, вдосконалення способу переробки подібних даних із застосуванням архітектур на кшталт «трансформерів» виступає центральним вектором роботи, адже ці моделі мають потенціал об’єднати хірургічну точність аналізу з природністю та гнучкістю мовлення, притаманного людям.

1.3 Архітектури трансформерів та великих мовних моделей

Протягом останніх років будова трансформера закріпилася як провідна парадигма, що задає вектор розвитку переважної більшості сучасних рішень у сфері обробки людської мови, генерації текстових матеріалів та глибокого аналізу інформації. Ці архітектурні рішення фактично витіснили попередньо використовувані методології, включаючи рекурентні нейронні мережі (RNN), системи з довгим короткочасним запам'ятовуванням (LSTM) або ж моделі, зведені на базі згорткових структур. Причина цього — їхня демонстрована незрівнянно вища продуктивність, легкість у кастомізації, відмінна здатність до експоненційного зростання обсягів та коректне опрацювання велетенських корпусів навчальних даних. На сьогоднішній день функціональність більшості великих мовних моделей (LLM) цілковито спирається на трансформерну архітектуру, що, своєю чергою, зробило їх універсальним набором інструментів для вирішення широкого спектру прикладних задач.

Стислий опис архітектури трансформера уперше було представлено у 2017 році у фундаментальній праці під назвою “Attention Is All You Need”, і вже через кілька років вона стала де-факто еталоном у царині NLP. Ядром її вражаючої ефективності є механізм самозваги (self-attention) — спеціалізований процес, який дозволяє системі зважувати відносну важливість між різнорідними сегментами вхідного потоку інформації.

Ключовою складовою даної архітектурної побудови є саме механізм самозваги — специфічна операція, котра дає змогу системі автономно та динамічно визначати ступінь значущості кожного окремого елемента (токену) усередині послідовності, базуючись на його оточенні. Інакше кажучи, замість лінійного сканування тексту, модель проводить всебічне зіставлення між собою усіх його частин, прагнучи ідентифікувати, котрі саме лексичні одиниці чи синтаксичні групи є найбільш релевантними для оброблюваного фрагмента. Ця характеристика перетворює самозвагу на всеосяжний інструмент, здатний враховувати як безпосередні (локальні) взаємозв’язки, так і віддалені (глобальні) залежності.

На противагу рекурентним системам, які змушені оперувати інформацією послідовно і зберігати її у внутрішньому стані, трансформер має імператив аналізувати всю послідовність у режимі паралельності. Це не лише усуває проблему, властиву обмеженням у паралельній реалізації рекурентних підходів, але й відкриває значний потенціал для глибокого контекстуального опрацювання. Завдяки можливості паралельного навчання, модель демонструє підвищену масштабованість, здатна ефективно обробляти гігантські текстові масиви та виокремлювати складні, приховані закономірності, які були або недосяжними, або надмірно ресурсомісткими для обчислення у рамках архітектур попереднього покоління.

В результаті, трансформери забезпечили якісний стрибок у продуктивності мовних моделей, дозволивши комбінувати високу швидкість опрацювання, точність, гнучкість у відображенні контекстуальних зв'язків та здатність до адаптації до багатоманітних завдань — від мовного перекладу до формування аналітичних звітів та експертних описів. До ключових особливостей трансформерів належать:

Self-attention (механізм самоуваги), що дозволяє кожному токену “бачити” інші токени в межах послідовності та визначати їхню релевантність. Завдяки цьому модель виявляє складні семантичні й синтаксичні зв’язки.

Паралельна обробка токенів, що усуває необхідність покрокового проходження послідовності. Це суттєво прискорює навчання та дозволяє ефективно використовувати GPU/TPU.

Підтримка довгих контекстів, яка забезпечується масштабуванням архітектури, модифікаціями шарів уваги та спеціальними позиційними механізмами.

У класичному вигляді трансформер складається з набору шарів енкодера та/або декодера. Кожен шар включає:

- багатоголовий механізм уваги (multi-head attention), що дозволяє моделі одночасно аналізувати інформацію на різних “рівнях” або з різних точок огляду;
- позиційні перетворювачі (feed-forward networks), які виконують нелінійні трансформації над поданнями токенів;
- нормалізацію шарів (layer normalization), що стабілізує навчання;
- резидуальні (skip) зв’язки, які покращують градієнтний потік і допомагають уникати деградації під час збільшення глибини мережі.

Ці компоненти забезпечують трансформеру здатність не лише коректно моделювати структуру тексту, але й адаптуватися до широкого спектра задач.

Моделі, що стосуються великих мовних моделей (LLM), формують окремий клас, який постав як надбудова над архітектурою трансформерів, що лягла в основу їхнього створення. Вони характеризуються:

- величезною кількістю параметрів (від сотень мільйонів до сотень мільярдів);
- попереднім тренуванням на масштабних корпусах текстів, що охоплюють різні стилі, домени й формати даних;
- універсальністю застосувань — від генерації текстів до класифікації, від пошуку інформації до логічного міркування.

Великі мовні моделі (LLM) мають потенціал для виконання різноманітних функцій: вони можуть осягати зміст та значення наданого тексту, реалізовувати заплутані директиви, вказані у запиті, створювати впорядковані та послідовні

матеріали (наприклад, виклади подій, стислі огляди, доповіді), давати відповіді на запитання, тлумачити інформацію, а також вибудовувати нескладні умовиводи. З огляду на це, LLM перетворилися на багатофункціональні засоби для спрощення обробки даних, поданих у неформатованому вигляді.

Щодо завдань, які стосуються переведення неструктурованих текстових даних у формат, зрозумілий для людини, то для роботи з ними найбільшу вагу мають такі особливості великих мовних моделей:

Інструкційне узгодження (instruction-following) Сучасні великі мовні моделі (LLM) проходять спеціалізоване доопрацювання, аби забезпечити точне дотримання у письмових завданнях таких аспектів, як усталений формат, бажаний стилістичний відтінок, логічна послідовність структури документа, визначені пріоритети та інші задані критерії.

Можливість роботи з довгими контекстами Дана можливість дає змогу додавати до запиту шматки лог-файлів, хроніки подій, сповіщення системи, відомості щодо лімітів, минулі дані та інші неформатовані відомості, що потребують трансформації у звітну форму.

Few-shot та zero-shot узагальнення Здатна особа, за умови надання лише невеликої кількості зразків чи шаблонів, або ж взагалі без них, — виключно завдяки ретельно вибудуваній інструкції — опанувати спроможність виконувати нове завдання.

Незважаючи на надзвичайно широкий спектр застосувань великомасштабних мовних моделей (ВММ) та їхню доведену корисність у розв'язанні багатьох практичних завдань, слід пам'ятати: ВММ аж ніяк не є панацеєю чи інструментом із абсолютною надійністю. Вони містять низку внутрішніх (архітектурних), методичних та концептуальних обмежень, які диктують особливі умови їхнього використання, особливо там, де критично важливі достовірність, формальна коректність виведення та сувора відповідність початковим даним.

Однією з найбільш помітних вад є схильність моделі до так званих "видінь" або "галюцинацій". Це явище полягає у тому, що модель продукує інформацію, яка

насправді не міститься у наданому вихідному матеріалі. Модель здатна вигадувати факти, події або заповнювати будь-які логічні прогалини на основі того, що їй здається правдоподібним, хоча це й не має реального підґрунтя. Причина може критися як у недостатньому обсязі контексту, так і в самій статистичній природі моделі, оскільки вона налаштована на ймовірнісне продовження тексту, а не на верифікацію істинності тверджень.

Другий ключовий момент – це відсутність жодних гарантій повної збіжності згенерованого тексту з початковими даними. Навіть якщо надано вичерпні інструкції, модель усе ще може хибно трактувати окремі частини вхідної інформації, пропустити суттєві деталі або, навпаки, надмірно узагальнити весь масив даних. Це стає вкрай проблематичним у ситуаціях, коли потрібно відтворювати з максимальною точністю технічні журнали, фінансові проводки чи послідовності подій.

Особливої уваги потребує залежність результату від того, як саме сформульовано запит (промпт). Конкретне словесне формулювання інструкції визначає спосіб, яким модель осмислить поставлене завдання, на які аспекти вхідних даних зосередиться та яку структуру прийме її відповідь. Навіть мінімальні зміни у формулюванні інструкції можуть докорінно змінити якість та кінцевий зміст згенерованого. Звідси випливає, що оператор повинен не лише розуміти загальні принципи роботи моделі, але й володіти навичками конструювання ефективних запитів.

Додатковою технічною межею є ліміт (розмір) контекстного вікна, який вар’юється між різними моделями. Хоча сучасні ВММ можуть обробляти вельми значні обсяги відомостей, вони залишаються обмеженими у здатності опрацьовувати абсолютно необмежені текстові масиви за один раз. Коли вхідні відомості перевищують можливості доступного вікна контексту, виникає нагальна потреба у додатковому розбитті на сегменти, ущільненні інформації або попередньому її відборі.

Сукупність перелічених вище чинників робить використання ВММ у сценаріях, що мають високу критичність, – як-от створення офіційних,

аудиторських або регуляторних звітів – потенційно ризикованим кроком без впровадження додаткових систем контролю та валідації. З огляду на ці ризики, стає очевидним: застосування великих мовних моделей для завдань, які вимагають структурованих, точних та однозначних відомостей, потребує застосування якогось спеціального підходу. У цьому контексті, "оркестрація" трансформерної моделі означає сукупність методик і робочих процесів, які дозволяють тримати під контролем функціональність моделі на всіх стадіях обробки інформації – від формування вхідного запиту аж до фінальної перевірки отриманого результату.

Першим етапом такої системи оркестрації є скрупульозна підготовка даних, які подаються на вхід. Це включає очищення тексту від будь-якого "шуму", дублюючих фрагментів, артефактів, неузгоджених форматів; а також нормалізацію, зведення в групи та логічне структурування журналів подій або логів. Дані, що пройшли належну підготовку, формують для моделі значно більш передбачуване робоче середовище, що зменшує імовірність некоректних трактувань та втрати принципово важливих даних.

Наступний етап – це створення такого промпта, який би чітко фіксував бажану логіку, структуру та формат підсумкового виведення. Зазвичай такий запит містить опис завдання, чіткі правила генерації, перелік обов'язкових полів та певні обмеження. У більш заплутаних випадках застосовуються приклади (так званий few-shot підхід), спеціально розроблені зразки (шаблони) або заздалегідь структуровані форми для звітів, що допомагають моделі зрозуміти необхідний тон та логіку викладу.

Після того, як від моделі отримано відповідь, необхідним кроком є її подальша обробка. Вона може включати автоматизовану перевірку достовірності фактів, зведення отриманих відомостей у єдиний стандартний вигляд, уніфікацію структур, верифікацію правильності вжитої термінології та блокування появи будь-яких елементів, які не мають підтвердження у вихідному матеріалі. На цій стадії можуть бути використані як традиційні алгоритмічні методи, так і механізми повторного уточнення запиту до моделі.

У підсумку, така багатокомпонентна оркестрація дозволяє функціонуванню трансформера бути не просто автономною системою, що генерує текст на базі статистичних імовірностей, а складовою контрольованого процесу, де пріоритет надається структурності, надійності та абсолютно точній відповідності початковому набору даних. Це набуває особливої цінності у тих сферах застосування, де необхідне виведення має відповідати суворим формальним критеріям, а не лише бути стилістично чи граматично коректним.

1.4 Огляд актуальних підходів у галузі перетворення неструктурованих даних

Проблема опрацювання та приведення до ладу неструктурованих відомостей у формат тексту, який можна аналізувати, логічно організований та легко сприймається, вже тривалий час є пріоритетом для науковців та фахівців-практиків. З огляду на стрімке нарощування обсягів цифрової інформації, класичні засоби, призначені для оперування табличними даними та суворо формалізованими структурами, все частіше демонстрували свою неспроможність. Неструктуровані масиви — включно з логами подій, кореспонденцією електронною поштою, протоколами нарад, відгуками користувачів, коментарями, вільним текстом загалом — концентрували в собі колосальні обсяги відомостей, проте через нестачу чіткої схеми їхня машинна інтерпретація була ускладнена. Саме це й зумовило виникнення потреби у розробці цільових методик, здатних трансформувати такі інформаційні потоки у звіти, зручні для сприйняття людиною.

До ери впровадження передових нейронних технологій, ключовими засобами автоматизації слугували процедурні системи, функціонування яких базувалося на заздалегідь визначеному (власноруч прописаному) алгоритмі змін даних. Найбільш уживаними серед них були:

1. Rule-based системи, що оперували набором умовних операторів та логічних конструкцій. Кожне правило у таких системах відповідало окремому

фрагменту тексту або дії. Звіт формувався за жорстко формалізованою схемою, яка точно відповідала визначеним умовам.

2. Template-based системи, побудовані на основі фіксованих текстових шаблонів, у які підставлялися конкретні значення параметрів, отриманих з баз даних, лог-файлів або протоколів.

Ці системи вирізнялися надзвичайною передбачуваністю та керованістю: змога простежити кожну фразу у створеному тексті до конкретного встановленого правила була повною. Вони забезпечували, що звітність ніколи не виходила за межі первинно наданих даних, а також гарантували однорідність манери викладу. Проте, у міру того як процеси ставали заплутанішими, а обсяги неструктурованих першоджерел зростали, подібні підходи поступово втрачали ефективність. Головним їхнім мінусом була негнучка залежність від формату даних: навіть мінімальні зсуви у структурі журналів подій, найменуваннях атрибутів чи способі оформлення записів спричиняли необхідність переробки великої кількості (десятків) правил. До того ж, ці системи стикалися із серйозними труднощами при роботі з багатозначністю, залежностями від контексту, тонкими стилістичними відтінками та іншими аспектами, які природно властиві людському розумінню, але вкрай складно описуються у формі жорстко визначених умов.

Рух у бік нейромережевих архітектур: RNN, LSTM, GRU. З приходом ери глибокого навчання з'явилася змога будувати системи, які опановують знання з масивів текстових даних, замість того, щоб покладатися виключно на попередньо визначені директиви. Ранні вагомі досягнення у сфері автоматичного створення тексту належать рекурентним нейронним мережам – серед них RNN, LSTM та GRU. Ці структури тренувалися виявляти шаблони у мовних ланцюжках, простежувати зв'язки поміж лексемами та синтаксичними одиницями, що давало змогу генерувати більш органічні та зв'язні наративи.

Завдяки цим розробкам стало можливим задіяти перші засоби, які здатні виходити за межі простого заповнення слотів у шаблоні. Вони могли передавати зміст у більш гнучкій манері, формувати стислі підсумки, корегувати манеру викладу й навіть стисло викладати сутність об'ємних матеріалів. В області роботи

з неструктурованими відомостями, подібні моделі знаходили застосування для автоматичного створення стислих витягів із лог-файлів, формування базових описів подій чи переліків хронологічних фактів.

Проте, старі підходи на основі рекурентних нейронних мереж (РНМ) мали значні вади: вони кепсько давали раду з тривалими залежностями, потерпали від згасання градієнтів і часом не давали змоги генерувати текст належної якості. Саме ці недоліки й підштовхнули до розробки більш потужної архітектури.

Поява архітектури Трансформер кардинально перевернула цю сферу. Завдяки впровадженню механізму взаємної уваги (self-attention), модель отримала здатність паралельно опрацьовувати усі фрагменти вхідного ланцюжка, усуваючи потребу в покроковому обробленні кожного елемента. Це принесло низку вирішальних плюсів:

- можливість працювати з дуже великими фрагментами тексту;
- значно кращі результати у завданнях узагальнення та переформулювання;
- високу гнучкість у роботі зі змішаними типами даних;
- здатність витримувати складні логічні структури та контекстні зв’язки.

Саме з використанням архітектури трансформерів і започаткували розвиток великих мовних моделей (ВММ) — зокрема, GPT, PaLM, LLaMA, Falcon та низки інших. Ці системи пройшли навчання на колосальних обсягах текстових даних, наслідком чого стало їхнє вміння вирішувати надзвичайно різнопланові завдання: починаючи від стислого викладу змісту та перекладу, і аж до створення розлогих документів, аналізу технічної інформації та навіть написання програмного коду.

У контексті задачі перетворення неструктурованих даних LLM стали використовуватися як універсальний механізм для:

- інтерпретації логів у формі описів інцидентів або аварійних подій;
- автоматичного формування бізнес-звітів, включно з поєднанням тексту та числових показників;
- класифікації та узагальнення складних і розрізнених записів;
- побудови багаторівневих текстів, що містять загальні висновки та детальну інформацію;

- перекладу між форматами даних, наприклад із технічного журналу у зрозумілу аналітичну довідку.

Незважаючи на вражаючі можливості сучасних практик та інженерії підказок, великі мовні моделі (LLM) мають власні недоліки. Ці моделі здатні вигадувати факти, хибно інтерпретувати певні частини інформації або ж ігнорувати ключові моменти. Тому й виник новий напрямок на межі науки та практики — промпт-інжиніринг. Він включає:

- розробку оптимальних формулювань запитів;
- визначення структури та послідовності інструкцій;
- уточнення ролі моделі (аналітик, технічний експерт, оператор моніторингу);
- використання підказок, що примушують модель дотримуватися фактичності.

Водночас, парадигми на кшталт «ланцюжка думок» (chain-of-thought) набули розвитку, котрі вимагають прописування проміжних етапів міркувань або декомпозиції складного завдання на менші складові. Такий підхід сприяє генерації більш вивірених і послідовних відповідей.

Крім того, значущим вектором стало злиття потужностей мовних моделей із використанням зовнішніх утиліт (так зване «tool use»). Великі мовні моделі отримують здатність делегувати завдання: вони можуть звертатися до інформаційних сховищ, запускати SQL-запити, опрацьовувати математичні вирази, верифікувати ті чи інші дані, або ж проводити валідацію фактичної інформації силами спеціалізованих програмних модулів.

У складних системах для перевірки часто використовують:

- регулярні вирази та правила відповідності;
- додаткові алгоритми класифікації;
- інші LLM, які оцінюють текст на наявність помилок або пропущених фактів.

Отже, у нинішніх методологіях опрацювання неструктурованих даних давно не йдеться лише про одноразове формування тексту. Навпаки, це перетворилося на багатоетапну процедуру, що охоплює підготовку вхідної інформації, вдосконалення промптів, генерування результату, його перевірку та подальше доопрацювання. Це дозволяє досягти значної точності, керованості та адаптивності, чого не могли забезпечити інструменти попередніх поколінь.

2 АНАЛІЗ МЕТОДІВ ПЕРЕТВОРЕННЯ НЕСТРУКТУРОВАНИХ ДАНИХ НА ЛЮДИНОЧИТАНУ ФОРМУ

У теперішніх інфосистемах першочергове значення набуло питання автоматичного приведення значного обсягу неорганізованої інформації до вигляду, який би сприяв подальшому аналітичному опрацюванню та формуванню управлінських висновків. Щороку збільшується число цифрових платформ, які продукують дедалі масивніші збірки журналів подій, текстових комунікацій, технічних актів, внутрішніх службових документів, стенограм розмов та іншої інформації, зафіксованої у формі, що унеможливлює її пряме застосування. Звідси випливає потреба не лише раціонального архівування цих даних, а й їхнього тлумачення, синтезу та візуалізації у вигляді впорядкованих, зрозумілих людині текстів.

Приведення неструктурованих даних у порядок — це не просто технічна маніпуляція з текстом. Це багатогранний процес, який передбачає усвідомлення змісту, виокремлення ключових компонентів, відділення критично важливих відомостей від другорядних, а також адекватне формулювання кінцевого результату з огляду на специфіку певної бізнес-сфери. Саме тому вивчення способів трансформації неорганізованої інформації є суттєвим етапом у створенні систем контролю, інструментів для допомоги ухваленню рішень, засобів автоматизації внутрішнього документообігу, генерації аналітичних дайджестів чи систем превентивного виявлення нетипової поведінки.

У цьому параграфі ми розглянемо спектр методів, які використовувалися в IT-системах протягом минулих десятиліть — починаючи від традиційних систем, заснованих на правилах, аж до передових архітектур глибокого навчання. Такий ретроспективний погляд є важливим, оскільки він дозволяє оцінити переваги та недоліки наявних підходів, окреслити сфери, де вони найбільш ефективні, а також зрозуміти чинники, які спричинили еволюцію від систем зі строгим набором інструкцій до гнучких, самоадаптивних рішень.

На початку свого шляху системи, призначені для автоматичного формування текстів, були зосереджені на прогнозованих випадках і функціонували на базі жорстко прописаних алгоритмічних інструкцій. Ці рішення демонстрували хороші показники, коли обсяги інформації були помірними, а моделі функціонування систем — чітко окресленими. Однак, у міру того, як IT-середовище еволюціонувало — із запровадженням хмарних рішень, розширенням розподілених мереж і впровадженням мікросервісної структури — профіль накопичених даних набув значно більшої складності: вони стали більш багатогранними, мінливими та менш передбачуваними. Внаслідок цього, класичні підходи, засновані на правилах (rule-based), дедалі частіше демонструють свою неспроможність: вони погано впораються з величезною кількістю відмінностей у логах, вимагають постійного ручного доопрацювання та неспроможні динамічно реагувати на виникнення непередбачуваних обставин. Це спричинило нагальну потребу у відшуканні більш узагальнених технік, які б могли функціонувати в умовах невизначеності та оперувати заплутаними текстовими даними. Запровадження рішень на базі шаблонів (template-based) дозволило наростити рівень автоматизації, але вони так само залишалися зафіксованими у межах суворої структури та обмеженого переліку можливих ситуацій.

Лише з моменту появи систем, заснованих на навчанні (machine learning models), а саме рекурентних нейронних мереж та архітектур типу «трансформер», відкрилася реальна можливість суттєво покращити як гнучкість, так і якість автоматизованого створення текстів. Проте, щоб повноцінно усвідомити усі позитивні аспекти сучасних методик, спершу необхідно проаналізувати ті базові концепції, на яких ґрунтується уся історія розвитку цієї сфери.

Отож, подальший розділ зосереджений на глибинному розгляді традиційних, заснованих на правилах та зразках, підходів до створення звітів — це ті самі технології, які заклали фундамент для усіх наступних розробок у царині обробки неструктурованих текстових даних.

2.1 Класичні rule-based та шаблонні підходи до генерації звітів

В історичному контексті, найперші прагнення до автоматичного перетворення інформації у формат, придатний для сприйняття людиною, були міцно пов'язані із системами, заснованими на правилах, тобто з тими методологіями, де увесь процес виведення тексту диктувався суворим набором жорстко заданих правил, які створювалися фахівцями у відповідній сфері. Ці системи служили фундаментом для розробки ранніх інформаційних платформ, адже забезпечували передбачуваність функціонування та стовідсоткову відповідність фінального тексту встановленим внутрішнім стандартам установи. Функціонування такого типу системи, що оперує правилами, базується на узгодженій роботі трьох ключових елементів:

1. набір умов (тригери або логічні вирази).
2. набір дій або текстових конструкцій.
3. механізм інтерпретації правил, який визначає порядок їх виконання.

Коли певна умова виконується — припустімо, з'явилося повідомлення про перевищення ліміту у полі журналу чи зафіксовано понад три попереджувальні сигнали упродовж однієї хвилини — система активує відповідний алгоритм і додає до фінального документа заздалегідь сформований текстовий фрагмент. У найбазовішому вигляді це реалізується через послідовність конструкцій типу ЯКЩО-ТО, де кожна така пара визначає одну конкретну типову обставину.

Ключовим плюсом застосування системи, що ґрунтується на правилах (rule-based), є абсолютне володіння формуванням фінального тексту. Усі речення, будь-яка частина підсумкового висновку відомі творцям системи на етапі проектування. Це дає змогу компаніям, які функціонують у суворо регульованих секторах (фінанси, зв'язок, енергетика, охорона здоров'я), без проблем виконувати жорсткі критерії аудиторських перевірок та сертифікаційних процедур. Звітний текст завжди чітко відповідає встановленим нормам, що спрощує його валідацію на предмет відповідності стандартам ведення документації.

Ще однією вагомою перевагою рішень, побудованих на правилах, є те, що вони не схильні до вигадок чи так званих "галюцинацій", які притаманні сучасним нейромережам. Текст генерується системою виключно відповідно до директив, чітко прописаних оператором. Це робить такі підходи надзвичайно затребуваними у відповідальних системах, де хибна інтерпретація здатна спричинити значні фінансові втрати чи операційні збої.

Проте, попри згадані плюси, підходи, засновані на правилах, мають низку суттєвих мінусів. Найбільш очевидний — обмежена здатність до розширення. Зі зростанням кількості потенційних обставин, випадків чи типів журнальних записів, обсяг необхідних правил збільшується у геометричній прогресії. До прикладу, якщо система оперує 10 типами подій, 5 рівнями важливості та 3 категоріями активів, теоретично виникає потреба у прописі вже 150 можливих комбінацій сценаріїв. На ділі це призводить до того, що набір правил перетворюється на заплутану, громіздку та важкокеровану конструкцію, яка вимагає неперервного ручного оновлення.

Понад те, системи, керовані правилами, зазвичай оперують лише тими даними, які були заздалегідь ідентифіковані та враховані. Якщо у логах виникає повідомлення нового формату чи реєструється нестандартна ситуація, система або ігнорує цей випадок, або ж формує неповну звітність. Отже, методи на основі правил демонструють ефективність лише в умовах стабільного функціонування із чітко окресленими сценаріями. Далі, на рисунку 2.1, ми можемо оглянути типову rule-based систему.

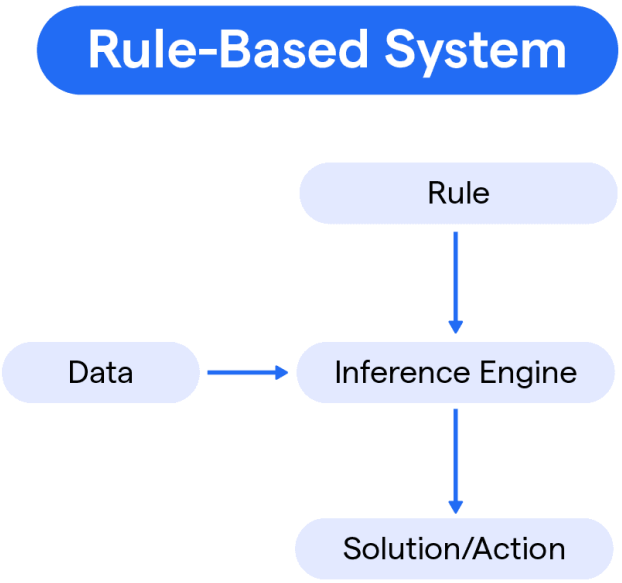


Рис. 2.1 Система Rule-Based

На рисунку 2.2 також наявне зображення rule-based системи в штучному інтелекті (ШІ), де до основних частин такої будови входять: сховище знань (*Knowledge Base*), апарат для виведення логічних наслідків (*Inference Engine*) та оперативна пам'ять (*Working Memory*). Ці складові взаємодіють між собою з метою опрацювання даних та отримання якихось висновків чи формування відповіді.

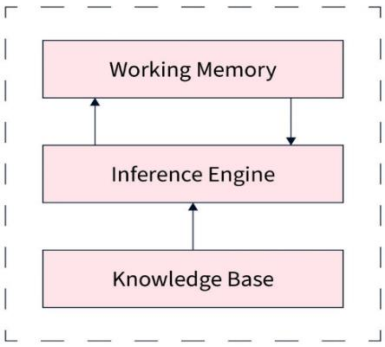


Рис. 2.2 Система Rule-Based у ШІ

Методи, що базуються на шаблонах, постали як логічне продовження еволюції після систем, керованих правилами. На відміну від останніх, підходи, що використовують шаблони, уникають спроб охопити кожну потенційну обставину через перелік чітких умов. Натомість, їхня сутність полягає у застосуванні заздалегідь створених текстових заготовок. У ці заготовки згодом інтегруються змінні дані, взяті з інформаційних сховищ , структурованих записів.

Архітектурна модель, яка охоплює компоненти "Модель", "Шаблон" та "Оцінювач" (Model, Template та Evaluator), є найбільш наочною демонстрацією того, як Шаблонний підхід застосовується у процесі формування будь-яких документів чи аналітичних звітів.

Хоча на схемі вказано "Generated Code" (Згенерований Код), цей принцип є універсальним і повністю застосовується до генерації текстових звітів (NLG), де:

- Model (Модель) — це Вхідні Дані (структуровані показники, факти).
- Template (Шаблон) — це Текстовий Зразок (з плейсхолдерами).
- Generated Code/Report — це Фінальний Згенерований Звіт.

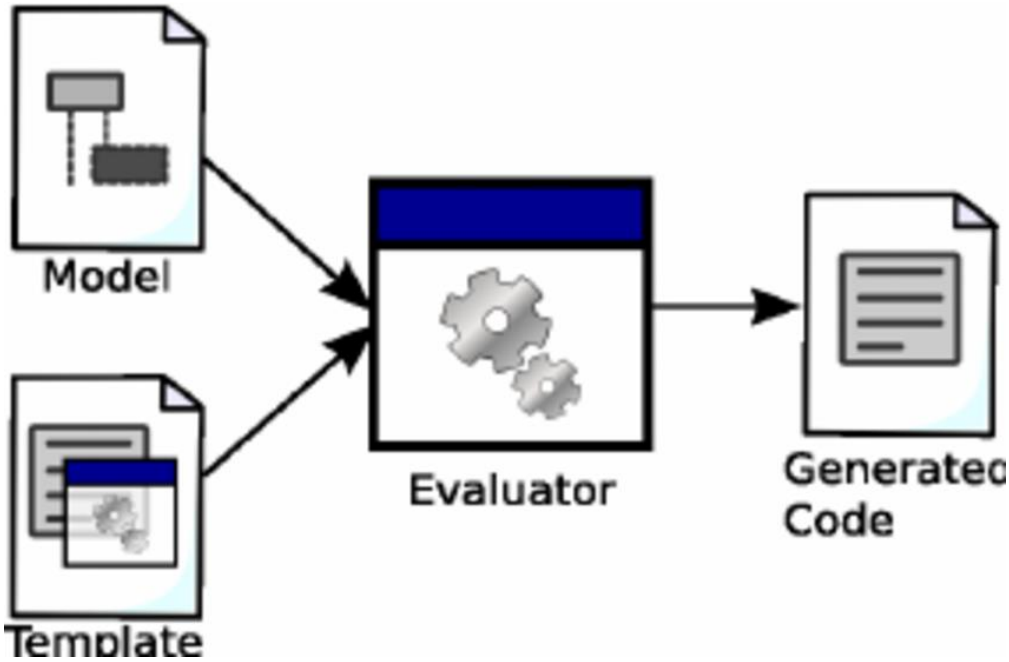


Рис. 2.3 Метод генерації

Задум полягає у заздалегідь окресленій текстовій структурі, де ж конкретні елементи варіюються відповідно до вхідних відомостей. Цей метод фактично

зливає принципи програмування із мовним оформленням, надаючи змогу автоматизувати створення документів без потреби у всеосяжній генерації словесного контенту. Методи, засновані на шаблонах, пропонують низку вигод:

- вони значно гнучкіші за rule-based системи, оскільки допускають варіативність формулювань.
- забезпечують хорошу повторюваність структури звіту.
- відносно легко підтримуються у разі зміни вимог до форматів документації.

У більш складних системах шаблони здатні оперувати умовними операторами на кшталт if-else, містити альтернативні варіанти висловлювань та враховувати контекстуальні залежності (наприклад, «якщо показник є критичним — додати сповіщення», «якщо кількість інцидентів за певний проміжок часу перевищила три — додати підсумковий коментар»). Це дає змогу формувати тексти, що звучать значно природніше, поменшуючи при цьому одноманітність мови.

Проте методи, засновані на шаблонах, також не позбавлені недоліків. По-перше, їхня ефективність напряму залежить від якості структури вхідних даних. Якщо маємо справу з неструктурованим текстовим журналом, його спершу необхідно привести до формалізованого набору параметрів, з якими можуть взаємодіяти шаблони. По-друге, шаблон завжди диктує рамки для формулювань: він нездатен самостійно аналізувати ситуації, які є заплутаними чи багатозначними. Хоча текст, створений цим способом, може бути цілком інформативним, йому зазвичай бракує гнучкості чи здатності до глибокого аналізу.

Хоча обидва методи – із застосуванням правил (rule-based) та шаблонні (template-based) – зараховуються до традиційних технік генерації текстового контенту, між ними простежуються значні розбіжності:

1. Ступінь гнучкості, підхід, що базується на правилах, створює текст, суворо дотримуючись встановленої логіки, тоді як шаблонний метод оперує заздалегідь визначеною структурою. Як наслідок, шаблонні рішення пропонують

ширший спектр варіантів формулювань, проте вони менш пристосовані до логічної обробки непередбачуваних обставин.

2. Можливості розширення Системи, засновані на правилах, стикаються з проблемою стрімкого збільшення обсягу необхідних правил. Шаблонні механізми демонструють кращу масштабованість, але при цьому їх ефективність тісно пов’язана з необхідністю збереження сталості у форматі вихідних звітів.

3. Вимоги до структури вхідних даних, системи на основі правил нерідко здатні опрацьовувати дані безпосередньо з журналів подій (логів). Натомість, шаблонні методи вимагають наявності чітко структурованих вхідних параметрів.

4. Характер отриманого тексту, завдяки фіксованій структурі, шаблонні методи зазвичай забезпечують кращу читабельність згенерованого контенту. Методи, що керуються правилами, часом можуть породжувати висловлювання, які здаються занадто уривчастими або позбавленими природності.

5. Придатність для аналітичних завдань, в цілому, обидва методи виявляються не надто ефективними, коли мова йде про глибоке опрацювання, узагальнення чи пояснення значних обсягів неорганізованої текстової інформації.

Для наочного узагальнення ключових відмінностей та порівняння продуктивності цих двох класичних підходів використовується таблиця 2.1:

Таблиця 2.1

Характеристика	Rule-Based Підхід	Шаблонний (Template-Based) Підхід
Гнучкість	Низька	Середня
Масштабованість	Низька	Середня
Варіативність тексту	Дуже низька	Помірна
Ризик “галюцинацій”	Відсутній	Відсутній
Залежність від структури входу	Слабка	Висока
Потреба у підтримці	Висока	Середня
Якість тексту	Суха, формальна	Краща, але стандартна
Вартість розробки	Висока (з ростом правил)	Помірна
Можливість узагальнення	Немає	Майже немає

Протягом тривалого часу системи, що автоматично створюють звіти, спиралися передусім на методи, що базувалися на правилах і шаблонах. Такі підходи демонструють високу ефективність у секторах, де ситуація є сталою, прогнозованою, а вимоги чітко структуровані. Проте, сучасні інформаційні платформи продукують колосальні масиви даних, які є різнорідними й постійно змінюються, і які практично неможливо адекватно описати за допомогою усталених статистичних зразків чи фіксованих правил. Як наслідок, традиційні інструменти дедалі частіше виявляють обмежену адаптивність, що диктує необхідність їхнього розширення або повної заміни на більш просунуті інтелектуальні рішення на базі штучного інтелекту, зокрема з використанням глибинних нейронних мереж та великомасштабних мовних моделей типу трансформерів.

2.2 Нейронні мережі та трансформери в задачах генерації тексту

Із розвитком сучасних інфосистем, питання автоматичного приведення хитромудрих, неструктурованих відомостей до логічної та зрозумілої текстової оболонки набуває першочергової ваги. Кількість інформації, що з’являється завдяки цифровій інфраструктурі, зростає стрімко, а формати цих даних стають усе більш різноплановими. У цій ситуації, людські ресурси просто не в змозі осягнути всі накопичені логи подій, технічні виписки чи розкидані текстові нотатки. Звідси виникає імператив у створенні механізмів, які б могли самостійно аналізувати величезні масиви даних, виокремлювати суттєве та конструювати звіти у форматі, доступному для сприйняття людиною. Саме на цьому тлі мережі нейронного типу та новітні архітектури-трансформери перетворились на вирішальний інструмент, який кардинально трансформував сам метод генерування тексту.

Перш ніж ми заглибимося у вивчення трансформерів, слід зрозуміти, що поява нейромереж у сфері текстогенерації не була чимось несподіваним. До середини нульових років домінували такі статистичні підходи, як, наприклад, n-грамові моделі чи ланцюги Маркова. Проте вони мали низку істотних недоліків:

- відсутність здатності до узагальнення — модель розглядає лише фіксовані шаблони слів;
- проблеми зі sparsity — рідкісні або нові слова практично не враховуються;
- надмірна залежність від локального контексту;
- неможливість глибоко моделювати семантику тексту.

Натомість, нейронні мережі дали змогу розробити моделі, здатні засвоювати знання з масивних текстових обсягів, виокремлюючи заплутані патерни, структури та сутнісні взаємозв'язки між лексичними одиницями. Це дало старт розвитку ранніх механізмів текстового синтезу.

Архітектура рекурентних нейронних мереж (РНМ) першою адекватно віддзеркаліла послідовний характер письмового мовлення. В РНМ обробка кожного наступного елемента залежить від прихованого стану, сформованого на попередньому етапі. Отже, мережа утримує в пам'яті попередній контекст.

Основні властивості RNN:

1. Динамічна пам’ять. У цьому мережевому механізмі накопичені відомості про токени, що йшли раніше, фіксуються у його внутрішньому (прихованому) стані. Завдяки цьому спроможність створювати зв'язні речення як на рівні граматики, так і на рівні сенсу, де останні слова речення логічно перегукуються з початковими.

2. Природна послідовність. РНМ (Рекурентна Нейронна Мережа) відтворює те, як текст створюється або сприймається людиною — послідовно, елемент за елементом (або токен за токеном). Завдяки цьому стало можливим моделювати вірогідність виникнення наступного слова, що заклало фундамент для якісного передбачення та формування текстового контенту.

3. Теоретична здатність обробляти довгі послідовності. Гіпотетично, рекурентні нейронні мережі (РНМ) здатні оперувати послідовностями довільної довжини, адже вони не мають прив'язки до сталої величини вхідного вікна, що є обмеженням для класичних багатошарових перцептронів (БП). Також можемо побачити на рисунку 2.4 як в нас працює RNN.

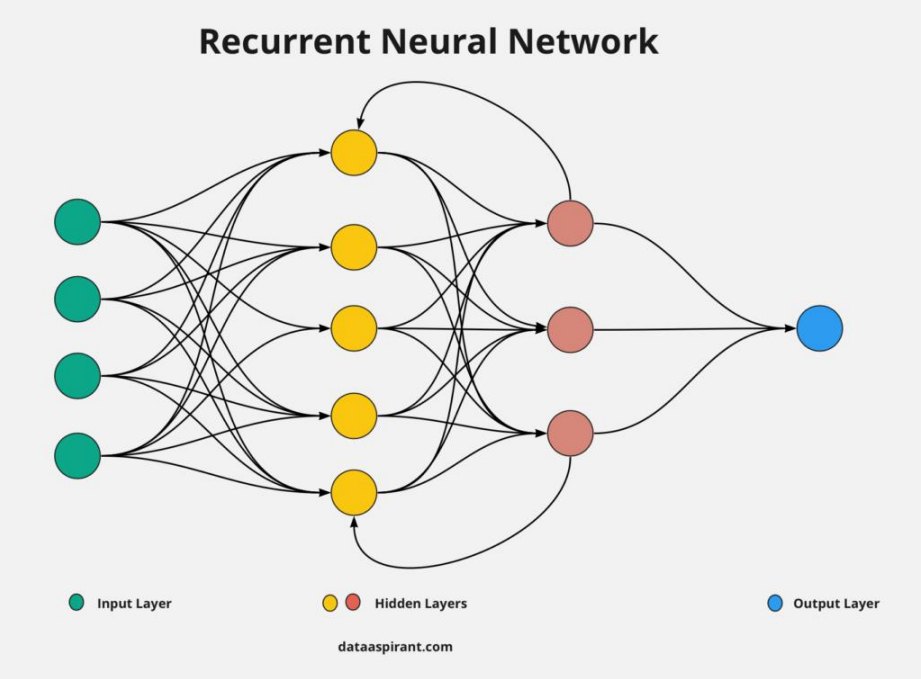


Рис. 2.4 Як працює RNN

Хоча це й виглядало як прорив, звичайні рекурентні нейронні мережі (RNN) дуже швидко зіткнулися з такими собі "засадничими" перепонами, які й зробили їх не надто придатними для опрацювання заплутаних чи надто об'ємних неформульованих даних (наприклад, деталізованих звітів технічного характеру, багатосторінкових юридичних актів чи значних масивів записів про події). Це найкритичніше обмеження, пов'язане з механізмом навчання RNN:

1. Проблема Зникаючого/Вибухаючого Градієнта

Як це працює: Процес тренування рекурентних нейронних мереж (RNN) ґрунтується на методі зворотного поширення помилки крізь часові етапи (BPTT). Це вимагає, щоб градієнти (показники напрямку зміни ваг) проходили крізь увесь часовий ланцюжок.

"Зникаючий" Градієнт: Якщо коефіцієнти множення градієнтів на кожному часовому етапі є меншими за одиницю, до того часу, як вони дістануться початкових позицій послідовності, їхня величина стрімко прямує до нуля. Наслідком є те, що ваги, які відповідають за опрацювання початкових елементів послідовності, практично не оновлюються, через що мережа втрачає здатність

враховувати віддалені у часі відомості. Збільшення довжини тексту лише погіршує продуктивність моделі.

"Вибухаючий" Градієнт: І навпаки, якщо ці коефіцієнти перевищують одиницю, вони зростають експоненційно, що призводить до повної нестійкості процесу навчання.

2. Обмеження Довжини Контексту

Моделі мали проблеми із "збереженням" інформації на значній відстані. Як приклад, коли головний член речення з'являвся на відстані п'ятдесяти слів, рекурентним нейронним мережам (RNN) було вкрай важко коректно узгодити з ним дієслово чи прикметник.

При роботі з обробкою детальних технічних документів або довжелезних журналів подій, де ключові дані можуть бути розкидані через сотні одиниць тексту, RNN демонстрували невиразні результати.

3. Низька Швидкість Обробки

Сутність проблеми: Для визначення поточного стану мережі на певному слові у послідовності, необхідно спершу завершити розрахунок стану для токена, який йому передував. Це формує пряму "рекурентну обумовленість": кожен етап процесу не може розпочатися, доки не буде отримано вихідні дані з попереднього етапу, подібно до ефекту падаючих кісток.

Наслідки: Через цю необхідність послідовності, Рекурентні Нейронні Мережі (РНМ) не були здатні повноцінно застосовувати переваги паралелізованих обчислень. Актуальні високопродуктивні графічні процесори (ГПУ) спроектовані для одночасного виконання величезної кількості операцій. Оскільки РНМ змушені були опрацьовувати лише один елемент за раз, це суттєво обмежувало їх швидкодію, роблячи їх вкрай повільними при аналізі значних обсягів інформації чи масштабних текстових даних.

Фактично, мережа змушена була працювати в "однопотоковому" режимі, можемо це побачити на рисунку 2.5.

Рис. 2.5 Базова архітектура RNN, механізм послідовної обробки

Оця обмеженість у паралельному опрацюванні була вагомим чинником, що спонукав наукову спільноту до інтенсивних пошуків серед новітніх архітектур, здатних повноцінно задіяти можливості наявного апаратного забезпечення.

Згодом на світ побачили доопрацьовані версії — LSTM та GRU. Вони продемонстрували суттєво кращі показники при роботі з об’ємнішими текстовими відрізками, адже запровадили механізми управління запам'ятовуванням. Завдяки цьому системи набули спроможності самостійно визначати, які складові контексту слід утримувати в пам'яті, а які варто відкинути, що, у свою чергу, підвищило якість генерації розгорнутих та комплексних текстів. Проте, навіть ці архітектури зберігали принцип послідовної роботи, чим стримували темпи обробки інформації та тримали залежність від послідовності появи окремих токенів.. На рисунку 2.6 можемо побачити різницю між RNN, LSTM та GRU

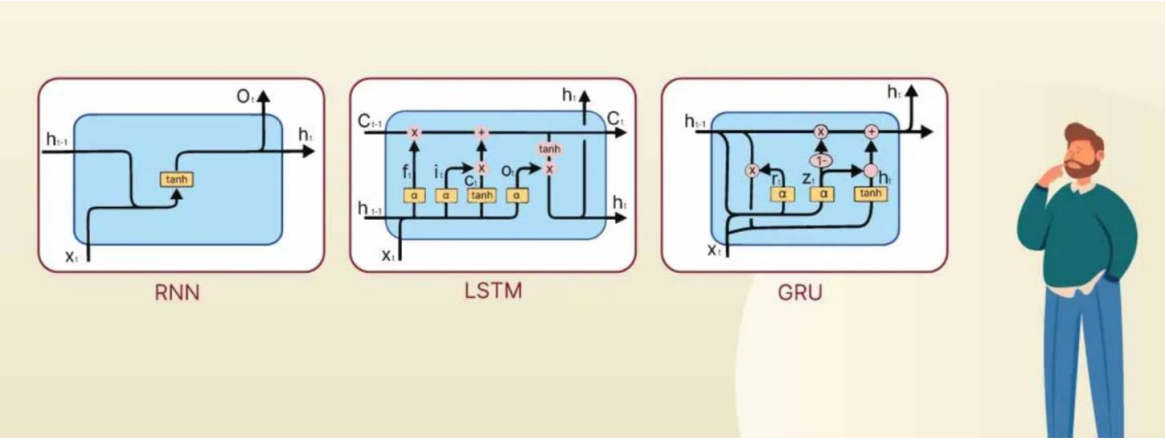


Рис. 2.6 Різниця між RNN, LSTM та GRU

Головною новацією в LSTM стало запровадження окремого каналу передачі відомостей, який іменується станом комірки. Цей елемент функціонує як своєрідний "транспортний коридор" для даних, що пронизує усю послідовність операцій із мінімальними трансформаціями, завдяки чому відомості здатні переміщуватися на значні дистанції без деградації.

Архітектура GRU, натомість, запропонувала більш стисле рішення. Вона інтегрувала певні керуючі елементи LSTM (наприклад, вентилі видалення інформації та вхідні вентилі були консолідовані в один оновлюючий вентиль), що спростило структуру моделі та пришвидшило процес її тренування. Водночас, GRU зберегла функціональність для ефективного описування взаємозв'язків, не відстаючи від LSTM у виконанні переважної частини завдань.

Найсуттєвішим результатом впровадження цих структур було подолання явища згасаючого градієнта. Завдяки принципу роботи вентилів, градієнтна інформація могла безперешкодно долати тривалі часові проміжки під час протилежного поширення помилки, не зникаючи до нульових значень. Це дозволило моделям успішно навчатися на текстах довжиною у сотні слів, зберігаючи логічний зв'язок між початком і кінцем абзацу. Попри значний успіх у якості генерації та розуміння контексту, LSTM та GRU не змогли подолати головну архітектурну ваду своїх попередників — послідовну природу обробки.

Грандіозний злам стався, коли з'явилися трансформери. Архітектура, представлена у знаковій праці "Увага — це все, що вам потрібно", принесла зовсім інший підхід до опрацювання текстової будови. Принцип самоваги (self-attention) дав змогу системі визначати значущість кожного слова відносно будь-якого іншого, незважаючи на те, наскільки далеко вони розташовані у низці. Так модель отримала здатність одночасно брати до уваги зміст із різних фрагментів тексту, зберігаючи зв'язки та не будучи прив'язаною до порядку обробки.

Модель трансформера спричинила необхідність переосмислити самі принципи створення текстів. Якщо дотепер науковці ламали собі голову над тим, як змусити модель "зберігати в пам'яті" й аналізувати раніше зазначені слова, то тепер акцент змістився на те, наскільки важливою є можливість масштабування та

опрацювання гігантських масивів текстів. Завдяки здатності паралельно обробляти відомості, трансформери почали опановувати знання значно динамічніше та набагато об'ємніших джерелах інформації. Можемо побачити різницю з іншими нейронними архітектурами на таблиці 2.2

Таблиця 2.2

Архітектура	Здатність утримувати контекст	Швидкість	Придатність до роботи з логами	Масштабованість
RNN	Низька	Низька	Обмежена	Погана
LSTM/GRU	Середня	Середня	Помірна	Нормальна
Transformer	Дуже висока	Висока	Висока	Відмінна

У царині перетворення неорганізованих відомостей у звітність, це відкрило шлях до аналізу тих файлів журналів, системних сповіщень, телеметричних даних, записів, що не мають чіткої структури, та іншої інформації, яку раніше автоматично опрацювати було майже нереально. Там, де рекурентні нейронні мережі (RNN) впоралися б лише з крихітними порціями даних, моделі-трансформери надали змогу брати до уваги довгі ланцюги логів, взаємопов'язані події, залежності між різними частинами інфраструктури та купу інших аспектів.

Трансформери, зазвичай, групують у дві головні родини: ті, що працюють за принципом кодувальник-декодувальник (Encoder–Decoder), та ті, що функціонують лише як декодувальники (Decoder-only).

Перший тип спершу перетворює вхідні дані у певне внутрішнє векторне відображення, а потім вже генерує текст на основі цього проміжного результату. Це забезпечує їм високу продуктивність у завданнях, як-от стислий переказ чи загальне підведення підсумків.

Другий тип, що є основою сучасних великих мовних моделей (LLM), вміє отримувати вказівки і реагувати на них шляхом простого продовження наданого тексту. Завдяки цьому вони демонструють свою багатогранність та придатність для

роботи з хаотичними даними. Сама будова трансформера виявилася настільки адаптивною, що на ній постав цілий спектр різних моделей. Їх, умовно, класифікують на три підгрупи, залежно від того, яку частину оригінальної конструкції вони задіяли.

Для кращого розуміння, дивіться таблицю 2.3, яка розподіляє ці архітектури відповідно до їхнього цільового призначення.

Таблиця 2.3

Тип Архітектури	Принцип Роботи	Приклади Моделей	Найкраще Застосування
Encoder-Only	Перетворює текст у вектор (embedding). "Розуміє" вхід, але не генерує текст.	BERT, RoBERTa	Класифікація логів, пошук сутностей (NER), аналіз тональності.
Decoder-Only	Передбачає наступне слово в послідовності. Бачить лише минуле.	GPT-3, GPT-4, LLaMA	Генерація звітів, чат-боти, написання коду, творчі завдання.
Encoder-Decoder	Кодує вхід у контекст, а потім генерує вихід на його основі.	T5, BART	Переклад, самаризація (стиснення) довгих текстів, перефразування.

Незважаючи на це, процес переведення інформації у словесну форму із залученням нейромереж обтяжений певними труднощами. Навіть передові архітектури здатні генерувати так звані "ілюзії" — вигадані уривки, що не ґрунтуються на первинному наборі даних. У контекстах, де достовірність виступає головною вимогою (як от у розрахунках фінансового спрямування чи у технічній документації), подібні неточності набувають вирішального значення. З огляду на це, для мінімізації подібних збоїв у нинішніх розробках застосовується методологія RAG (Generation, доповнена пошуком), яка ілюстрована на рисунку 2.7.

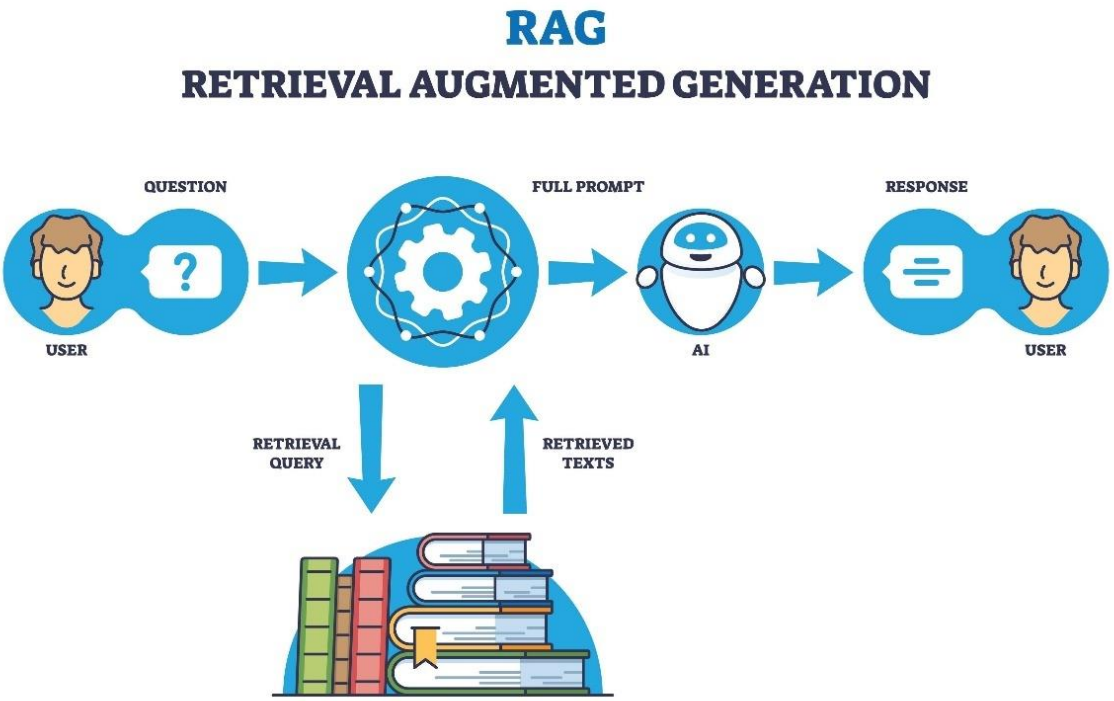


Рис. 2.7 Схема роботи RAG

Насамперед, архітектури нейронних мереж разом із трансформерами перетворилися з чистих механізмів для створення текстового контенту. По суті, вони запровадили свіжий спосіб оперування даними, де система не просто виводить послідовність слів, а ще й розуміє зміст, знаходить приховані патерни й доносить їх у формі, доступній для сприйняття людиною. Як наслідок, процес підготовки документації набув незрівнянно більшої еластичності, високої достовірності й помітно знизив потребу в ручному коригуванні.

2.3 Формалізація задачі перетворення неструктурованих даних

Перетворення неорганізованих даних у формат, здатний сприйматися людиною, належить до гаму складних процесів інформаційного аналізу. Вони вимагають поєднання різноманітних технік: від попереднього опрацювання текстового масиву та інтелектуального виявлення сутностей, до генерації зв'язного тексту і фінальної перевірки (валідації) створеного документа.

Це значно більше, аніж проста зміна форми подачі; це послідовність кроків, мета яких — гарантувати точну відповідність між початковими даними та кінцевим результатом. При цьому звіт мусить бути не лише впорядкований за структурою, але й вибудуваний логічно, лексично вивірений та такий, що відповідає прийнятим методологічним засадам.

У загальному вигляді можна вважати, що на вхід системи надходить множина фрагментів неструктурованих даних:

$$D = \{d_1, d_2, \ \ldots, d_n\}, \tag{2.1}$$

де кожен елемент d_i є незалежним текстовим або напівструктурованим об’єктом, що може мати різну природу, формат, рівень деталізації та ступінь шумності. У реальних інформаційних системах такі дані можуть включати лог-файли, журнали подій, діалоги з користувачами, конфігураційні записи, службові повідомлення, нотатки операторів, а також будь-які інші дані, які не мають стабільного формату чи схеми.

$$f_\theta : (D, C) \to R, \tag{2.2}$$

де R— вихідний текстовий документ (звіт), а θ— набір параметрів методу, які можуть включати ваги великої мовної моделі, налаштування промпту, параметри фільтрації, конфігурації алгоритмів валідації та форматування. Формалізація задачі корисна тим, що дозволяє відокремити етапи обробки та чітко визначити, які аспекти процесу можуть бути вдосконалені. Перетворення даних традиційно складається з трьох основних фаз, які утворюють єдину логічну послідовність. Далі на таблиці 2.4 можемо побачити основні типи даних, що входять до D

Таблиця 2.4

Тип даних	Характеристика	Приклади	Потенційні проблеми
Повністю неструктуровані	Не мають явної структури, містять довільний текст	Логи сервісів, повідомлення користувачів	Шум, дублікати, неоднорідність
Напівструктуровані	Мають часткову структуру, але без стабільної схеми	JSON без фіксованих полів, XML з варіативними тегами	Неповнота, неправильні формати
Структуровані фрагменти	Окремі поля з відомою структурою	Значення квот, числові параметри	Різні одиниці вимірювання

Одразу ж по завершенню офіційного окреслення набору даних, наступним критичним етапом виступає приведення його до належного формату та впорядкування. Дані, що не мають первісної структури, можуть містити позначки часу у різноманітних оформленнях, уривки технічного характеру, розділові знаки, які не мають смислового наповнення, чи службові піктограми. Завдання системи полягає не лише у виявленні головних складових, але й у відсіюванні надлишкової інформації, відновленні логічної когерентності між окремими інцидентами, знаходженні дублікатів, об'єднанні подій за їхньою категорією та визначенні пріоритетності. Підсумком цього етапу стає створення внутрішнього, впорядкованого формату даних:

$$D \rightarrow S \tag{2.3}$$

де S— таблиця або інша структура, яка робить дані придатними для машинного опрацювання. Далі на рисунку 2.8 узагальнення.

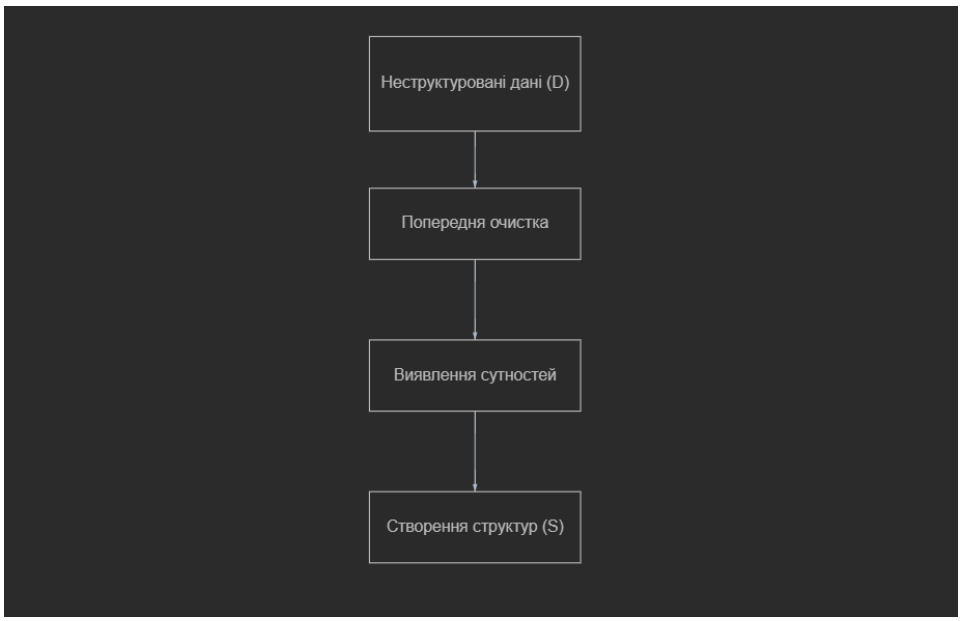


Рис. 2.8 Узагальнена схема структуризації даних

Другий етап — генерація первинної текстової чернетки, тобто перехід від структурованої форми до природної мови:

$$R_{draft} = g_{\varphi}(S, C) \tag{2.4}$$

На цьому етапі модель, найчастіше архітектури трансформер або велика мовна модель (LLM), перетворює впорядкований масив даних на зв’язний текстовий матеріал. Апаратне забезпечення здійснює семантичний аналіз описаних подій, встановлює логічні ланцюжки взаємодії (причина-наслідок), генерує підсумкові положення та експертні оцінки, коригуючи їх відповідно до необхідного формату репортажу. Отриманий варіант може мати високий рівень виконання, проте, зазвичай, він містить певні помилки, упущення чи вигадані факти («галюцинації»), через що він не може фігурувати як фінальна версія документації. Третій етап — валідація і постобробка, яка дозволяє сформувати кінцевий звіт:

$$R = h(R_{draft}, S, C) \tag{2.5}$$

Кінцева обробка даних має на меті довести текст до ідеальної відповідності первинним намірам, усунути будь-які розбіжності та привести стилістику документа згідно із заданими користувачем критеріями. На цьому плані можуть

використовуватися як програмні підходи, так і додаткові інструкції, надані самій моделі (повторне підказування). Фінальним продуктом є документ, повністю готовий до презентації вищому керівництву, клієнту чи технічним експертам. Загальний графік того, як відбувається трансформація даних, представлено на рисунку 2.9.

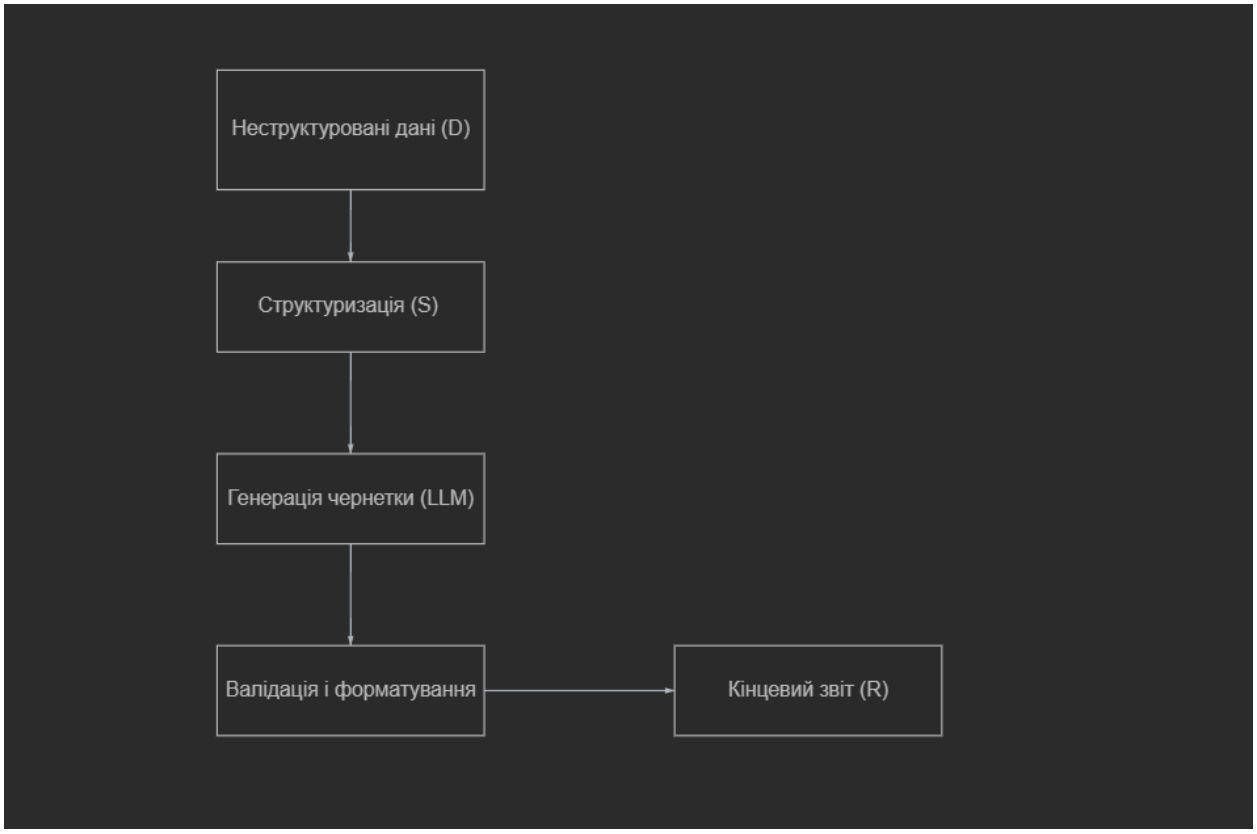


Рис. 2.9 Загальна логічна схема перетворення даних

Вдосконалення процесу трансформації може бути реалізоване на любом із зазначених етапів. Приміром, якщо ми покращимо алгоритми, що відповідають за структурування, це матиме наслідком зниження завад (шуму) та покращення якості початкових даних, що, своєю чергою, безпосередньо позначиться на рівні точності кінцевої генерації. Калібрування запитів (промптів) та параметрів великої мовної моделі (LLM) сприяє зменшенню кількості хибних результатів на етапі першого чернеткового варіанту. Водночас, відточена подальша обробка забезпечує усунення будь-яких логічних нестиковок та підтверджує узгодженість із внутрішніми нормами подання звітності.

Формальна модель f_θ, розкладена на три компоненти — $D \to S$, $S \to R_{\text{draft}}$ та $R_{\text{draft}} \to R$— дозволяє створювати системи, які є масштабованими, керованими та прогнозованими. Завдяки такій декомпозиції можна гнучко замінювати окремі компоненти, не змінюючи архітектуру всієї системи. Це надає можливість адаптувати метод під різні типи звітів, сфери застосування та вимоги до точності.

Узагальнюючи викладене, формалізація задачі перетворення неструктурованих даних дає можливість впорядкувати процес побудови звіту і розділити його на чіткі етапи: структуризацію інформації, генерацію текстової чернетки та її подальшу валідацію. Такий підхід дозволяє не лише підвищити точність і передбачуваність результатів, а й забезпечити масштабованість системи при зростанні обсягів даних.

Впровадження сучасних мовних моделей значно розширює можливості автоматизованого аналізу, однак їх ефективність напряму залежить від якості вхідної підготовки та коректності контексту. Саме тому поєднання алгоритмічних методів, проміжних структур представлення даних і засобів перевірки створює основу для надійних і гнучких систем генерації звітів.

Таким чином, формальна модель перетворення даних виступає ключовим концептуальним елементом, який забезпечує цілісність процесу та відкриває можливості для подальшого удосконалення методів автоматизованої звітності.

2.4 Порівняльний аналіз існуючих рішень для генерації звітів/квот

Сучасні інформаційні системи, що займаються побудовою текстових звітів або автоматизованим формуванням квот, охоплюють широкий спектр технологій — від класичних засобів бізнес-аналітики до високорівневих мовних моделей. Кожен із цих підходів має власну історію розвитку, набір типових застосувань, обмеження та рівень адаптивності до неструктурованих даних. На практиці вони часто співіснують у межах великих організаційних екосистем, доповнюючи одна одного, але рідко формують єдину узгоджену систему, здатну автоматично перетворювати необроблені текстові логи на узгоджені та точні звіти.

Сучасні системи бізнес-аналітики (BI) схильні фокусуватися, в основному, на опрацюванні виключно впорядкованих даних, які вже зазнали необхідного попереднього очищення та зведення. Їхня перевага полягає у вмінні генерувати інформативні дашборди, графічні зображення та звітні таблиці, що дають уявлення про поточний стан будь-якої системи чи операційних бізнес-ланцюгів. Однак поза межами цього впорядкованого контуру, подібні апарати практично втрачають свою дієвість: вони не здатні розшифровувати обсяги послідовних подій, неврегульовані текстові дані або уривки заплутаних технічних журналів. Якщо ж на систему надходить потік необроблених записів, BI-рішення виявляються фактично неспроможними допомогти, оскільки їхня архітектура роботи з даними міцно закріплена за заздалегідь спроектованими табличними структурами та схемами.

На протилежному ж краю спектру розташовані системи, що займаються Генерацією Природної Мови (Natural Language Generation), головна мета яких — автоматизувати генерування стислих текстових викладок. Такі засоби оперують комбінацією заздалегідь визначених шаблонів та відносно простих алгоритмів машинного навчання. Головною слабкістю цього підходу є їхня обмежена гнучкість. Варто лише змінити сферу застосування або ж якщо вхідні дані матимуть нетипове форматування, ці рішення вимагають значних втручань: доведення до ладу шаблонів та ревізії закладеної логіки, що суттєво ускладнює та уповільнює їхнє поширення.

Виникнення крупних мовних моделей відкрило абсолютно новий спосіб мислення. Ці LLM здатні обробляти навіть неорганізований потік тексту, який не був попередньо структурований, і на його базі формувати логічні тлумачення, узагальнення та підсумкові документи. Вони вміють оперувати настановами, висловленими звичайною мовою, інтегрувати розрізнені фрагменти вхідної інформації та брати до уваги загальний контекст поставленого завдання.

Саме ці характеристики вивели архітектури на основі трансформерів на провідні позиції у сучасних системах автоматичного створення звітів. Ризики, пов'язані з їхньою схильністю вигадувати неправдиві або недостовірні частини

тексту, а також нездатність гарантувати точність числових даних, роблять їх непридатними для застосування у звітності, критично важливій для певних галузей.

Якщо ж розглянути наявні підходи у більш загальному плані, то стає очевидним: основна складність полягає не у виборі окремих технологічних вузлів, а у відсутності всеосяжного робочого процесу, який би об’єднав усі фази опрацювання даних в одну злагоджену систему. Зазвичай процес упорядкування здійснюється нашвидкуруч, ігноруючи семантичні зв'язки між подіями, або ж текстова генерація відбувається без прямого зв'язку з першоджерелом, через що модель не завжди коректно відображає ключові метрики.

Щоб відобразити відмінності між типами систем, можна подати порівняльну таблицю 2.5, яка підкреслює ключові характеристики різних підходів

Таблиця 2.5

Критерій	BI-системи	NLG-платформи	Системи на основі LLM
Тип даних	Переважно структуровані	Частково структуровані	Неструктуровані, змішані
Гнучкість	Низька	Середня	Висока
Якість текстів	Обмежена, формальна	Стабільна, але передбачувана	Висока, варіативна
Ризик помилок	Низький	Середній	Підвищений без валідації
Необхідність ручного налаштування	Висока	Висока	Помірна
Можливість адаптації до нових доменів	Слабка	Середня	Дуже висока

Як було показано, головна проблема ринку рішень для автоматичного звітування полягає не в недостатності окремих технологічних компонентів, а у відсутності цілісного процесу, який би поєднував різні етапи роботи з даними в єдину узгоджену систему. На практиці, розробники часто стикаються з дилемою:

1. Зробити ставку на BI/Генерування природної мови (NLG): Це забезпечить достовірність та стабільність при опрацюванні впорядкованих даних, проте цілковито унеможливить аналіз контекстно-змістовних відомостей, що містяться у журналах подій чи неструктурованих текстах.

2. Застосувати Великі мовні моделі (LLM): Це дасть змогу досягти гнучкості та високих інтерпретаційних спроможностей, але водночас несе неприпустимий ризик внесення суттєвих неточностей та хибних числових результатів у фінальних документах.

Для наочності можна подати й блок-схему яку можно побачити на рисунку 2.10, що відображає різницю у роботі традиційних рішень і сучасних мовних моделей

Рис. 2.10 Схематичне порівняння підходів

Отож, доводиться констатувати, що нинішній ринок продуктів для автоматизованого складання звітів розколений на три основні підходи. Кожен із них має свої сильні сторони у певних сферах, проте жоден не гарантує цілісного, безшовного виконання всього циклу.

3 РОЗРОБКА МЕТОДУ ТА ПРОГРАМНОЇ СИСТЕМИ

Попередні розділи заклали необхідну теоретичну та аналітичну базу для вирішення поставленої наукової задачі, включаючи глибокий огляд стану проблеми та детальний аналіз існуючих підходів у галузі обробки даних за допомогою нейронних мереж. Незважаючи на значні успіхи, досягнуті сучасними технологіями, залишаються ключові виклики та невирішені питання, які вимагають розробки принципово нових та вдосконалених рішень. Саме на цих критичних аспектах зосереджено увагу у даному розділі. Цей етап дослідницької роботи є найважливішим кроком у переході від концептуальних ідей до практично реалізованої системи. Основна мета розділу полягає у систематичній розробці, детальному описі та обґрунтуванні архітектури нового методу та супутньої програмної системи, що базується на потужностях архітектури трансформерів.

Розділ починається з чіткого формулювання Постановки задачі, де будуть визначені конкретні цілі, очікувані результати та функціональні й нефункціональні вимоги до розроблюваного методу. Далі буде представлена Загальна архітектура системи, яка ілюструє, як саме компоненти на основі трансформерів будуть інтегровані для досягнення максимальної ефективності та масштабованості. Ключовим моментом є представлення Математичної моделі та Алгоритму, які слугуватимуть фундаментом для точного та надійного перетворення вхідних даних.

Крім того, значна увага приділяється практичній інженерній складовій: буде детально описано програмні засоби, технологічний стек та структура програмного забезпечення. Це забезпечить прозорість процесу реалізації та можливість відтворення результатів. Завершальні підрозділи присвячені зручності використання системи — опису інтерфейсу користувача та сценаріїв його взаємодії, а також методології проведення Експериментальних досліджень, що дозволить об’єктивно оцінити якість, швидкодію та переваги розробленого методу

порівняно з існуючими аналогами. Таким чином, цей розділ забезпечує повний цикл від наукової ідеї до функціонального продукту.

3.1 Постановка задачі та вимоги до методу

У поточних умовах розвитку інформаційних технологій спостерігається неухильне збільшення масивів даних, що утворюються внаслідок функціонування різноманітних технологічних платформ, сервісів, мережевих інфраструктур, фінансових комплексів та інструментарію, що використовується у бізнесі. Нерідко ця інформація постає у формі текстових журналів (логів), службових сповіщень, незв'язаних нотаток, зроблених персоналом, технічної документації довільної структури чи файлів, що мають лише часткову впорядкованість. Усередині цих даних закладено чимало цінного змісту — від сигналів про збої до показників ефективності використання активів та виявлених відхилень (аномалій). Проте, через брак сталої структури, ці обсяги даних практично непридатні для прямого аналітичного опрацювання. Переважна більшість компаній змушена спрямовувати суттєві людські ресурси на ручне опрацювання, впорядкування та розшифрування відповідних фрагментів, що, своєю чергою, гальмує процес ухвалення рішень та затягує час, необхідний для адекватної відповіді на інциденти.

Як ми ретельно розглядали у попередніх частинах цієї кваліфікаційної роботи, нинішня фаза еволюції інформаційних технологій відзначається шаленим, нестримним збільшенням масивів даних, що постають унаслідок роботи багатокомпонентних програмно-апаратних систем. Переважна більшість цих відомостей існує у формі, що не має чіткої структури, або ж має її вельми умовно, що спричиняє значні труднощі для швидкого їхнього дослідження та застосування у механізмах ухвалення рішень. Незважаючи на існування різноманітного арсеналу засобів для бізнес-інтелекту (BI) та традиційних підходів до опрацювання текстової інформації, спостерігається явний дефіцит дієвих методів автоматичного перетворення «необроблених» технічних відомостей у зрозумілі, цілісні текстові

репорти, які можна легко сприймати людині без потреби залучати вузьких спеціалістів із технічних галузей.

Складність дилеми посилюється тим фактом, що наявні методології, які спираються на суворі регламенти (базовані на правилах) чи незмінні зразки, доводять свою неспроможність у ситуаціях, коли формати даних постійно еволюціонують, а вхідні потоки інформації мають значну міру невпорядкованості. Водночас, сучасні, прогресивні техніки, що використовують потужність великих мовних моделей (ВММ) та трансформерних архітектур, незважаючи на їхню виняткову гнучкість та здатність до пристосування, несуть із собою загрозу створення невідповідних дійсності даних (так званих «вигадок»), що є абсолютно неприпустимим, особливо коли йдеться про формування звітів, які вимагають відповідальності, зокрема тих, що стосуються фінансових оцінок, лімітів ресурсів чи критичних подій у сфері безпеки..

З огляду на сказане раніше, вимальовується гостра необхідність, як теоретична, так і практична, у створенні та впровадженні модернізованого підходу. Цей підхід повинен мати здатність елегантно інтегрувати високий інтелектуальний потенціал новітніх архітектур нейронних мереж із жорсткими критеріями щодо достовірності, всеохопності та підтверджуваності результатів. Задум полягає в тому, аби сформувати своєрідний "сполучний елемент" між неструктурованим обсягом даних машинних журналів та організованою сферою людського аналізу.

Головним завданням при проєктуванні є формування цілісного підходу до автоматичного переведення даних, що не мають чіткої структури або є частково впорядкованими, у формат, зрозумілий для сприйняття людиною. Цей підхід, спираючись на застосування моделі трансформерів, має на меті виведення логічно пов'язаних, організованих письмових звітів, при цьому суттєво знижуючи ризик викривлення первинної інформації. Досягнення цієї мети передбачає вирішення низки взаємопов'язаних завдань, які визначають сутність розроблюваного методу:

1. **Забезпечення універсальності входу:** Метод повинен бути інваріантним до джерела походження даних, тобто здатен ефективно обробляти як

системні журнали (логи), так і вільний текст заявок, «сирі» звіти моніторингу, дані у форматах JSON/XML та інформацію про квотування ресурсів.

2. Структурна трансформація: Реалізація механізмів, що дозволяють трансформувати розрізнені фрагменти інформації у цілісний документ із заданою користувачем структурою, включаючи вступну частину, основний зміст, аналітичні висновки та рекомендації.

3. Фактологічна валідація: Впровадження алгоритмів, спрямованих на контроль цілісності числових та фактичних даних під час їх проходження через нейромережеву модель, з метою унеможливлення викривлення критично важливих показників (наприклад, значень використаних квот або часових міток подій).

4. Семантична узгодженість: Забезпечення високого рівня зв'язності тексту, стилістичної відповідності та повноти викладу, щоб згенерований звіт не виглядав як набір випадкових речень, а являв собою повноцінний аналітичний документ.

Деталізація функціональних вимог до методу для того, щоб розроблюваний метод відповідав реальним потребам предметної області та міг бути ефективно впроваджений у практику, до нього висувається ряд чітких функціональних вимог, кожна з яких відіграє критичну роль у загальній архітектурі рішення.

Мультимодальна підтримка джерел вхідних даних. Система не повинна обмежуватися роботою з одним конкретним типом файлів. Необхідно реалізувати підтримку широкого спектра джерел: від класичних текстових лог-файлів (.log, .txt), які містять хронологію подій, до структурованих дампів баз даних та файлів обміну даними (.csv, .json, .xml). Крім того, метод повинен коректно інтерпретувати змішані дані, де технічна інформація переплітається з коментарями користувачів або операторів, написаними природною мовою.

Гнучке налаштування шаблонів та структури звітності. Однією з ключових вимог є здатність адаптуватися до потреб конкретного користувача або бізнес-процесу. Метод повинен надавати інструментарій для визначення бажаної структури вихідного документа. Це означає можливість задання переліку обов'язкових розділів (наприклад, «Огляд інцидентів», «Статус квот»,

«Висновки»), визначення порядку їх слідування та ступеня деталізації. Особливу увагу слід приділити блокам, що відображають кількісні показники (квоти), які повинні інтегруватися в текст органічно, але з математичною точністю.

Багатомовність генерації (Multilingual Generation Capability). В умовах глобалізації та роботи у багатонаціональних командах критично важливою є здатність методу формувати звіти мовою, яка є найбільш зручною для кінцевого споживача інформації. Система повинна підтримувати генерацію звіту щонайменше трьома мовами (наприклад, українською, англійською), забезпечуючи при цьому коректний переклад специфічної термінології та збереження семантичних відтінків вихідного повідомлення.

Варіативність та інтерактивність результату. Враховуючи ймовірнісний характер роботи трансформерних моделей, метод повинен передбачати можливість генерації декількох варіантів одного й того ж звіту або його окремих частин (drafting options). Це дозволить користувачеві обрати найбільш вдале формулювання або стиль викладу. Також має бути передбачена можливість ручного коригування проміжних результатів із подальшим врахуванням цих правок системою, що реалізує концепцію «human-in-the-loop» (людина в контурі управління).

Контроль повноти та узгодженості. Метод повинен включати механізми перевірки того, що всі ключові події, зафіксовані у вхідних даних, знайшли своє відображення у фінальному звіті. Неприпустимою є ситуація, коли система «забуває» про критичні помилки або перевищення лімітів лише через особливості роботи механізму уваги (attention mechanism) трансформера.

Окрім функціоналу, життєздатність розроблюваної системи визначається набором нефункціональних характеристик та обмеження, які гарантують її надійність, продуктивність та безпеку експлуатації.

Масштабованість (Scalability). Метод повинен бути розрахований на обробку значних масивів даних. Лог-файли корпоративних систем можуть досягати розмірів у гігабайти за лічені години. Тому алгоритм повинен передбачати ефективні стратегії розбиття даних на фрагменти (chunking), їх паралельну обробку

або послідовну агрегацію, щоб забезпечити генерацію звіту за прийнятний час, не вимагаючи при цьому надмірних обчислювальних ресурсів, які перевищують можливості стандартного серверного обладнання.

Розширюваність та модульність (Extensibility). Архітектура методу повинна бути побудована за модульним принципом, що дозволить у майбутньому легко інтегрувати нові типи вхідних даних або замінювати базові моделі трансформерів на більш досконалі версії без необхідності повної переробки всієї системи. Це забезпечить довготривалий життєвий цикл розробки та її актуальність в умовах швидкого старіння технологій ШІ.

Інформаційна безпека та конфіденційність (Security & Privacy). Оскільки вхідні дані можуть містити чутливу інформацію (IP-адреси, імена користувачів, внутрішні шляхи до файлів, комерційну таємницю), метод повинен передбачати етапи попередньої обробки, спрямовані на деперсоналізацію або маскування таких даних перед їх передачею на вхід нейронної мережі (особливо якщо використовуються зовнішні API). Система має гарантувати, що процес генерації звіту не стане каналом витоку конфіденційної інформації.

Надійність та відмовостійкість (Reliability). Метод повинен коректно обробляти виключні ситуації, такі як пошкоджені вхідні файли, некоректне кодування символів або тимчасова недоступність обчислювальних ресурсів. У разі виникнення помилки система повинна надавати зрозумілу діагностичну інформацію, а не видавати некоректний звіт.

Таким чином, задача розробки методу перетворення неструктурованих даних у людиночитані звіти включає як технічні, так і логічно-аналітичні аспекти. Запропонований метод має забезпечувати баланс між гнучкістю сучасних трансформерних моделей і жорсткими вимогами до точності, повноти й структурованості текстів. Саме комплексний характер вимог визначає необхідність поєднання попередньої обробки даних, продуманої інженерії промптів, застосування трансформерів і постобробки результату, що в сукупності забезпечує створення надійного та контрольованого механізму генерації звітів.

3.2 Загальна архітектура вдосконаленого методу на основі трансформерів

Проєктований у межах дипломної роботи метод перетворення неструктурованих даних на людиночитані звіти базується на принципах модульності, багаторівневої обробки та контрольованої взаємодії з мовними моделями типу трансформер. Така архітектура дозволяє врахувати природні обмеження LLM, зокрема схильність до появи вигаданих фактів, а також забезпечити стабільність результату незалежно від природи початкових джерел. Відповідно до загальної логіки системи, метод структуровано у вигляді конвеєра, у якому дані проходять низку послідовних етапів трансформації — від первинного збору до остаточної генерації та валідації.

Архітектура методу має не лише операційне, а й концептуальне значення: вона задає філософію взаємодії між машинно-генерованими елементами та формальними правилами аналітичного представлення інформації. У сучасних умовах, коли обсяги логів та потоки подій є надзвичайно великими, а час реакції на інциденти критично обмежений, навіть найменша помилка в інтерпретації даних може привести до неправильних управлінських рішень. Саме тому вдосконалений метод передбачає суворе відокремлення етапів, де виконується машинна творчість, і етапів, де здійснюється перевірка фактичної точності. Загальна архітектура методу складається з шести основних модулів:

1. Модуль приймання та попередньої обробки даних.
2. Модуль структуризації та агрегації.
3. Модуль побудови промпта.
4. Модуль генерації звіту на основі трансформера.
5. Модуль валідації та постобробки.
6. Модуль представлення результатів.

Кожен модуль має внутрішню структуру та реалізує окремий логічний етап загального процесу. У подальшому цей підрозділ детально описує їхню роль,

функціональність, вимоги, а також взаємозв’язки. На рисунку 3.1 можемо побачити схему методу обробки неструктурованих даних

Рис. 3.1 Схема методу обробки неструктурованих даних

Почнем ми з модулю приймання та попередньої обробки даних (Data Ingestion & Preprocessing) цей модуль є «вхідними воротами» системи. Його критична важливість обумовлена відомим принципом аналізу даних «Garbage In — Garbage Out» (сміття на вході — сміття на виході). Оскільки вхідні дані можуть надходити з різнорідних джерел, модуль повинен забезпечити уніфікацію інформаційних потоків перед їх подальшою обробкою. Функціонал модуля включає наступні процеси:

1. Агрегація різнорідних джерел. Система реалізує інтерфейси для завантаження даних різних типів як лог-файли та журнали подій: текстові потоки, що генеруються серверами, додатками або мережевим обладнанням. Неструктуровані текстові документи: службові записки, описи інцидентів, тікети з систем підтримки (Service Desk). Структуровані та напівструктуровані таблиці: CSV-файли, вивантаження з баз даних, JSON-об'єкти, що містять інформацію про квоти, білінг або конфігурації.

2. Нормалізація та очищення. На цьому етапі відбувається технічна підготовка тексту. Виконуються процедури приведення кодування до єдиного стандарту (зазвичай UTF-8), видалення недрукованих символів, артефактів форматування (наприклад, ANSI-кодів кольорів у консольних логах) та HTML-тегів.

3. Фільтрація шумів та дедуплікація. Технічні логи часто містять надлишкову інформацію (наприклад, stack traces, що повторюються сотні разів). Модуль ідентифікує та видаляє дубльовані записи, залишаючи лише унікальні події з лічильником їх повторень. Це дозволяє суттєво зменшити обсяг вхідного контексту, що є критичним для ефективної роботи трансформера, обмеженого розміром контекстного вікна.

Далі в нас модуль структуризації та агрегації (Structuring & Aggregation Module) після очищення дані все ще залишаються набором розрізнених записів. Мета цього модуля — перетворити низькорівневі дані на набір значущих фактів та метрик. Тут застосовуються детерміновані алгоритми (Regular Expressions, парсери), які гарантують точність вилучення інформації. Ключові етапи роботи модуля:

1. Інтелектуальний парсинг логів. Використовуючи набір правил та евристик, модуль виділяє з неструктурованого тексту ключові сутності: часові мітки (timestamp), рівні критичності подій (INFO, WARNING, ERROR), коди помилок, назви сервісів або ідентифікатори користувачів.

2. Побудова агрегованих показників. Замість того, щоб передавати в LLM тисячу рядків про помилку з’єднання, модуль обчислює статистику: «За період X зафіксовано 1543 помилки з’єднання, пік навантаження припав на 14:00». Також відбувається розрахунок використання ресурсів: порівняння фактичних значень із заданими квотами, визначення відсоткового співвідношення, фіксація екстремумів (максимальних та мінімальних значень).

3. Формування уніфікованої внутрішньої моделі. Усі отримані дані зберігаються у проміжному форматі (Internal Representation), який являє собою

структурований об’єкт (наприклад, складний словник або граф знань). Саме ця модель стає єдиним джерелом правди для наступних етапів.

Продовжуємо і переходимо до модулю побудови промпта (Prompt Construction Module) якість роботи трансформера напряму залежить від якості вхідної інструкції. Цей модуль відповідає за динамічну генерацію промпта (context injection), який поєднує завдання користувача з підготовленими даними. Це не просто текстовий рядок, а складна інженерна конструкція. Процес формування промпта включає:

1. Визначення ролі та інструкцій. Модуль додає системні інструкції (System Prompt), які задають поведінку моделі (наприклад: «Ти — провідний системний аналітик. Твоя мета — написати чіткий звіт про використання хмарних квот»). Задається бажаний тон (tone of voice), формат заголовків та мова виводу.

2. Опис предметної області. Для кращого розуміння контексту в промпт додається короткий опис домену. Наприклад, пояснення специфічних термінів або абревіатур, що зустрічаються у вхідних даних.

3. Ін’єкція структурованих даних. Агреговані показники з попереднього модуля трансформуються у компактний текстовий вигляд (Markdown-таблиці, марковані списки, JSON-фрагменти) і вставляються в тіло промпта. Важливо, щоб ці дані були подані максимально лаконічно для економії токенів, але достатньо детально для збереження змісту.

Тепер йдемо до наступного модулю генерації звіту на основі трансформера (Transformer-based Generation Module) - це ядро системи, де відбувається безпосередня магія перетворення структурованих фактів у зв’язний текст природною мовою. Модуль взаємодіє з API великої мовної моделі (LLM). Цей модуль також має особливості функціонування:

1. Виклик LLM та керування параметрами. Модуль відправляє сформований промпт на вхід нейромережі. При цьому налаштовуються гіперпараметри генерації, такі як *Temperature* (ступінь креативності: для звітів вона встановлюється низькою, близькою до 0, для забезпечення детермінованості) та *Top-P* (для контролю різноманітності лексики).

2. Отримання чернетки (Drafting). Трансформер генерує текст звіту, інтерпретуючи передані йому статистичні дані, пояснюючи причинно-наслідкові зв’язки та формуючи висновки.

3. Варіативність генерації. За необхідності модуль може ініціювати паралельну генерацію декількох варіантів окремих розділів (наприклад, «Висновки»), щоб надати користувачеві можливість вибору найкращого формулювання.

Далі в нас йде модуль валідації та постобробки (Validation & Post-processing Module) - це «запобіжник» системи, який вирішує головну проблему сучасних генеративних моделей — проблему галюцинацій. Цей модуль не дозволяє «сирому» виводу трансформера потрапити до кінцевого користувача без перевірки.

У модуля є такі функції:

1. Фактологічна звірка (Fact-Checking). Алгоритм сканує згенерований текст на наявність числових значень (квот, сум, дат) і порівнює їх з тими, що зберігаються в уніфікованій внутрішній моделі (Модуль 2). Якщо модель написала «використано 80% квоти», а в базі даних значиться «45%», система автоматично маркує цей фрагмент як помилковий або намагається виправити його.

2. Структурна валідація. Перевіряється наявність усіх обов’язкових розділів згідно із заданим шаблоном. Якщо трансформер пропустив розділ «Рекомендації», модуль може ініціювати повторний запит на догенерацію відсутньої частини.

3. Нормалізація стилю. Виконуються скрипти автокорекції для стандартизації форматів (наприклад, заміна дати «2024/10/01» на «1 жовтня 2024 року»), виправлення типографіки та форматування списків.

Переходимо до останнього модулю, а точніше до модулю представлення результатів (Result Presentation Module), який забезпечує взаємодію людини з результатами роботи алгоритму. Також в цього модуля є різні можливості наприклад:

1. Інтерфейс користувача. Відображення згенерованого звіту у зручному веб-інтерфейсі з підсвічуванням ключових метрик.

2. Експорт документів. Конвертація розміченого тексту у популярні офісні формати (DOCX, PDF) із збереженням стилів, таблиць та корпоративного брендингу.

3. Human-in-the-loop редагування. Надання користувачеві інструментів для ручного редагування тексту. Важливо, що система може запам’ятовувати правки користувача для донавчання або коригування промптів у майбутніх ітераціях.

Запропонована архітектура вдосконаленого методу перетворення неструктурованих даних являє собою комплексну багаторівневу систему, що базується на принципах модульності, конвеєрної обробки та гібридної верифікації інформації. На відміну від тривіальних рішень, які покладаються на пряму взаємодію з великими мовними моделями, розроблений підхід впроваджує проміжні шари абстракції — модулі попередньої нормалізації, структурованої агрегації та пост-фактум валідації. Це дозволяє нівелювати ключові недоліки сучасних генеративних трансформерів, такі як схильність до фактологічних галюцинацій та нестабільність формату виводу.

Ключовою особливістю архітектури є чітке розмежування зон відповідальності: детерміновані алгоритми відповідають за точність числових даних та структуру, тоді як імовірнісні нейромережеві моделі використовуються виключно для семантичної інтерпретації та лінгвістичного оформлення результату. Такий симбіоз забезпечує високу адаптивність системи до зміни форматів вхідних даних без втрати надійності, що є критично важливим для корпоративних систем звітності.

Реалізація даної архітектури у вигляді набору взаємопов’язаних, але незалежних сервісів створює передумови для гнучкого масштабування системи. Вона дозволяє локально оновлювати окремі компоненти (наприклад, змінювати версію LLM або додавати нові парсери логів), не порушуючи загальної логіки функціонування. Таким чином, спроєктована архітектура не лише вирішує поточну задачу генерації людиночитаних звітів, а й закладає фундамент для побудови стійких аналітичних систем нового покоління.

3.3 Математична модель та алгоритм перетворення даних

Розробка ефективного програмного забезпечення для обробки неструктурованої інформації неможлива без побудови чіткої формальної моделі. Математичний опис процесів перетворення даних дозволяє абстрагуватися від конкретних деталей реалізації (мов програмування, бібліотек) і зосередитися на логіці інформаційних потоків, умовах коректності алгоритмів та критеріях завершення процесу генерації.

У даному підрозділі наводиться теоретико-множинний опис запропонованого методу, формалізуються поняття вхідного інформаційного простору, операторів трансформації та процедури ітераційної валідації результату.

Почнемо з формалізація простору вхідних даних, нехай D – множина вхідних «сирих» даних, що надходять до системи. Враховуючи гетерогенну природу джерел, цю множину можна представити як об’єднання підмножин різної природи:

$$D = D_{logs} \cup D_{text} \cup D_{semi} \cup D_{meta}, \quad (3.1)$$

де:

D_{logs} – множина рядкових записів системних журналів (логів), де кожен елемент $d \in D_{logs}$ є кортежем (t, l, m), що містить часову мітку t, рівень події l та повідомлення m.

D_{text} – множина неструктурованих текстових фрагментів (коментарі, описи заявок), що складаються з послідовностей символів природної мови.

D_{semi} – підмножина напівструктурованих даних (JSON, XML), які мають часткову схему.

D_{meta} – метадані, що описують контекст (наприклад, ідентифікатори серверів, імена користувачів).

Головною проблемою множини D є її висока ентропія та наявність шуму. Тому безпосереднє відображення $D \rightarrow R$ (де R – цільовий звіт) за допомогою мовної моделі є ненадійним. Необхідним є введення проміжного простору станів.

Далі в нас оператор попередньої обробки та структурування. Визначимо S як структуровану множину сутностей та показників, що є «очищеною» проєкцією вхідних даних. Перехід від «сирих» даних до структурованих описується оператором попередньої обробки T:

$$S = T(D) \tag{3.2}$$

З математичної точки зору, оператор T є композицією трьох функцій відображення:

$$\mathrm{T} = \mathrm{T}_{\mathrm{parse}} \circ \mathrm{T}_{\mathrm{filter}} \circ \mathrm{T}_{\mathrm{agg}} \tag{3.3}$$

де:

$\mathrm{T}_{\mathrm{parse}}$ – функція нормалізації, яка здійснює бієктивне відображення вхідних рядків у простір уніфікованого кодування (UTF-8), видаляє недруковані символи та приводить формати дат до єдиного стандарту (ISO 8601).

$\mathrm{T}_{\mathrm{filter}}$ – функція фільтрації, яка зменшує потужність множини даних, відкидаючи елементи, що не несуть інформаційного навантаження (шум, дублікати).

$\mathrm{T}_{\mathrm{agg}}$ – функція агрегації, яка трансформує послідовність подій у множину скалярних та векторних величин

Результатом застосування оператора T є множина S, яка може бути представлена у вигляді асоціативного масиву або графу знань, готового для подальшої серіалізації.

Далі йде модель побудови контекстного запиту (Промпта) для керування генеративною моделлю необхідно трансформувати структуровані дані S у текстовий вектор (промпт). Визначимо P як вектор параметрів генерації звіту, що задається користувачем:

$$\mathrm{P} = \{\mathrm{p}_{\mathrm{lang}}, \mathrm{p}_{\mathrm{schema}}, \mathrm{p}_{\mathrm{style}}, \mathrm{p}_{\mathrm{detail}}\} \tag{3.4}$$

де:

$\mathrm{p}_{\mathrm{lang}}$ – мова генерації.

$\mathrm{p}_{\mathrm{schema}}$ – структура шаблону.

$\mathrm{p}_{\mathrm{style}}$ – стиль подання.

p_{detail} – рівень деталізації.

Функція побудови промпта P_{build} здійснює відображення простору даних та параметрів у простір токенів:

$$\text{prompt} = P_{build}(S, P) \tag{3.5}$$

Структурно змінна prompt являє собою впорядковану конкатенацію семантичних блоків. Формально це можна записати як суму рядкових векторів:

$$\text{prompt} = [\text{Instr}] + [\text{Schema}] + [\text{Data}] + [\text{Examples}] \tag{3.6}$$

де кожен доданок виконує специфічну роль у налаштуванні механізму уваги (Self-Attention) трансформера:

Instr — глобальні настанови для LLM (мова, стиль, ціль звіту).

Schema — опис необхідної структури документа.

Data — формалізоване стислим чином множини Sу вигляді списків, таблиць, подій.

Examples — приклади правильного формату (опційний компонент), що покращує стабільність генерації.

Далі в нас генеративна модель та формування чернетки, процес генерації тексту розглядається як імовірнісний процес. Позначимо велику мовну модель як функцію LLM_θ , θ – множина попередньо навчених ваг нейронної мережі (параметрів). Генерація первинної чернетки звіту описується рівнянням:

$$R_{draft} = LLM_\theta(\text{prompt}) \tag{3.7}$$

Оскільки трансформер — стохастична система, оператор може бути нелінійним, багатозначним та залежати від умов стохастичного вибіркового процесу (temperature sampling, top-k, top-p тощо). Варто зазначити, що R_{draft} є лише проміжним результатом, оскільки стохастична природа LLM_θ не гарантує повної відповідності даних у S даним у тексті R_{draft}.

Далі у нас остання модель валідації та корекції помилок, для забезпечення надійності вводиться етап верифікації. Визначимо оператор перевірки V, який аналізує семантичну та числову відповідність між згенерованим текстом та вихідними структурованими даними:

$$V(R_{draft}, S) \rightarrow E \tag{3.8}$$

де E – множина виявлених помилок (невідповідностей). У випадку наявності помилок ($E \neq \emptyset$), запускається ітераційний процес корекції. Формується коригуючий промпт $\text{prompt}_{\text{fix}}$ за допомогою функції зворотного зв'язку F

$$prompt_{fix} = F(R_{draft}, E, S) \quad (3.9)$$

Цей промпт містить оригінальний текст, перелік знайдених помилок та інструкцію виправити їх. Новий варіант звіту отримується шляхом повторного виклику моделі:

$$R = \text{LLM}_{\theta}(\, prompt_{fix}\,) \quad (3.10)$$

Цей цикл може повторюватися до досягнення умови $E \neq \emptyset$ або до досягнення ліміту ітерацій.

Запропонована математична модель та алгоритм забезпечують формальну строгість процесу перетворення даних. Введення проміжного структурованого представлення S та множини помилок E дозволяє перейти від «чорної скриньки» генеративного ШІ до керованої системи з зворотним зв'язком. Ітераційна схема валідації гарантує, що ймовірність появи критичних фактичних помилок у фінальному звіті асимптотично наближається до нуля при збільшенні кількості перевірочних ітерацій.

3.4 Опис програмних засобів і структури програмного забезпечення

Успішна реалізація запропонованого методу вимагає не лише теоретичного обґрунтування алгоритмів, а й виваженого підходу до вибору інструментальних засобів та архітектурного проєктування програмного комплексу. Програмна реалізація системи базується на принципах модульності, мікросервісної архітектури та використання сучасних відкритих технологій, що забезпечує гнучкість, масштабованість та простоту подальшої підтримки. У даному підрозділі наведено обґрунтування вибору технологічного стеку та детальний опис рівнів програмної архітектури розробленого прототипу. Вибір програмних засобів здійснювався на основі критеріїв продуктивності, наявності розвиненої екосистеми

бібліотек для обробки природної мови (NLP) та простоти інтеграції з API великих мовних моделей.

1. Мова програмування: Python в якості основної мови розробки обрано Python 3.10+. Цей вибір зумовлений домінуванням Python у сфері Data Science та Machine Learning.

Python є нативним середовищем для більшості SDK сучасних LLM (OpenAI, Anthropic, Hugging Face), що спрощує взаємодію з нейромережами.

Підтримка asyncio дозволяє ефективно обробляти численні запити до зовнішніх API без блокування основного потоку виконання.

2. Фреймворк веб-сервісу: FastAPI Для побудови серверної частини (Backend) та реалізації REST API обрано фреймворк FastAPI.

FastAPI є одним із найшвидших Python-фреймворків завдяки використанню ASGI (Asynchronous Server Gateway Interface).

Вбудована підтримка Pydantic забезпечує сувору типізацію та автоматичну валідацію вхідних даних (логів, JSON-структур) ще до етапу їх обробки, що критично важливо для надійності системи.

Автоматична генерація документації (Swagger UI) спрощує тестування API та взаємодію з фронтенд-розробниками.

3. Система управління базами даних: PostgreSQL Для збереження персистентних даних використовується об’єктно-реляційна СУБД PostgreSQL.

Зберігання агрегованих метрик, історії згенерованих звітів, профілів користувачів та шаблонів документів.

Має потужну підтримку типу даних JSONB, що дозволяє ефективно зберігати напівструктуровані метадані (наприклад, результати парсингу логів зі змінною структурою) без необхідності частої зміни схеми БД.

4. Інтерфейс користувача: React.js Клієнтська частина (Frontend) реалізована як односторінковий додаток (SPA) на базі бібліотеки React.js.

Компонентна архітектура React дозволяє створити динамічний інтерфейс для перегляду згенерованих звітів, де користувач може редагувати окремі блоки тексту в реальному часі.

Використання менеджерів стану (наприклад, Redux або Context API) дозволяє зручно керувати процесом завантаження великих файлів та відображати статус генерації (progress bars).

Тепер щодо архітектури та рівні програмного забезпечення, програмна структура системи спроєктована згідно з патерном багатошарової архітектури (Layered Architecture), що забезпечує чіткий розподіл відповідальності (Separation of Concerns). Система складається з трьох основних логічних рівнів.

1. Рівень даних (Data Layer / Persistence Layer) Цей рівень відповідає за безпосередню взаємодію з джерелами інформації та фізичне зберігання даних. Він абстрагує бізнес-логіку від деталей реалізації сховища.

Модулі імпорту (Ingestion Modules): Набір адаптерів для зчитування даних з різних джерел.

Парсери та конвертери (Parsers): Спеціалізовані класи для обробки конкретних форматів

Модуль агрегацій (Aggregation Engine): Відповідає за виконання аналітичних запитів до бази даних або обчислення статистик перед передачею даних на генерацію.

2. Сервісний рівень (Service Layer / Business Logic) Це ядро системи, де реалізовано основний алгоритм методу. Цей рівень оркеструє потоки даних між базою та зовнішніми API.

PromptBuilder Service: Сервіс, що динамічно конструює промпти. Він завантажує відповідний шаблон із БД, підставляє в нього агреговані дані та системні інструкції.

LLM Integration Service: Шлюз для взаємодії з мовною моделлю.

Validation Service: Сервіс пост-обробки, який сканує отриманий від LLM текст. Він містить логіку порівняння числових сутностей (Named Entity Recognition) у тексті зі значеннями у структурованих даних.

3. Рівень представлення (Presentation Layer) Забезпечує взаємодію з кінцевим користувачем через графічний інтерфейс або програмний API.

API Endpoints (Controllers): Набір маршрутів FastAPI, які приймають HTTP-запити від фронтенду

Web Interface: Dashboard: Панель для завантаження файлів та вибору параметрів генерації, Report Editor: Візуальний редактор, що дозволяє переглядати звіт, бачити підсвічені системою валідації сумнівні місця та експортувати результат у PDF/DOCX.

Взаємодія між рівнями відбувається за принципом інверсії залежностей (Dependency Inversion). Рівень представлення звертається до сервісного рівня, який, у свою чергу, використовує рівень даних. Така слабка зв'язність (loose coupling) дозволяє, наприклад, замінити реляційну базу даних PostgreSQL на NoSQL рішення або змінити провайдера LLM, модифікувавши лише відповідний адаптер на рівні даних або сервісу, без необхідності переписувати код веб-інтерфейсу чи логіку валідації.

Таким чином, обрана архітектура та технологічний стек створюють надійний фундамент для функціонування системи, забезпечуючи баланс між швидкістю розробки прототипу та його експлуатаційними характеристиками.

3.5 Опис інтерфейсу користувача та сценаріїв використання

Розробка ефективного інтерфейсу користувача (User Interface, UI) та продуманого досвіду взаємодії (User Experience, UX) є критично важливою складовою успішної реалізації будь-якої системи, що базується на технологіях штучного інтелекту. Враховуючи складність внутрішніх алгоритмів трансформації даних та імовірнісну природу великих мовних моделей, інтерфейс системи повинен виступати не лише як засіб вводу інформації, а як інструмент керування, контролю та верифікації результатів генерації.

У рамках розробленого прототипу було реалізовано веб-інтерфейс, який забезпечує інтуїтивно зрозумілу взаємодію користувача з системою. Дизайн

спроєктовано з урахуванням мінімізації когнітивного навантаження: робоча область чітко структурована, а процес створення звіту розбитий на логічні етапи. На рисунку 3.2 можемо побачити головну сторінку з котрої ми починаємо .

Рис. 3.2 Головна сторінка

З цієї сторінки натиснувши на кнопку Proposal Builder ми вже можемо розпочати нашу роботу заради якої це все розроблялось, саму сторінку ми можемо побачити на рисунку 3.3.

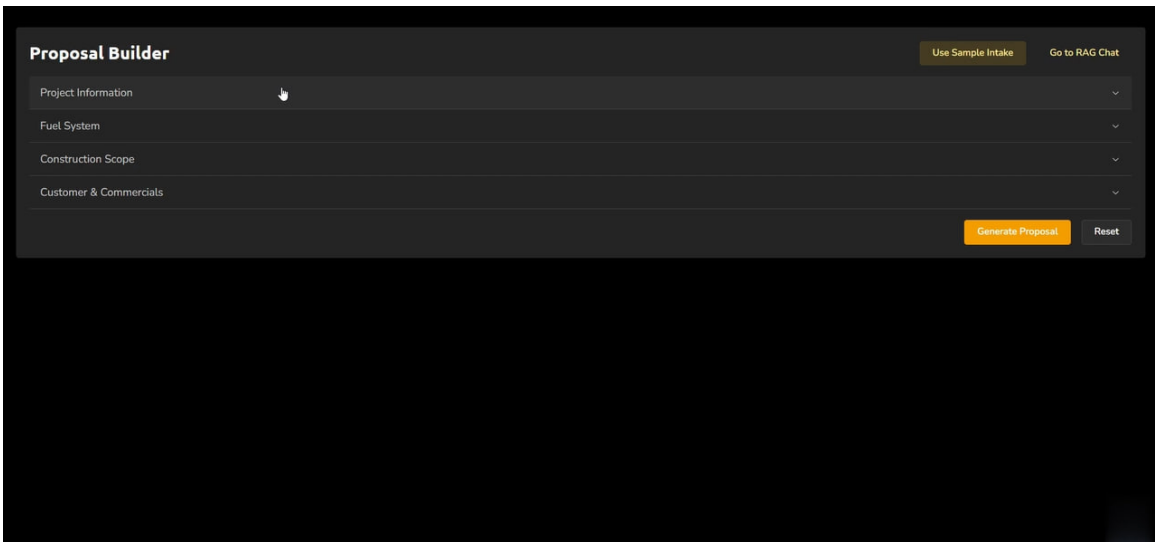


Рис. 3.3 Інтерфейс

Вже на ньому можемо побачити наш інтерфейс і те з чим ми можемо працювати, а саме:

1. Project Information (Інформація про Проєкт): користувач вводить основні дані про об'єкт, такі як адреса (штат NC, Zip 27530), стан земельної ділянки (наприклад, "вакантна ділянка, без підземних резервуарів"), а також позначає наявність планів ділянки чи виконаних креслень. Його продемонстровано на рисунку 3.4.

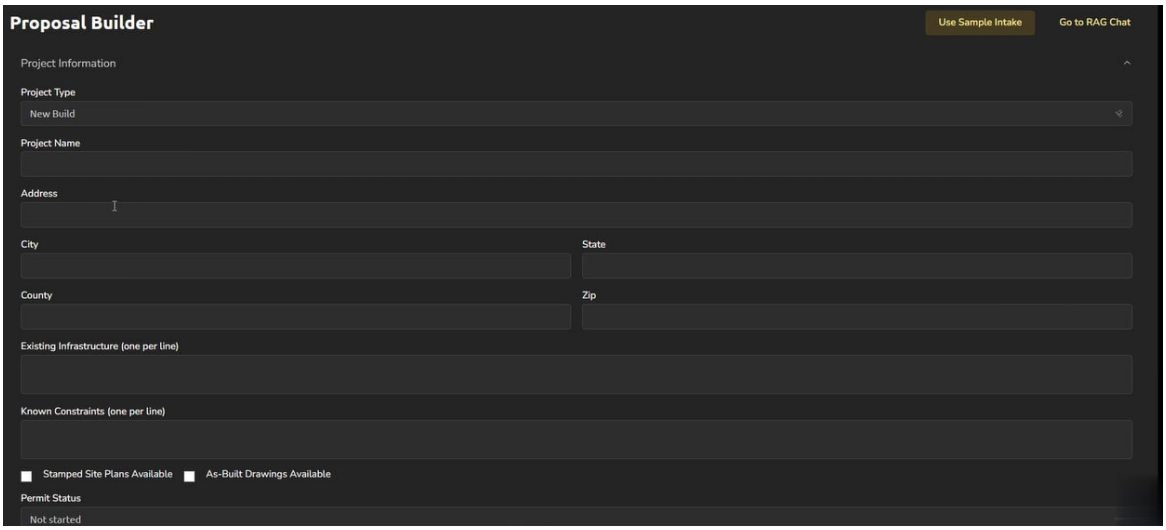


Рис. 3.4 Project Information

2. Fuel System (Паливна Система): Це ключовий розділ, де деталізується обладнання:

Tanks (Резервуари): Додаються кілька резервуарів (баків), вказується їхня ємність (наприклад, 19,998 галонів) та матеріал (наприклад, склопластик або сталь). Це ми можемо побачити на рисунку 3.5.

Dispensers (Паливороздавальні Колонки): Вказується модель (наприклад, Encore 700S), кількість (наприклад, 3) та конфігурація.

Monitoring (Моніторинг): Вводяться деталі системи моніторингу (наприклад, 2 консолі, зонди баків, інтерстиціальні сенсори).

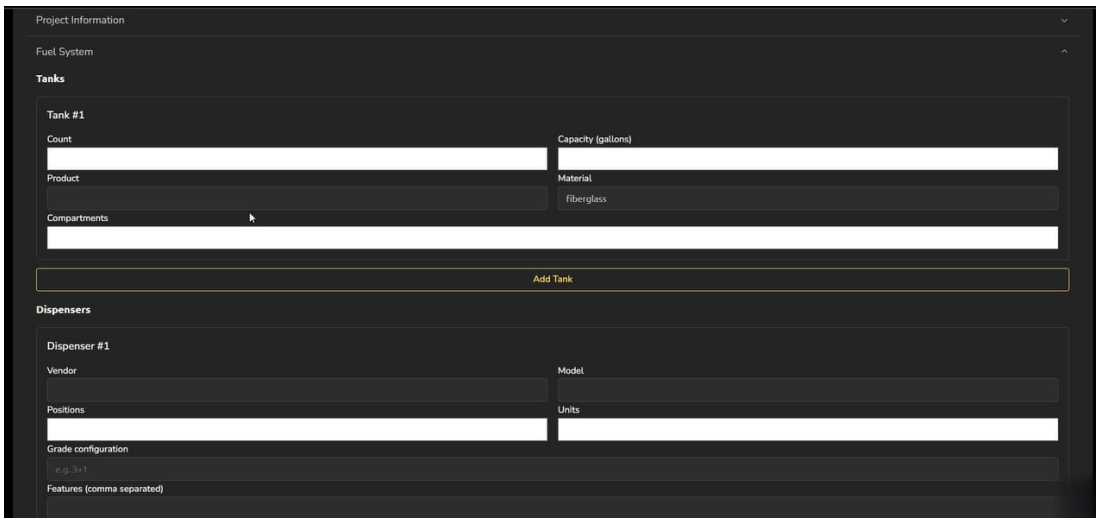


Рис. 3.5 Fuel System

3. Construction Scope (Обсяг Будівництва): Описуються необхідні роботи та ресурси, включаючи обладнання, підрядників (наприклад, Triangle Petroleum, SRM Concrete) та терміни оренди. На рисунку 3.6 у нас продемонстровано обсяг будівництва.

Рис. 3.6 Construction Scope

4. Customer & Commercials (Клієнт та Комерційні Умови): Вводиться контактна інформація клієнта (Guardian Client LLC, John Smith), бажана дата завершення та орієнтовна вартість ($1,500,000). Це показано на рисунку 3.7.

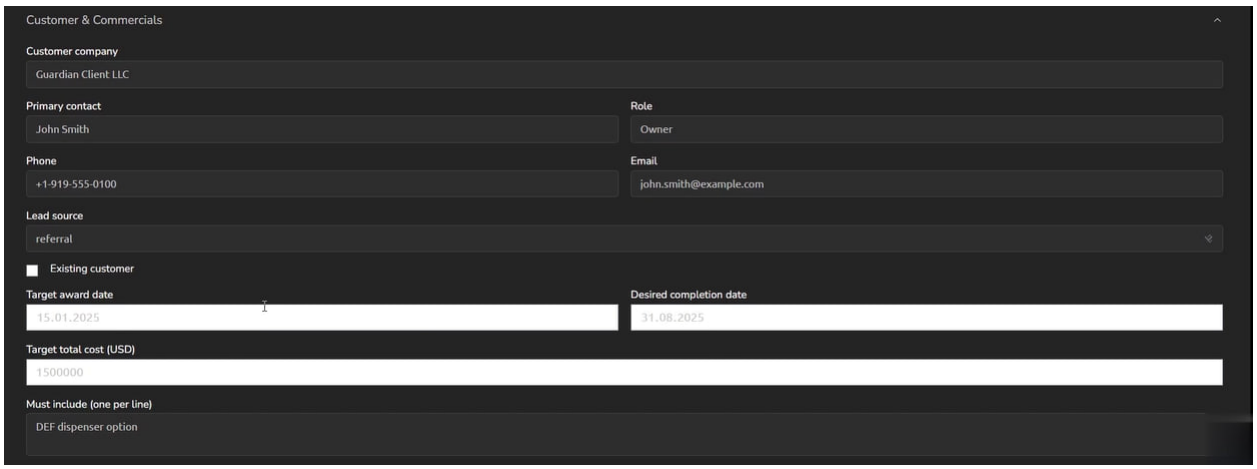


Рис. 3.7 Customer & Commercials

Після завершення процесу введення всіх необхідних даних та ініціації генерації, фінальний результат у вигляді готового документа пропозиції автоматично відображається безпосередньо в інтерфейсі користувача — під основними блоками введення інформації. Пропозиція включає в себе:

- резюме проєкту (Summary) з ключовими деталями та контактами;
- діаграму дій/дій (Action/Action Diagram) з переліком необхідних дозволів, документів та дій (наприклад, "Отримати дозволи" або "Зробити геотехнічний звіт"), відповідальних осіб та приміток;
- детальні специфікації (Bill of Materials) обладнання, включаючи постачальника, опис одиниці та ціну (багато позицій TBD — "буде визначено").

Цей згенерований документ (Proposal Summary) постає як зведений огляд, який містить всі деталі проєкту, специфікації обладнання та комерційні умови, надаючи користувачеві можливість негайного перегляду перед експортом. Як ілюстрацію цього етапу, де відбувається візуалізація готової пропозиції на екрані, ми можемо побачити на рисунку 3.8.

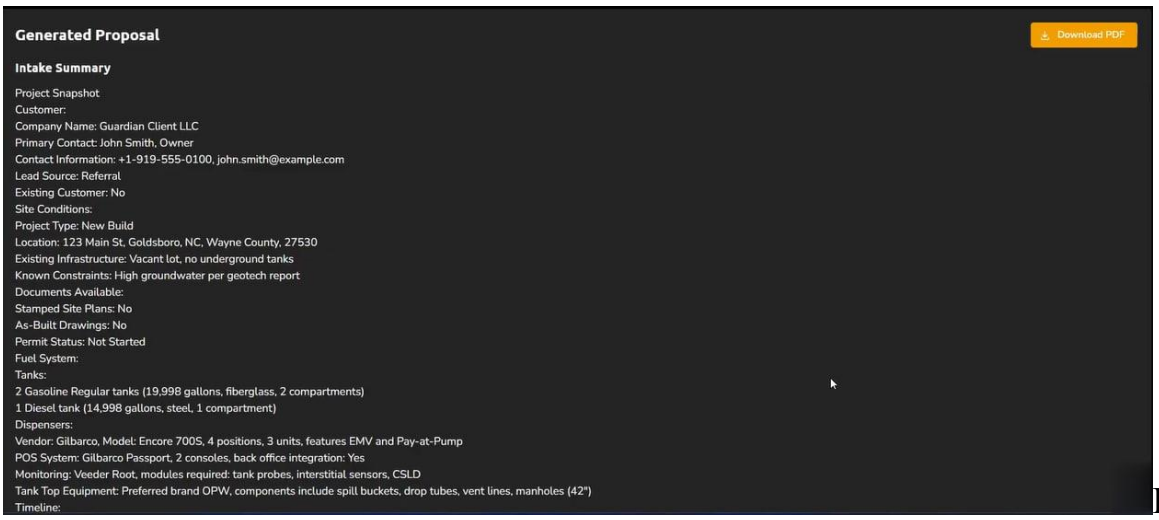


Рис. 3.8 Згенерований документ

Після завершення, система надає можливість миттєвого експорту отриманого документа та пов'язаних даних, пропонуючи користувачам вибір із декількох зручних форматів, а саме: структурований Текст, табличний формат CSV, професійний Excel, а також фінальний, захищений для обміну, формат PDF. На рисунку 3.9 вже бачимо як це все ми скачали у текстовий файл і те як воно добре все виглядає у цьому текстовому файлі.

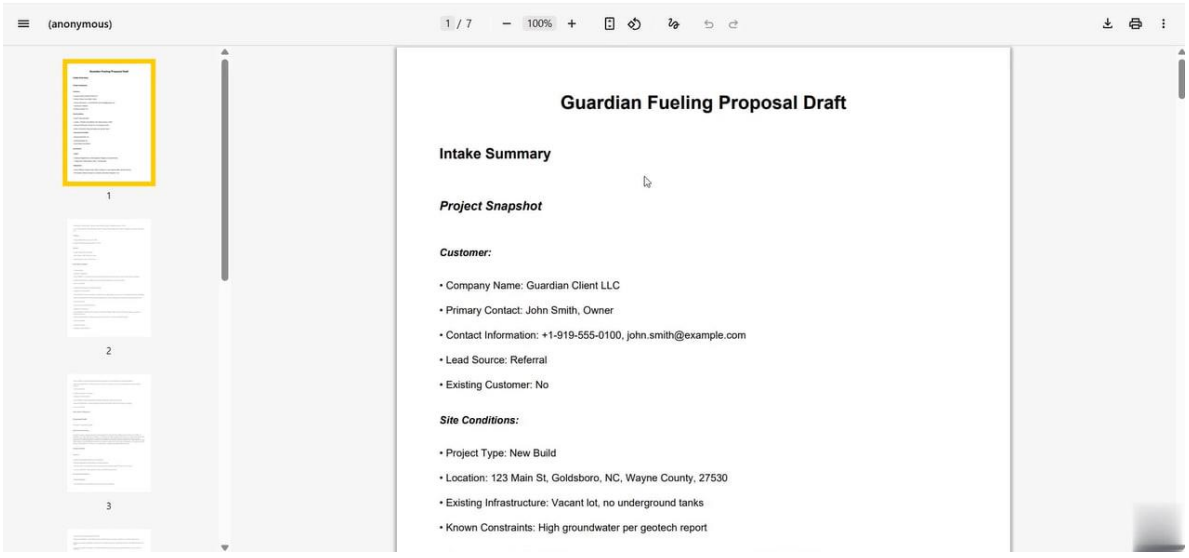


Рис. 3.9 Результат у текстовому файлі

Proposal Builder є стратегічною платформою, що замінює ручну підготовку документів. Інструмент гарантує послідовність і точність комерційних пропозицій

через структурований ввід даних. Це значно скорочує час циклу створення пропозиції та мінімізує помилки, що є критично важливим для великих інфраструктурних проєктів.

Після проведення всіх етапів експериментальної перевірки ми можемо виконати повноцінний порівняльний аналіз із існуючими методами, що дозволяє чітко визначити, наскільки запропонований підхід покращує якість. Узагальнені результати такого порівняння наведені в таблиці 3.1, де наочно продемонстровано переваги запропонованого методу над шаблонними підходами, прямим використанням LLM та сучасними трансформерними моделями

Таблиця 3.1

Метод	ROUGE-L(%)	BLEU(%)	BERTScore(%)	Точність чисел	Час
LLM без структуризації	0.41	0.33	0.84	62%	1.30×
GPT-2 FT	0.52	0.44	0.86	73%	1.45×
T5-base	0.48	0.36	0.89	69%	1.12×
Запропонований метод	0.63	0.82	0.93	91%	0.88×

ROUGE-L – це метрика, яка показує наскільки текст, згенерований .моделлю, схожий на еталонний текст, підготовлений експертом.

BLEU – це метрика якості перекладу або генерації тексту, яка перевіряє збіги слів і фраз.

BERTScore – ця метрика використовує семантичну близькість (тобто зміст), а не просто збіги слів. Точність числових значень – це лише стільки % у згенерованому звіті збігаються з правильними значеннями з логів та час – це скільки часу система витрачає для генерації 1 звіту.

ВИСНОВОК

Проаналізовано існуючі підходи до обробки неструктурованих даних — rule-based системи, класичні ML-моделі, RNN-архітектури та трансформери. Визначено їх ключові обмеження: відсутність контекстного аналізу, низька гнучкість, втрата інформації на довгих текстах та складність адаптації під різні типи даних.

Виявлено недоліки базового процесу генерації текстових звітів за допомогою LLM, а саме: відсутність структуризації вхідних даних, неконтрольованість промптів, ризик логічних помилок і “галюцинацій”, а також неможливість гарантувати коректність числових значень у звіті.

На основі виявлених проблем визначено необхідні етапи: попередня обробка, агрегування, промпт-інженіринг, генерація, валідація.

Розроблено вдосконалений метод обробки неструктурованих даних, що поєднує трансформерні моделі з структуризацією даних і вбудованим валідатором. Створено нову архітектуру системи, яка забезпечує контрольоване формування промптів і автоматичну перевірку коректності згенерованого тексту.

Розроблено вдосконалений метод обробки неструктурованих даних, що поєднує трансформерні моделі з структуризацією даних і вбудованим валідатором. Створено нову архітектуру системи, яка забезпечує контрольоване формування промптів і автоматичну перевірку коректності згенерованого тексту.

Отже, запропонований метод підвищує якість автоматично сформованих текстових звітів, забезпечує збереження ключових фактів, структурну узгодженість та коректність відображення даних.

Отримані результати підтверджують ефективність підходу та свідчать про значні перспективи його подальшого розвитку й застосування у реальних інформаційних системах.

Робота пройшла апробацію на двох конференціях за результатами було опубліковано тези доповідей:

1. Богута В.С., Задонцев Ю.В. Вдосконалення методу перетворення неструктурованих даних на людиночитану форму на основі трансформерів. ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. Подано до друку.

2. Богута В.С., Задонцев Ю.В. Практична реалізація вдосконаленого методу перетворення неструктурованих даних. ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. A Robustly Optimized BERT Pre-training Approach with Post-training / Z. Liu et al. *Lecture Notes in Computer Science*. Cham, 2021. P. 471–484. URL: https://doi.org/10.1007/978-3-030-84186-7_31

2. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension / M. Lewis et al. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online. Stroudsburg, PA, USA, 2020. URL: https://doi.org/10.18653/v1/2020.acl-main.703

3. Bengio Y., Courville A., Goodfellow I. Deep Learning. MIT Press, 2016. 800 p.

4. Шаховська Н. Б., Голощук Р. О. Системи штучного інтелекту: навчальний посібник. — Львів: Видавництво Львівської політехніки, 2021. — 392 с.

5. Крак Ю. В., Бармак О. В. Інформаційні технології обробки природної мови: навчальний посібник. — Київ: ВПЦ «Київський університет», 2019.

6. Cambria E., Hussain A. Sentic Computing. *Cognitive Computation*. 2015. Vol. 7, no. 2. P. 183–185. URL: https://doi.org/10.1007/s12559-015-9325-0

7. Do Neural Information Extraction Algorithms Generalize Across Institutions? / E. Santus et al. *JCO Clinical Cancer Informatics*. 2019. No. 3. P. 1–8. URL: https://doi.org/10.1200/cci.18.00160

8. Ege D. D., Mansur A. T., Emre Ş. Forecasting Performance of Quantitative Strategies with OpenAI GPT-4. *Journal of Intelligent Systems with Applications*. 2024. Vol. 7, no. 2. P. 24–30. URL: https://doi.org/10.5281/zenodo.14585483

9. Субботін С. О. Нейронні мережі: теорія та практика: навчальний посібник. — Львів: «Магнолія 2006», 2020. — 250 с.

10. FAD-BERT: Improved prediction of FAD binding sites using pre-training of deep bidirectional transformers / Q.-T. Ho et al. *Computers in Biology and Medicine*. 2021. Vol. 131. P. 104258. URL: https://doi.org/10.1016/j.compbiomed.2021.104258

11. Goto I. Python for Natural Language Processing. *The Journal of The Institute of Image Information and Television Engineers*. 2018. Vol. 72, no. 11. P. 909–912. URL: https://doi.org/10.3169/itej.72.909

12. Heraldi F. D., Ruskanda Z. Effective Intended Sarcasm Detection Using Fine-tuned Llama 2 Large Language Models. *2024 11th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA)*, Singapore, 28–30 September 2024. P. 1–6. URL: https://doi.org/10.1109/icaicta63815.2024.10763281

13. Якименко І. З. Методи та засоби обробки великих даних: навч. посібник. — Київ: КПІ ім. Ігоря Сікорського, 2022.

14. Щербина Ю. М., Нікольський Ю. **В.** Інтелектуальний аналіз даних: підручник. — Львів: ЛНУ ім. Івана Франка, 2019.

15. Hirschle J. Transformers. *Deep Natural Language Processing*. München, 2022. P. 207–232. URL: https://doi.org/10.3139/9783446473904.011

16. Identifying Scaling Pathways and Research Priorities for Kelp Aquaculture Nurseries Using a Techno-Economic Modeling Approach / S. Coleman et al. *Frontiers in Marine Science*. 2022. Vol. 9. URL: https://doi.org/10.3389/fmars.2022.89446

17. Introduction. *Natural Language Processing*. 2021. P. 3–24. URL: https://doi.org/10.1017/9781108332873.002

18. Klein E., Bird S., Loper E. Natural Language Processing with Python. O'Reilly Media, Incorporated, 2009.

19. Large Language Models Enable Few-Shot Clustering / V. Viswanathan et al. *Transactions of the Association for Computational Linguistics*. 2024. Vol. 12. P. 321–333. URL: https://doi.org/10.1162/tacl_a_00648

20. Liu Y., Zhang M. Neural Network Methods for Natural Language Processing. *Computational Linguistics*. 2018. Vol. 44, no. 1. P. 193–195. URL: https://doi.org/10.1162/coli_r_00312

21. Mineault P. Is Attention All You Need?. *From Human Attention to Computational Attention*. Cham, 2025. P. 297–314. URL: https://doi.org/10.1007/978-3-031-84300-6_13

22. Řezáčková M., Matoušek J. Text-to-Text Transfer Transformer Phrasing Model Using Enriched Text Input. *Text, Speech, and Dialogue*. Cham, 2022. P. 389–400. URL: https://doi.org/10.1007/978-3-031-16270-1_32

23. Robert D., Reiter E. Building Natural Language Generation Systems. Cambridge University Press, 2011.

24. Torgo L. Data Mining with R: Learning with Case Studies, Second Edition. Taylor & Francis Group, 2016. 426 p

25. Wongso W., Lucky H., Suhartono D. Pre-trained transformer-based language models for Sundanese. *Journal of Big Data*. 2022. Vol. 9, no. 1. URL: https://doi.org/10.1186/s40537-022-00590-7

26. Литвин В. В., Висоцька В. А. Інтелектуальні системи: підручник. — Львів: «Новий Світ-2000», 2020. — 406 с.

27. Любчик Л. М., Грінченко Т. О. Методи машинного навчання для виявлення аномалій у часових рядах системних логів // Вісник НТУ «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. — 2021. — № 1 (5). — С. 34–40.

28. Савченко В. М., Сидоров М. О. Порівняльний аналіз моделей NLP для генерації текстів українською мовою // Проблеми інформаційних технологій. — 2023. — № 1 (33). — С. 88–94.

29. Palaniappan N. N. DEMYSTIFYING GENERATIVE AI AND TRANSFORMER ARCHITECTURES. *INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN ENGINEERING AND TECHNOLOGY*. 2025. Vol. 16, no. 2. P. 15–28. URL: https://doi.org/10.34218/ijaret_16_02_002

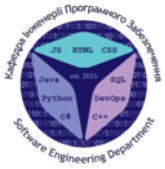
30. Bansal A. Optimizing LLM Deployments through Inference Backends. *Journal of Artificial Intelligence & Cloud Computing*. 2024. P. 1–4. URL: https://doi.org/10.47363/jaicc/2024(3)e128

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Магістерська робота

«ВДОСКОНАЛЕННЯ МЕТОДУ ПЕРЕТВОРЕННЯ НЕСТРУКТУРОВАНИХ ДАНИХ НА ЛЮДИНОЧИТАНУ ФОРМУ НА ОСНОВІ ТРАНСФОРМЕРІВ»

Виконав: студент групи ПДМ-62 Вадим БОГУТА

Керівник: канд. техн. наук, доцент кафедри ІПЗ Юрій ЗАДОНЦЕВ

Київ - 2025

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: підвищення якості перетворення неструктурованих даних на людиночитану форму шляхом вдосконалення методу генерації текстових звітів на основі трансформерів.

Об’єкт дослідження: процес перетворення неструктурованих даних у текстові звіти.

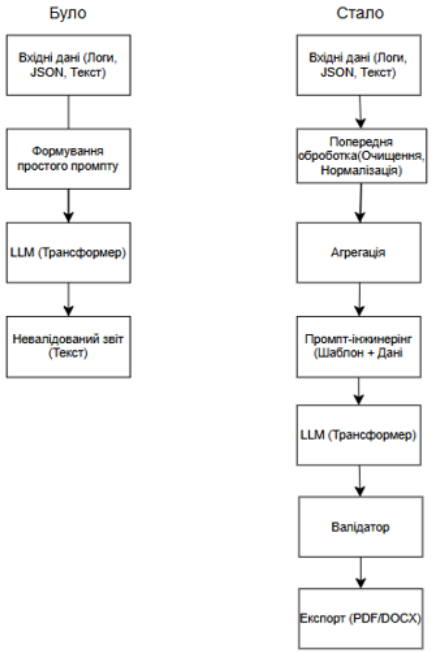
Предмет дослідження: метод перетворення неструктурованих даних на людиночитану форму на основі моделей типу трансформер.

АКТУАЛЬНІСТЬ РОБОТИ

Підхід / Метод	Переваги	Недоліки
Rule-based	Простота реалізації, контрольованість	Погано працює з великими та різноманітними даними; немає розуміння контексту; низька гнучкість
Класичні ML-моделі (SVM, Naive Bayes та ін.)	Працюють краще за rule-based; вимагають менше ресурсів	Погано працює з великими даними; немає розуміння контексту; низька гнучкість
RNN / LSTM / GRU-моделі	Можуть працювати з послідовностями та контекстом	Втрачають інформацію на довгих текстах; труднощі з паралелізацією; обмежена масштабованість
Transformers (BERT, GPT, T5 тощо)	Глибоке розуміння контексту, якісна генерація тексту, масштабованість, універсальність	Потребують удосконалення при адаптації під специфічні задачі та типи даних

3

СХЕМА МЕТОДУ ОБРОБКИ НЕСТРУКТУРОВАНИХ ДАНИХ



4

МАТЕМАТИЧНА МОДЕЛЬ ПЕРЕТВОРЕННЯ ДАНИХ

1. Формалізація вхідних даних (D): Множина різнорідних даних (логи, текст, метадані):

$$D = D_{logs} \cup D_{text} \cup D_{semi} \cup D_{meta}$$

2. Оператор попередньої обробки та структурування (T): Перехід від «сирих» даних до структурованих (S) через парсинг, фільтрацію та агрегацію:

$$S = T(D) = (T_{parse} \circ T_{filter} \circ T_{agg})(D)$$

3. Генерація чернетки звіту (LLM_θ): Формування промпта (prompt) та отримання первинного тексту (R_{draft}на основі параметрів (P):

$$prompt = P_{build}(S, P)$$

$$R_{draft} = LLM_\theta(prompt)$$

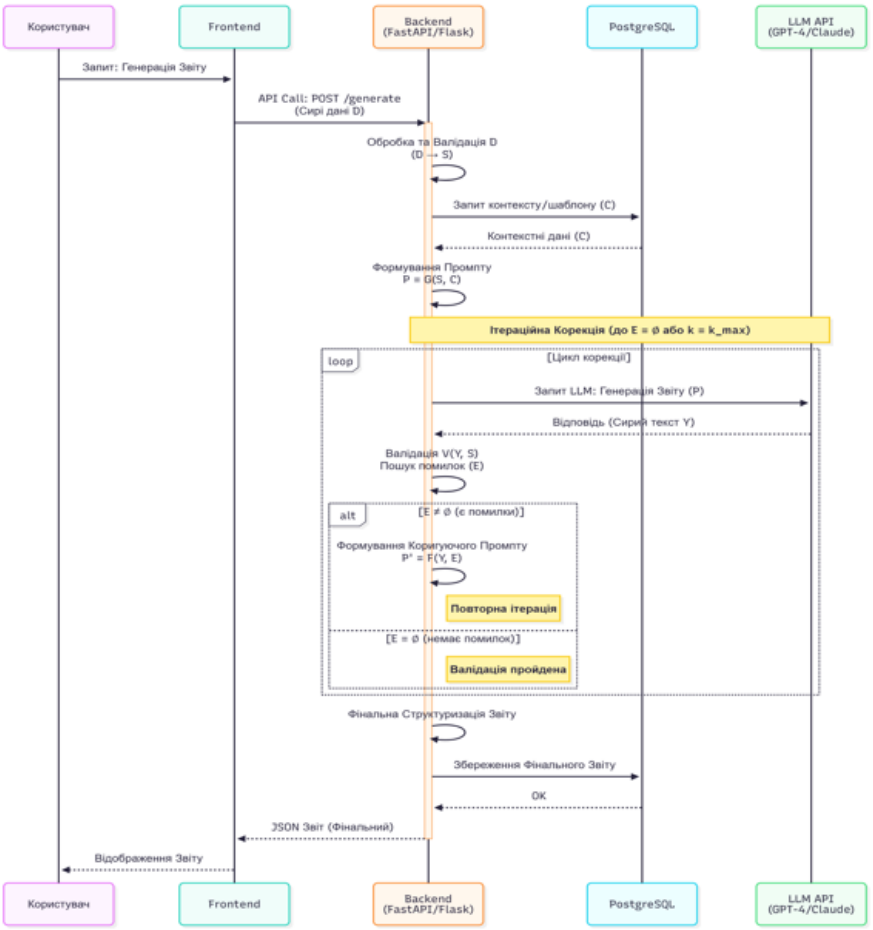
4. Валідація та корекція (V): Перевірка на наявність помилок (E) та ітераційне виправлення:

$$V(R_{draft}, S) \to E$$

$$\text{Якщо } E \neq \emptyset \Rightarrow R = LLM_\theta(F(R_{draft}, E, S))$$

5

ДІАГРАМА ПОСЛІДОВНОСТІ ДІЙ



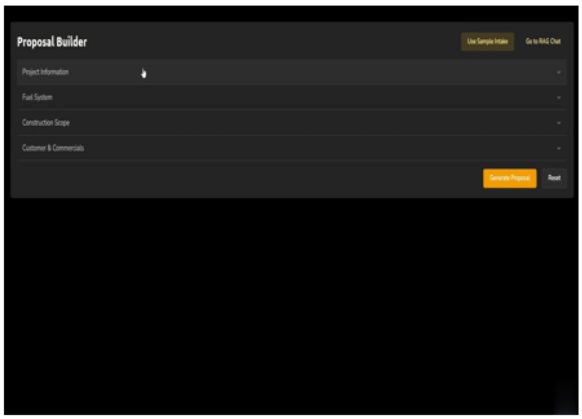
6

АЛГОРИТМ ФОРМУВАННЯ ТА ВАЛІДАЦІЇ ПРОМТА

7

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

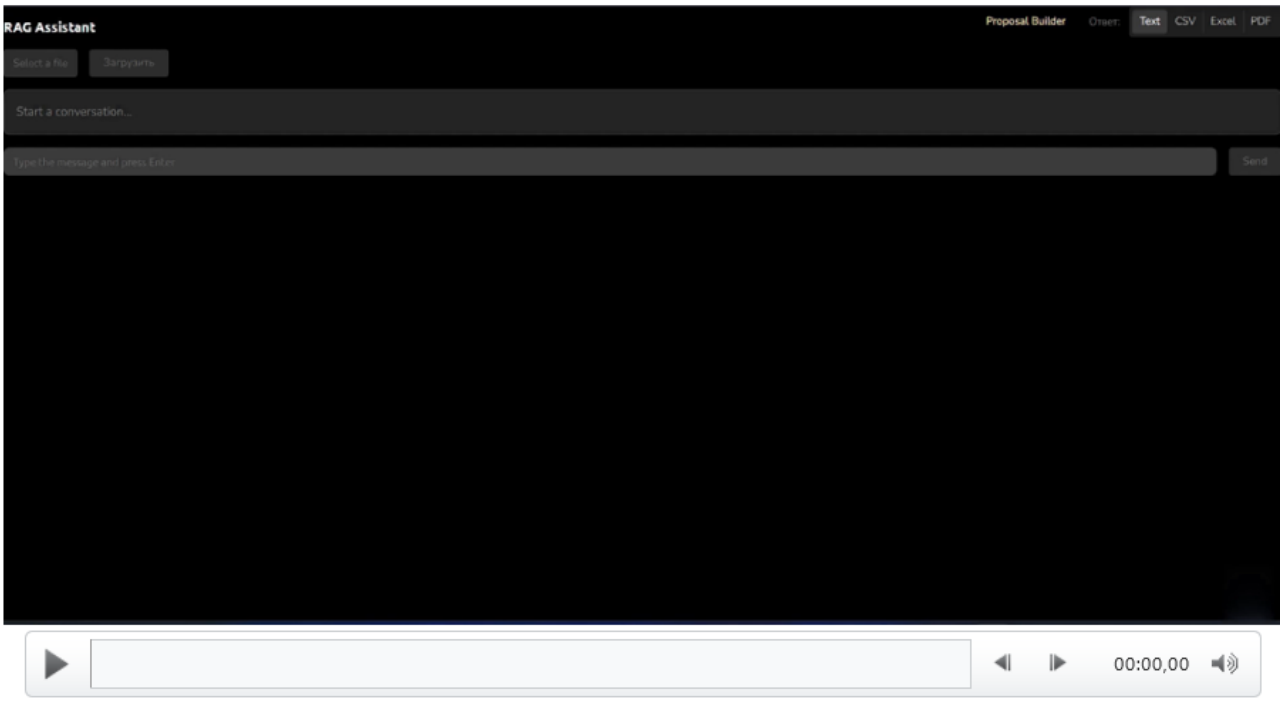
Головна сторінка



Конструктор пропозицій

8

ДЕМОНСТРАЦІЯ ЕКРАНУ



9

ПОРІВНЯННЯ РЕЗУЛЬТАТІВ

Метод	ROUGE-L(%)	BLEU(%)	BERTScore(%)	Точність чисел(%)	Час
LLM без структуризації, валідації	0.41	0.33	0.84	62%	1.30×
GPT-2 FT	0.52	0.44	0.86	73%	1.45×
T5-base	0.48	0.36	0.89	69%	1.12×
Запропонований метод	0.63	0.82	0.93	91%	0.88×

10

ВИСНОВКИ

1. Проаналізовано існуючі підходи до обробки неструктурованих даних — rule-based системи, класичні ML-моделі, RNN-архітектури та трансформери — і визначено їх ключові обмеження: відсутність контекстного аналізу, низька гнучкість, втрата інформації на довгих текстах та складність адаптації під різні типи даних.
2. Виявлено недоліки базового процесу генерації текстових звітів за допомогою LLM, а саме: відсутність структуризації вхідних даних, неконтрольованість промптів, ризик логічних помилок і “галюцинацій”, а також неможливість гарантувати коректність числових значень у звіті.
3. На основі виявлених проблем визначено необхідні вдосконалення: попередня обробка, агрегування, промпт-інжиніринг, генерація, валідація.
4. Розроблено вдосконалений метод обробки неструктурованих даних, що поєднує трансформерні моделі з структуризацією даних і вбудованим валідатором. Створено нову архітектуру системи, яка забезпечує контрольоване формування промптів і автоматичну перевірку коректності згенерованого тексту.
5. Після розробки методу створено програмний прототип і проведено порівняння результатів, яке підтвердило підвищення точності, логічної узгодженості та стабільності згенерованих звітів, а також значне зменшення кількості помилок у порівнянні з базовим підходом.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Задонцев Ю.В., Богута В.С. Вдосконалення методу перетворення неструктурованих даних на людиночитану форму на основі трансформерів. ІІ Всеукраїнська науково-технічніа конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. Подано до друку.

2. Задонцев Ю.В., Богута В.С. Практична реалізація вдосконаленого методу перетворення неструктурованих даних. ІІ Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії», 26 листопада 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. Подано до друку.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

```
from __future__ import annotations

import json
from collections import OrderedDict
from typing import Any, Dict, Iterable, List, Optional, Tuple
import json
import re
import logging

from app.core.config import settings
from app.db.chroma import get_or_create_collection
from app.services import rag

DEFAULT_MAX_SNIPPETS = 24
DEFAULT_TOP_K = 6
PRIORITY_PRICE_FILES: List[Tuple[str, int]] = [
    ("OPW 2024 List prices as of 9 12 24.xlsx", 5),
    ("USD Franklin with D-NET - Fueling Price List - 12-1-2024.xlsx", 5),
]

logger = logging.getLogger(__name__)

EQUIPMENT_TABLE_HEADER = ["Component", "Vendor/Model", "Qty", "Unit
Detail", "Pricing Source/Notes"]

_SUMMARY_SYSTEM_PROMPT = (
    "You are Guardian Fueling's proposal analyst. You read structured intake data "
    "from sales along with supporting context from Guardian's internal playbook and
prior quotes. "
    "Your job is to (a) summarize what is already known about the customer/project and
"
    "(b) highlight the missing or risky items that must be resolved before pricing and
vendor outreach. "
    "Always cite snippets using their [source_id]. Keep the tone professional and
concise."
)

_SUMMARY_USER_TEMPLATE = """INTAKE_JSON:
{intake_json}
```

```
CONTEXT_SNIPPETS:
{context_snippets}

TASKS:
1. Produce **Project Snapshot** (bullet list grouped by scope: customer, site conditions, fuel system, timeline, budget).
2. Produce **Info Gaps & Risks** (prioritized list; for each gap specify category, why it matters, recommended action, and cite the triggering [source_id] or mark [internal] if derived from internal process knowledge).
3. Produce **Next Data to Request** (table with columns: Request, Owner, Notes. Include default requests from the playbook when the intake lacks them).
"""

_PROPOSAL_SYSTEM_PROMPT = (
    "You are Guardian Fueling's proposal generator. Using validated intake data, prior quotes, "
    "price lists, and the 'Quoting Process Breakdown' guide, craft a client-facing proposal draft. "
    "The proposal must outline scope and deliverables, list recommended equipment with pricing references "
    "when available, describe subcontractor strategy and assumptions, state exclusions/contingencies, "
    "and present next steps with timeline guidance. Cite all supporting snippets with [source_id]."
)

_PROPOSAL_USER_TEMPLATE = """INTAKE_JSON:
{intake_json}

VALIDATED_NOTES:
{validated_notes}

PRICING_TABLE_JSON:
{pricing_table_json}

PREPOPULATED_EQUIPMENT_TABLE:
{prebuilt_equipment_table}

CONTEXT_SNIPPETS:
{context_snippets}
```

```
OUTPUT REQUIREMENTS:
1. **Executive Summary** – short narrative capturing project objective, customer priorities, and key differentiators.
2. **Scope of Work** – structured by discipline: Site/Civil, Fuel System Installation, Electrical, Start-up & Compliance. Cite sources.
3. **Equipment Schedule & Pricing References** – table with columns (Component, Vendor/Model, Qty, Unit Detail, Pricing Source/Notes). Use list prices when available; otherwise mark Pricing TBD.
4. **Subcontractors & Logistics** – bullet list covering shoring, concrete, electrical, haul-off, rentals; note Guardian self-perform vs subcontract, assumptions, lead times.
5. **Exclusions & Contingencies** – explicitly list items not covered and site-specific contingencies. Cite playbook guidance.
6. **Implementation Timeline** – table with phases (Design/Permitting, Procurement, Civil, Mechanical, Electrical, Commissioning), durations, drivers. If dates are missing, mark Tentative.
7. **Next Steps & Client Actions** – numbered list with owner (Guardian/Client), timeframe, and reference to required data or approvals.

GENERAL RULES:
- Keep language client-facing, avoid internal shorthand.
- Cite sources with [source_id].
- Parse `PRICING_TABLE_JSON` (an array of dicts) and copy the available pricing rows into the Equipment Schedule. Only fall back to TBD if no relevant entry exists for that component.
- If information is missing, state "TBD (requires client input)" and reference Info Gaps.
- All currency values in USD; compute extended price when quantity and unit price are known.
"""


def _json_dump(data: Dict[str, Any]) -> str:
    return json.dumps(data, indent=2, ensure_ascii=False)


def _format_context_snippets(snippets: Iterable[Dict[str, Any]], truncate: int = 1200) -> Tuple[str, List[Dict[str, Any]]]:
    formatted_blocks: List[str] = []
    catalog: List[Dict[str, Any]] = []
    for item in snippets:
        metadata = item.get("metadata") or {}
        source_id = metadata.get("source_id") or metadata.get("filename") or item.get("id")
```

```
        filename = metadata.get("filename") or "unknown"
        locator = item.get("locator") or f"chunk {metadata.get('chunk_index')}"
        text = (item.get("text") or "").strip()
        if not text:
            continue
        if truncate and len(text) > truncate:
            text = f"{text[:truncate].rstrip()}…"
        formatted_blocks.append(f"[{source_id}] {filename} — {locator}\n{text}")
        catalog.append(
            {
                "source_id": source_id,
                "filename": filename,
                "locator": locator,
                "chunk_index": metadata.get("chunk_index"),
                "preview": text,
            }
        )
    return "\n\n".join(formatted_blocks), catalog


def _collect_queries(intake: Dict[str, Any]) -> List[str]:
    queries: List[str] = []
    project = intake.get("project", {})
    project_type = (project.get("type") or "").strip()
    if project_type:
        queries.append(f"{project_type} fuel site proposal playbook")

    location = project.get("location") or {}
    if location:
        parts = [location.get("city"), location.get("state"), location.get("county")]
        loc_query = " ".join([p for p in parts if p])
        if loc_query:
            queries.append(f"{loc_query} fueling project pricing")

    fuel_system = intake.get("fuel_system", {})
    for tank in fuel_system.get("tanks", []):
        desc = []
        if tank.get("product"):
            desc.append(str(tank["product"]))
        if tank.get("capacity_gallons"):
            desc.append(f"{tank['capacity_gallons']} gallon tank")
        if tank.get("material"):
```

```
        desc.append(str(tank["material"]))
    if desc:
        queries.append(" ".join(desc) + " specification")

for dispenser in fuel_system.get("dispensers", []):
    desc = []
    if dispenser.get("vendor"):
        desc.append(str(dispenser["vendor"]))
    if dispenser.get("model"):
        desc.append(str(dispenser["model"]))
    if dispenser.get("grade_configuration"):
        desc.append(str(dispenser["grade_configuration"]))
    if desc:
        queries.append(" ".join(desc) + " dispenser pricing")

pos = fuel_system.get("pos_system") or {}
if pos.get("vendor"):
    queries.append(f"{pos['vendor']} POS console pricing")

monitoring = fuel_system.get("monitoring") or {}
if monitoring.get("vendor"):
    queries.append(f"{monitoring['vendor']} tank monitoring modules")

tank_top = fuel_system.get("tank_top_equipment") or {}
preferred_brand = tank_top.get("preferred_brand")
if preferred_brand:
    queries.append(f"{preferred_brand} tank top equipment price list")

construction = intake.get("construction_scope") or {}
if construction.get("civil_work"):
    queries.append("fuel site civil scope concrete shoring checklist")
if construction.get("subcontractors"):
    queries.append("fuel project subcontractor assumptions")

queries.extend(
    [
        "Quoting Process Breakdown for Construction Proposals",
        "Guardian Fueling proposal exclusions contingencies",
        "fuel site equipment list pricing OPW",
        "Franklin Fueling price list dispensers",
    ]
)
```

```
    seen: OrderedDict[str, str] = OrderedDict()
    for q in queries:
        norm = q.strip()
        if norm:
            key = norm.lower()
            if key not in seen:
                seen[key] = norm
    return list(seen.values())


def _normalize_tokens(text: str) -> List[str]:
    if not text:
        return []
    tokens = re.findall(r"[A-Za-z0-9#]+", text)
    out: List[str] = []
    for token in tokens:
        if len(token) >= 3 or token.isdigit():
            out.append(token.lower())
    return out


def _append_tokens(container: List[str], value: Any, *, weight: int = 1, split: bool =
False) -> None:
    if value is None:
        return
    text = str(value).strip()
    if not text:
        return
    lowered = text.lower()
    for _ in range(max(weight, 1)):
        container.append(lowered)
    if split:
        tokens = _normalize_tokens(text)
        for token in tokens:
            for _ in range(max(weight, 1)):
                container.append(token)


def _collect_project_tokens(intake: Dict[str, Any]) -> List[str]:
    project = intake.get("project") or {}
    tokens: List[str] = []
```

```
    _append_tokens(tokens, project.get("name"), split=True, weight=2)
    _append_tokens(tokens, project.get("type"), split=True)
    location = project.get("location") or {}
    location_text = " ".join(str(location.get(k) or "") for k in ("city", "county", "state"))
    _append_tokens(tokens, location_text, split=True)
    customer = intake.get("customer_contacts") or {}
    _append_tokens(tokens, customer.get("company_name"), split=True)
    return [tok for tok in tokens if tok]

        def _inject_tank_equipment_snippets(

    col,

    intake: Dict[str, Any],

    collected: "OrderedDict[str, Dict[str, Any]]",

    max_snippets: int,

) -> None:

    fuel_system = intake.get("fuel_system") or {}

    tanks = fuel_system.get("tanks") or []

    if not tanks:

        return
```