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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програми для автоматичного резюмування текстів із використанням методів обробки природної мови"»

на здобуття освітнього ступеня бакалавра
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 супроводжуватись посилання на відповідне джерело

_____ Артем КАНАРСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Артем КАНАРСЬКИЙ

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Канарському Артему В’ячеславовичу

1. Тема кваліфікаційної роботи: "Розробка програми для автоматичного резюмування текстів із використанням методів обробки природної мови "
керівник кваліфікаційної роботи доктор філософії (PhD) Богдан ХУДІК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки природної мови, опис методів вилучення інформації з тексту та виділення ключових слів, технічна документація з описом бібліотек для обробки природної мови.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз наявних методів і технологій обробки природної мови в контексті автоматичного резюмування текстів.
2. Проєктування застосунку для автоматизованого резюмування тексту, включаючи архітектуру, UML-діаграми та опис модульної структури.
3. Програмна реалізація та опис функціонування застосунку
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. UML-діаграма архітектури
5. Блок-схема алгоритму роботи програмного забезпечення.
6. Екранні форми.
7. Апробація результатів дослідження

6. Дата видачі завдання«23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	08.03.-15.03.2026	
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	16.03-19.04.2026	
5	Програмна реалізація застосунку	20.03-19.04.2026	
6	Тестування застосунку	20.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-03.05.2026	
8	Розробка демонстраційних матеріалів	04.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Артем КАНАРСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _52_ стор., __3__ табл., _5_ рис., _20__ джерел.

Мета роботи – оптимізація процесу обробки текстових документів шляхом автоматичного створення резюме, що забезпечує швидке отримання основної інформації та зменшує навантаження на користувача.

Об'єкт дослідження – процеси автоматичної обробки тексту та аналізу текстової інформації

Предмет дослідження – методи та технології автоматичного резюмування текстів (TF-IDF, TextRank, Semantic TextRank).

Короткий зміст роботи: У роботі здійснено огляд і аналіз методів обробки природної мови в контексті завдання автоматичного резюмування текстів. Розглянуто засоби для впровадження алгоритмів резюмування: NLTK, SentenceTransformers. Створено архітектуру додатка, програмно реалізовано ключові функціональні можливості, зокрема: завантаження текстових документів різних типів (TXT, PDF, DOCX), вибір методів резюмування, автоматичне та ручне налаштування довжини резюме, збереження отриманого резюме з можливістю обрати формат файлу. Проведено функціональне та модульне тестування додатка. У роботі використано бібліотеку NLTK для обробки текстів, Tkinter для створення користувацького інтерфейсу, PyPDF2 та python-docx для роботи з файлами, а також SentenceTransformer для семантичного аналізу. Сферою використання застосунку є автоматизація процесів обробки текстової інформації в навчальних та наукових дослідженнях, а також у практичних завданнях зі скорочення великих обсягів текстів.

КЛЮЧОВІ СЛОВА: NLP, NLTK, TF-IDF, TEXTRANK, SEMANTIC TEXTRANK, PYTHON, TKINTER.

ЗМІСТ

ВСТУП

Актуальність. Обсяги текстових даних зростає надзвичайно швидко. Наукові статті, навчальні матеріали, службова документація та інші джерела інформації формують колосальні обсяги даних, які потребують аналізу й узагальнення. Звичайна людина, якщо не володіє спеціальними засобами, не в змозі самостійно та продуктивно опрацювати таку значну кількість матеріалів. Отже виникає нагальна потреба у розробці програмного забезпечення, що здатне автоматизувати процес узагальнення текстів з метою відокремити найсуттєвішу інформацію

Мета роботи полягає у створенні програмного забезпечення, що дозволяє оптимізувати процес роботи з великими обсягами текстової інформації шляхом автоматизації її резюмування. Це дає змогу користувачеві стрімко ознайомлюватися з основною суттю документів без потреби їх повністю читати. Цей метод надзвичайно важливий у науковій, освітній та фаховій галузях, де щодня постає потреба в опрацюванні великих обсягів тексту. Автоматизація процесу резюмування сприяє скороченню часу аналізу, зростанню продуктивності та поліпшенню якості ухвалення рішень.

Об'єкт дослідження — процеси автоматичної обробки та аналізу текстової інформації.

Предмет дослідження є методи та технології автоматичного резюмування текстів, а саме статистичні (TF-IDF), графові (TextRank) та семантичні (Semantic TextRank) алгоритми, що використовуються для обробки природної мови. У межах роботи ці методи інтегровані у програмне забезпечення, яке втілює модульну структуру, гарантує завантаження файлів різних зразків, вибір методу резюмування та збереження у зручному форматі.

Наукова новизна роботи полягає у поєднанні різних підходів до підсумовування текстів в одному застосунку та можливості їх порівняння на практичних прикладах.

Це дає змогу оцінити дієвість кожного алгоритму та визначити сфери їх найбільш ефективного застосування.

Практична цінність роботи полягає у створенні програмного забезпечення, яке забезпечує швидке отримання узагальненого тексту з великих обсягів інформації. Розроблений додаток може бути використаний у навчальних процесах, наукових дослідженнях та фаховій діяльності, сприяючи економії часу й підвищенню ефективності роботи. Крім того, програма має потенціал застосування у сферах бізнес-аналітики, журналістики та медіа, де важливо швидко виділяти головне з великих обсягів даних.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ NLP

Однією з важливих галузей обробки природної мови (Natural Language Processing, NLP) є резюмування текстів. Суть цього процесу полягає у скороченні тексту зі збереженням його основного сенсу. Цей підхід дає змогу суттєво скоротити час на ознайомлення з документами та підвищити загальну ефективність роботи з великими обсягами тексту. Автоматичне резюмування має широке застосування: у науково-дослідній діяльності, освітньому середовищі, журналістиці, аналітиці та багатьох інших сферах, де критично важливим є швидке отримання інформації.

Актуальність даної теми зумовлена потребою сучасних інформаційних систем у високоякісних інструментах для швидкого аналізу текстів. Застосування алгоритмів TF-IDF, TextRank, Semantic TextRank дозволяє втілити різні підходи до підсумовування текстів: від статичного аналізу частоти слів до семантичного аналізу змісту речень. Це забезпечує системі гнучкість і спроможність працювати з різними типами текстів.

1.1 Аналіз алгоритмів та методів для обробки природної мови

У цьому дослідженні було розглянуто теоретичні підходи до обробки природної мови, здійснено аналіз наявних алгоритмів вилучення даних і формування скороченого тексту. Значний акцент зроблено на семантичні методи, що ґрунтуються на застосуванні векторних представлень і нейромережевих структур. На відміну від традиційних статистичних підходів, що беруть до уваги лише часткові ознаки слів та їх розміщення у тексті, семантичні методи дають змогу досліджувати глибинні змістовні взаємозв’язки між реченнями та уривками. Це досягається шляхом перетворення текстових складових на багатомірні векторні відображення, які демонструють змістову спорідненість між словами та фразами. Застосування моделі

SentenceTransformers посилює спроможність зіставляти текстові частини на підставі їхнього змісту, а не лише статистичних характеристик. Таким чином, текст, узагальнений за допомогою семантичного методу, є більш змістовним, коректним та наближеним до того, як його розуміє людина, оскільки він бере до уваги контекст, основну ідею та смислові зв’язки між частинами тексту.

1.2 Методи резюмування

У сучасних системах автоматичного резюмування тексту застосовуються три основні методи: статичні, графові та семантичні. Статичні базуються на статистичному аналізі частоти слів. Графові методи це розгляд тексту як мережа взаємопов’язаних речень. Семантичні методи, використовують векторні представлення слів та речень щоб враховувати їх змістовний зв’язок.

Статичні методи

Це найпоширеніший і найстаріший підхід до резюмування текстів, який з’явився ще до епохи сучасних нейромереж. Ці методи базуються на математичному аналізі частоти появи слова в тексті та його значущості в усьому тексті. До прикладу, TF-IDF простий і швидкий, але його слабким місцем є те, що він ігнорує контекст, а також зв’язки між словами. Наприклад, для цього методу слова «авто» та «машина» є абсолютно різними, незв’язними.

Графовий метод

Цей метод є логічним продовженням статичних методів, але на більш просунутому рівні аналізу. Вони сприймають весь текст як математичний граф, де є вузли це окремі речення, та ребра, тобто зв’язки між реченнями. Вони з’являються, якщо в двох або більше реченнях є спільні слова. Такий підхід, як TextRank, є найвідомішим. Він працює за таким принципом: якщо речення має багато зв’язків з іншими вагомими реченнями, то воно тим важливіше. Так він визначає не лише частоту слів, а й структуру всього тексту. Ці методи дозволяють створити якісніше скорочення.

Семантичні метод

Наразі це завершальний і найсучасніший етап розвитку методів обробки природної мови. Машина перетворює текст на багатовимірний простір, тобто комп'ютер не зберігає слово як набір літер; він приписує йому сотні чисел-характеристик, їх також називають координатами (векторами). Наприклад слово «Яблуко»:

1. наскільки це «їстівне»? (0.9);

2. наскільки це «червоне»? (0.8);

3. наскільки це «комп'ютерна компанія»? (0.5).

І на основі цього підходу і працює Semantic TextRank. Припустимо, одне речення містить слово «авто», а інше «машина», то цей метод побачить, що їх цифрові координати знаходяться поруч. Метод зрозуміє, що речення говорять про одне й те саме, і поєднає їх у графі зв'язком. Це дозволить алгоритму провести аналіз глибшого змісту.

1.3 Попередня обробка текстової інформації та інструменти NLP

Ефективність будь-якого алгоритму резюмування залежить від якості попередньої обробки тексту. Перед застосуванням методів TF-IDF, TextRank та Semantic TextRank текст має пройти певні процедури, які забезпечать коректність результатів.

Є такі основні етапи обробки:

1. Токенізація – це розбиття тексту на окремі менші частини. Це дозволяє алгоритмам працювати з доступними одиницями тексту.

2. Видалення стоп-слів – це усунення службових слів, які не несуть важливого сенсу для аналізу.

3. Нормалізація – це приведення слів до стандартної форми (наприклад, перетворення їх у нижній регістр).

4. Лематизація та стемінг – це зведення слів до базової форми або кореня. Це робиться для того, щоб зменшити кількість варіантів для одного і того самого слова.

5. Фільтрування та очищення тексту – видалення зайвих символів, пунктуації, спецсимволів, HTML-тегів.

Інструментальні засоби NLP

NLTK (Natural Language Toolkit) є основною бібліотекою для створення програм на мові програмування Python, призначенням яких є аналіз лінгвістичних даних. Вона є найважливішим інструментом у галузі обробки природної мови (NLP). Саме вона забезпечує основні етапи попередньої обробки. Ключові особливості цієї бібліотеки є:

– Величезна колекція лінгвістичних ресурсів.

– Свобода вибору алгоритмів в залежності від потреби. Це дозволяє точно налаштувати обробку під поставлені задачі.

– Модульність та прозорість.

– Бібліотека переважно працює з простими рядками та списками.

– Підтримка Machine Learning, тобто, є вбудовані класифікатори, які дозволяють швидко створити моделі для токенізації, фільтрації тощо.

Scikit-learn найпопулярніша бібліотека Python для машинного навчання. У галузі обробки природної мови вона є основним інструментом для математичного моделювання та перетворення тексту у числовий формат. Цей інструмент дозволяє реалізувати алгоритм TF-IDF, для якого не просто підраховуються частота слів, а й математично зважується значущість у межах усього документа, що є найважливішим для статистичних методів резюмування.

SentenceTransformers - спеціалізований фреймворк на основі Python, створений для генерації високоякісних векторних представлень абзаців, текстів. Бібліотека використовує сіамську архітектуру нейронних мереж це такий тип структури, що складається з двох підмереж, які працюють паралельно. Такий підхід дозволяє моделі ігнорувати поверхневі відмінності в написанні слів (наприклад, синоніми) та зосередитися на змісті інформації. Саме на цих

векторних представленнях працює Semantic TextRank. Він будує граф із зв’язками між реченнями, спирається на семантичну близькість речень, яку було обчислено сіамськими мережами. Оскільки кожне речення вже має представлення як точка у багатовимірному просторі, він математично визначає саме ті речення, які є найзмістовнішими.

SpaCy є сучасною бібліотекою для обробки природної мови, була розроблена спеціально, для використання в проєктах де важлива максимальна швидкість та робота з великими обсягами даних. Головні особливості даної бібліотеки:

– Максимальна швидкість завдяки написанню на Cython.

– Готові нейромережові моделі: надаються попередньо навчені моделі для багатьох мов.

– Об’єктно-орієнтований підхід: працює з об’єктами, такими як Doc, Token та Span.

– Дозволяє легко створювати ланцюжки для автоматичної обробки текстів.

PyPDF2 популярний інструмент для роботи з PDF-файлами. Оскільки формат PDF є досить складним, ця бібліотека фокусується на низькорівневих операціях з документами. А саме: вилучення тексту, маніпуляція зі сторінками, підтримка роботи з паролями, тобто шифрування та розшифрування файлів, а також зчитування інформації про файл (назва, дата створення тощо).

Python-docx спеціалізований інструмент для роботи з форматом Microsoft Word. Працює зі структурою документа, а саме: абзацами, заголовками, таблицями тощо. Дозволяє не тільки читати, а й створювати нові документи з нуля або редагувати наявні. Дозволяє змінювати шрифти, розміри, кольори тощо. Також він має зручні інструменти для роботи з таблицями.

ReportLab інструмент для генерації PDF-файлів. Він забезпечує створення документів із текстовим наповненням, можливістю форматування та додавання графічних елементів. У застосунку ReportLab реалізує функцію експорту скороченого тексту у форматі PDF, що дозволяє користувачеві зберігати отримане резюме у зручному для нього форматі.

1.4 Практичне застосування методів NLP у задачі резюмування

Методи обробки природної мови мають широке застосування в різних сферах, проте особливе значення вони мають саме в задачах автоматичного резюмування тексту. У цьому підрозділі буде розглянуто, які алгоритми можуть бути використані на практиці та які результати вони здатні забезпечити.

Використання методу TF-IDF

TF-IDF - статистичний метод, який оцінює, наскільки значущим є слово в межах одного тексту. Основна ідея полягає в тому, що слово має більшу вагу, якщо воно часто зустрічається в конкретних реченнях, але рідко в інших текстах.

Сам метод складається з двох частин:

– TF (Term Frequency): Частота слова. Визначає, як часто слово зустрічається в межах одного документа. Чим більше повторень, тим вищий показник значущості отримує це слово для даного тексту.

– IDF (Inverse Document Frequency): Показник унікальності. Він зменшує значущість слів, які зустрічаються майже в кожному реченні. Завдяки цьому стає можливим виокремити унікальні та специфічні терміни.

Ключовими перевагами методу

Виявлення тематичної специфіки: Метод виконує функцію фільтрації, який автоматично відсіває загальновживані слова, що не несуть змісту, і фокусується на термінах, які визначають суть документа. Наприклад, у технічній документації слово «система» може зустрічатися часто і мати низьку вагу, тоді як термін «мікроконтролер» отримає високий бал, чітко вказуючи на тематику матеріалу.

Висока продуктивність та масштабованість: Завдяки математичній простоті алгоритм працює надзвичайно швидко. Він не потребує величезних обчислювальних потужностей, що дозволяє миттєво обробляти величезні обсяги інформації від локальних архівів до баз даних із безліччю новинних статей, технічних засобів тощо.

Прозорість та легкість застосування: TF-IDF дає повністю зрозумілий результат: можна простежити, як саме формувалась оцінка кожного слова, чи воно стало вагомим через часті повторення в тексті, чи завдяки своїй унікальності для всього обсягу тексту.

Економічність у навчанні: Метод не потребує тривалого навчання на наборі даних. Він дозволяє запустити його у будь-якому тексті, і він миттєво одразу почне видавати головні слова, виключно базуючись на поточному тексті.

Сфери застосування TF-IDF

Пошукові системи та інформаційний пошук: Це класичний спосіб для визначення, які сторінки в інтернеті найкраще відповідатимуть запиту. Коли користувач вводить ключові слова, алгоритм шукає самі ті вебресурси, які мають найбільш значущими. Це дозволяє системі ставити на перше місце не просто тексти, де слово може згадуватися випадково, а найбільш доречні матеріали, що максимально відповідатимуть запиту користувача.

Кластеризація та групування контенту: Цей метод автоматично розподіляє великі потоки інформації за темами. Оскільки TF-IDF перетворює текст на набір цифр, що дозволяє комп’ютеру легко і швидко порівнювати різну інформацію між собою. Це застосовується для автоматичного об’єднання схожих новин у стрічці, сортування звернень клієнта за категоріями або швидкого виявлення повторів і дублікатів у базі даних.

Персоналізовані рекомендаційні: На основі методу працюють рекомендації контенту. Якщо користувач шукатиме статтю на певну тему, скажемо, про «квантову фізику», алгоритм шукатиме інші матеріали, які мають схожу значущість. Таким чином, система може пропонувати схожі за змістом статті, відео або товари, базуючись на аналізу інших текстового опису.

Підготовка даних для машинного навчання: TF-IDF є незамінним етапом оцифрування тексту перед навчанням моделей. Оскільки більшість моделей машинного навчання не розуміють слова, а працюють лише з числами, перетворення тексту дозволяє передати моделям інформацію про значущість кожного слова, що значно підвищує точність класифікації.

Автоматичне створення анотацій: Метод допомагає виділити найбільш значущі речення в тексті. Обираючи речення, які містять слова з найвищим показником, що дозволяє автоматично сформувати короткий конспект, що передає основну суть.

Обмеження TF-IDF

Попри простоту та швидкість роботи, метод має низку суттєвих обмежень, які знижують його ефективність.

Ігнорування контексту та багатозначності: TF-IDF аналізує слова як окремі одиниці, не розуміючи їхнього значення. До прикладу, він не розрізняє слово «замок» як архітектурну споруду або механізм у дверях. Якщо слова мають однакову частоту, алгоритм вважатиме їх ідентичними за важливістю, навіть якщо контекст документу зовсім інший.

Втрата порядку слів: Метод не враховує, у якій послідовності стоять слова. Для нього речення «Компанія купує стартап» та «Стартап купує компанію» будуть однаковими за значенням, оскільки набір частот слів у них збігається. Через це може втратить логіка складних речень.

Чутливість до синонімів: Алгоритм не може розуміти, що різні слова можуть означати одне й те саме значення. Якщо в одному тексті вживають термін «автомобіль», а в іншому «машина», TF-IDF сприйматиме їх як два абсолютно різних поняття. Це може призвести до того, що схожі за змістом тексти отримають різні оцінки та не будуть згруповані разом.

Проблема довгих документів: У дуже великих текстах слова зазвичай повторюються частіше, що може штучно завищувати показник значущості. Хоч існують способи нормалізації, довгі та розмиті за тематикою статті часто дають менш точні результати порівняно з короткими та спеціалізованими матеріалами.

Залежність від обсягу: Ефективність значною мірою залежить від обсягу тексту. У випадку обробки невеликих наборах даних показники можуть бути спотворені, через брак статистичної вибірки, що може до помилкового виділення другорядних речень і, як наслідок призведе до значного зниження якості скороченого тексту.

Використання TextRank:

TextRank - це алгоритм обробки тексту, заснований на графах, який використовується для автоматичного виявлення ключових слів та створення стислих анотацій. Основна ідея полягає в тому, що текст розглядається як мережа, де слова або речення це вузли, а зв'язки між ними це ребра. Алгоритм визначає найбільш значущі елементи за принципом, чим більше зв'язків має слово з іншими важливими словами, тим вищою є його вага.

Метод базується на двох основних принципах:

– Локальний контекст: Алгоритм аналізує слова, що стоять поруч одне з одним у певному діапазоні, наприклад, в межах 2-5 слів. Якщо два слова часто з'являються поряд, між ними встановлюється міцний зв'язок.

– Авторитетність вузлів: Вага слів залежить не від кількості їхніх зв'язків, а й від якості цих зв'язків. Слова стають важливими, якщо вони пов'язані з іншими словами, які вже визнані значущими в цьому тексті.

Ключовими перевагами методу TextRank

Незалежність від контексту: Найбільша перевага TextRank є здатність працювати одразу з одним конкретним текстом. Йому не потрібна велика база даних чи попередньо навчені моделі, щоб зрозуміти, які слова є важливими. Це робить його ідеальним інструментом для аналізу унікальних документів, рідкісних тем або нових текстів, аналогів яких ще немає в системі.

Розуміння внутрішньої структури тексту: На відміну від методів, які рахують кількість слів, TextRank аналізує їхні взаємозв'язків. Він враховує, які терміни стоять поруч і як вони формуються змістовні пари. Це дозволяє методу знаходити не просто часто знаходити не просто часто вживані слова, а справжній центр текст, навколо якого будується вся розповідь.

Продуктивність у створенні скорочень: TextRank є одним з найкращих інструментів для екстрактивного резюмування. Він вміє оцінювати цілі речення за їхньою важливістю в межах тексту і обирати з них ті, що містять найбільшу інформативність. У результаті отримуємо стислий конспект із тексту, який складається з найінформативніших речень.

Відсутність потреби у словниках та мовних базах: Алгоритм базується на математичній структурі графа, він є універсальним для будь якої мови. Йому не потрібно знати граматику чи мати готові стоп-слів (хоча їх використання може покращити результат), визначається важливість елементів на основі розташування в документі.

Стійкість до повторів: На відміну від методів, що базуються лише а підрахунку, TextRank оцінює важливість слів через його взаємодію між іншими частинами тексту. Слова не стають ключовими лише тому, що вони часто зустрічаються; вони мають бути пов’язані з різними значущими словами. Це дає методу точніше визначити головну тему, ігноруючи слова, які часто зустрічаються в тексті, але не мають смислового навантаження.

Сфери застосування TextRank

Автоматичне резюмування текстів: Це основна сфера використання даного методу. TextRank дозволяє швидко стиснути велику статтю, звіт до кількох ключових речень. Завдяки вибору найбільш значущих речень з оригінального тексту, що дозволяє читачеві миттєво зрозуміти головну суть без необхідності читати весь текст.

Генерація ключових слів: Алгоритм ідеально підходить для автоматичного створення списку ключових слів, чудово підходить для блогів, новинних порталів або наукових праць. Метод здатний побачити зв’язки між термінами, він вибирає слова, які найкраще описують тематику документа.

Аналіз новин: У великих медіа-моніторингових системах цей метод можуть використовувати для швидкого виділення головних подій дня. Він дозволяє автоматично формувати заголовки або короткі анонси для стрічки новин.

Обмеження TextRank

Вища обчислювальна складність: Побудова графа та розрахунок зв’язків між усіма словами або реченнями потребує значно більше ресурсів, ніж просто підрахунок. На дуже великих текстах робота алгоритма може суттєво сповільнитися.

Залежність від обсягу вхідних даних: Метод погано працює з дуже короткими текстами. Якщо слів замало, алгоритм не може побудувати достатню кількість зв’язків, щоб об’єктивно визначити значущість того чи іншого слова.

Чутливість до структури речень: TextRank сильно залежить від того, наскільки якісно написано текст. Якщо речення побудовані хаотично або логічні зв’язки між абзацами слабкі, алгоритм може вибрати другорядні фрази як ключові, просто через їхнє випадкове розташування поруч із іншими словами.

Відсутність розуміння глобального контексту: Оскільки метод аналізує лише один документ, він не знає, які слова є рідкісними чи важливими. До прикладу, якщо в статті про медицину часто зустрічається слово «пацієнт», буде зустрічатись в кожному реченні і буде пов’язане з усіма іншими термінами, TextRank може визначити його як головне слово, хоча для фахівців воно буду загальним і не матиме великої важливості.

Проблема розуміння синонімів: Як і більше класичних методів, TextRank сприймає слова «машина» і «авто» як два різних вузли графа. Через це значущість теми може розділитись між різними синонімами, і жодне з таких слів не отримає достатньо високий бал, щоб стати ключовим. Це може призвести до того, що по справжньому важлива тема може залишитись непоміченою або недостатньо поміченим.

Використання Semantic TextRank

Semantic TextRank це вдосконалена версія класичного алгоритму, яка поєднує структурний аналіз графів із потужністю сучасних нейромережевих моделей. Якщо звичайний TextRank бачить лише слова, то Semantic TextRank може розуміти їх сенс. Метод використовує семантичну близькість слів, для побудови зв’язків у тексті. Замість того, щоб просто з’єднувати слова, які стоять поруч, цей метод аналізує, наскільки близькими вони є за значенням, навіть якщо вони пишуть зовсім інакше.

Основна ідея полягає у використанні векторних представлень. Кожне слово або речення перетворюється на числовий вектор у багато вимірному

просторі. Відстань між цими поняттями показує, наскільки схожими є поняття за змістом. Особливості використання алгоритму:

Семантичні зв’язки: Алгоритм будує графи, де зв’язки базуються на логічній схожості. Наприклад, слова «космос» і «всесвіт» будуть мати міцний зв’язок, навіть якщо вони будуть знаходитись в різних частинах.

Врахування контексту: Якщо в тексті зустрічається слово «кран», алгоритм подивиться на сусідні слова, якщо поруч слово «вода» він зрозуміє, що це про сантехніку, а якщо «будівництво» про техніку. Це дозволяє не мішати різні теми в одну купу.

Фільтрування важливого: Метод вміє ігнорувати слабкі, випадкові зв’язки. Він фокусується на тих словах, які мають сильний смисловий зв’язок із головною темою всього тексту. Це дає можливість отримати дуже чистий список ключових слів без зайвого.

Ключові переваги Semantic TextRank

Успішна робота із синонімами: Є головною перевагою методу. Оскільки алгоритм розуміє, значення слів це дозволяє не втрачати важливу тему, якщо в тексті було використані різні назви для одного і того самого. Метод об’єднає їхню ваг, що дозволить залишити тему незмінною.

Висока точність у складних текстах: Semantic TextRank чудово розуміє професійні статті, наукові праці, або технічною документацією. Метод бачить логічні зв’язки між специфічними термінами, які звичайні алгоритми могли б просто проігнорувати.

Якісне стиснення тексту: Коли метод створює скорочення статті, він обирає не за кількістю знайомих слів, а за їхньою насиченістю. У результаті отримується резюме, яке передає головну думку, а не просто набір фраз.

Стійкість до зміни формулювань: Якщо стиль написання тексту буде змінений, але збережеться його зміст, метод видасть схожий результат. Це робить Semantic TextRank надійним інструментом для аналізу тексту, написаних, різними авторами на одну й ту саму тему.

Зменшення шуму в тексті: Оскільки метод може розуміти контекст, метод рідше помиляється і не заносить випадкові слова до важливих, які можуть часто зустрічатись в тексті, але не стосуватись суті.

Обмеження Semantic TextRank

Вищі вимоги до обчислювальних можливостей: Метод використовує нейромережі для обчислення значень слів, йому потрібно значно більше оперативної пам’яті та потужності процесора. Обробка може тривати в рази довше, ніж класичний TextRank.

Залежність від якості нейромоделей: Точність методу напряму залежить від того, наскільки якісну словесну базу було використану. Якщо модель погано знає специфічну мову, або професійні терміни, вона почне робити помилкові зв’язки, що може призвести до втрати сенсу.

Складність перевірки: На відміну від простого підрахунку слів, не завжди можна пояснити, чому алгоритм обрав, що речення подібні. Якщо була зроблена помилка в логіці, знайти її причину всередині складних математичних векторів буває важко.

Проблема обробки рідкісних слів: Якщо в тексті використано унікальні слова, нові сленгові вирази або специфічні коди, яких не було в базі навчання нейромережі, метод не зможе визначити їхній зміст. У такому разі метод може спрацювати гірше ніж звичайний пошук за словами.

Складність налаштування: Для запуску Semantic TextRank не достатньо просто завантажити бібліотеку. Потрібно підібрати правильну модель, налаштувати її та підготувати середовище. Це робить його складнішим для швидкого впровадження в порівнянні з іншими методами.

2. ПРОЄКТУВАННЯ ТА ЗАСОБИ РЕАЛІЗАЦІ

Проєктування програмного забезпечення(ПЗ) є одним з основних етапом у процесі його створення, так як на цьому рівні визначається структура системи, взаємодія модулів та засоби реалізації функціональних можливостей. Для задачі автоматичного резюмування текстів важливе значення має вибір архітектури, що дасть змогу легко адаптувати, розширювати та додавати нові алгоритми обробки природної мови.

У цьому розділі розглядаються вимоги до програмного забезпечення, визначається модульна частина застосунку та описуються інструменти, що використовуються для його реалізації. Значна увага була сфокусована на розробці, відображенні взаємодії між складовими частинами системи, а також на описі логіки роботи програми, що послідовно демонструє ключові етапи виконання операцій.

Мета цього розділу полягає, в обґрунтуванні архітектурні підходи та засоби впровадження, які будуть гарантувати продуктивність роботи програми, призначені для автоматичного резюмування текстів.

2.1 Вимоги до ПЗ

Функціональні вимоги

Вимоги визначають набір операцій, які має виконувати програма для автоматичного резюмування текстів. Вони описують, що саме користувач може робити із застосунком, які формати підтримує та які варто очікувати результати. У таблиці (нижче) наведено основні функціональні можливості системи із зазначенням головних характеристик та параметрів роботи.

Таблиця 2.1

Функціональні вимоги

№	Функціональна вимога	Опис	Параметри/Обмеження
1	Завантаження тектсу	Підтримка різних форматів документу	Імпорт файлів TXT, DOCX, PDF; або вставка через буфер обміну
2	Перевірка коректності файлів	Перевірка завантажених документів на відповідність формату чи пошкодження	Перевірка форматів файлів, відсутність пошкоджень
3	Керування списку стоп-слів	Перелік слів, які мають видалятись під час генерування резюме	Способи задання: ручне введення, завантаження з файлу. Можливість повного вимкнення функцій видалення стоп-слів
4	Вибір параметрів резюмування	Налаштування перед початком обробки тексту	Встановлення рівня стиснення, задання кількості речень, вибір алгоритму
5	Генерація та виведення результату	Використання методу резюмування та відображення згенерованого тексту	Відображення результату у вікні графічного інтерфейсу програми
6	Збереження результату	Експорт резюме у різні формати	TXT, DOCX, PDF

Нефункціонаональні вимоги

Якість функціонування додатку та його межі визначаються нефункціональними вимогами. Вони не стосуються окремих дій, які виконуватиме система, а встановлюють значення, за якими буде вимірюватись продуктивність, надійність та зручність використання додатку. Стосовно програми для автоматичного резюмування текстів, важливим стають вимоги до захисту інформації, швидкодії, здатність працювати без постійного підключення до мережі інтернет та стабільність роботи.

Таблиця 2.2

Нефункціональні вимоги

№	Нефункціональна вимога	Опис	Параметри/Обмеження
1	Безпека даних	Локальна обробка без передачі інформації у мережу	Повна локальна обробка даних у межах ПК користувача
2	Надійність	Стабільна робота з різними форматами файлів	Обробляти винятки при відкритті пошкоджених або некоректних файлів; повідомлення про помилки
3	Швидкість обробки	Час виконання алгоритмів резюмування.	TF-IDF: < 1сек., для текстів для 10000 слів; textRank, < 1сек., для текстів для 10000 слів; semantic TextRank. ~10 сек., для текстів до 5000 слів
4	Автономність	Повна функціональність без доступу до Інтернету	Відсутність залежності від серверів чи онлайн- сервісів; локальна обробка даних

Висновки

Підсумовуючи вищенаведене, вимоги до програмного забезпечення для автоматичного резюмування тексту охоплюють дві категорії: функціональні та нефункціональні. Функціональні: вимоги окреслюють набір можливостей, які має виконувати система: прийом файлів різних форматів, створення резюме їх збереження тощо. Вони відповідають за втілення головних можливостей, які очікуватиме користувач від застосунку. Натомість нефункціональні вимоги диктують критерії якості роботи програми: рівень захищеності інформації, надійність, швидкість обробки даних та здатність працювати без доступу до інтернету. Саме ці критерії забезпечують те, що програма буде не лише функціональною, але й продуктивною, стабільною під час роботи та комфортною у взаємодії.

Поєднання цих вимог формує основу для деталізації архітектури та побудови модульної структури застосунку, що стане предметом обговорення у наступному розділі.

2.2 Архітектура та модульна структура

Основа програмного забезпечення, що задає його структуру, організацію, взаємодію складових частин та потенціал для розвитку. Для системи, спроектованої для автоматичного скорочення змісту текстів, критично важливим є застосування модульної структури, яка гарантує адаптивність, здатність до розширення функціональності та простоту підтримки. Завдяки цьому стає можливим впровадження нових алгоритмів обробки природної мови, що дасть змогу розширити функціональність без значних змін у базовій структурі.

У цьому підрозділі буде розглянуто архітектурна модель застосунку, яка охоплює основні модулі: завантаження файлів, попередня обробка тексту, резюмування, інтерфейс користувача та збереження результатів. Для наочності буде продемонстроване UML-діагрма, яка ілюструватиме зв'язки між компонентами, а також детально описано послідовність операцій, які виконує система.

Завданням цього підрозділу є обґрунтування обраного модульного підходу до архітектури та демонстрації його переваг у межах хавдання автоматичного резюмування текстів. Це створює основу для подальшої програмної розробки та тестування програмного продукту

Основні принципи архітектури

Архітектура програмного забезпечення для автоматичного резюмування текстів ґрунтується на кількох ключових принципах, які забезпечує продуктивність, гнучкість та надійність системи.

Модульність означає поділ системи на окремі компоненти, кожен з яких відповідає за конкретний етап обробки тексту. Такий підхід дозволяє:

– ізолювати функціональність: завантаження файлів, попередня обробка, резюмування, інтерфейс;

– спрощувати тестування та налагодження окремих частин;

– можливість легкого розширювання, додавання нових алгоритмів резюмування чи зміна інтерфейсу без впливу на іншікомпоненти.

Гнучкість

Гнучкість архітектури полягає у можливості інтеграції нових алгоритмів та інструментів без необхідності змінювати основну структуру. Наприклад модуль резюмування може підтримувати кілька алгоритмів (TF-IDF, TextRank, Semantic TextRank), і згодом матиме змогу бути розширеним новими методами без потреби перебудовувати всю систему.

Прозорість передбачає взаємодію складових частин системи, що є цілком очевидною. Кожен модуль повинен мати чітко визначені параметри, що приймається на вхід, та того, що видаватиме на виході. Це дає можливість без зусиль прослідкувати весь шлях опрацювання тексту, починаючи з моменту завантаження файлу і закінчуючи формування готового резюме. Це робить структуру системи зрозумілою, як розробникам так і користувачам, які здатні побачити логіку функціонування програми через інтерфейс.

Стабільність архітектури досягається завдяки обробці помилок та використанню перевірених бібліотек. Передбачається, що система повинна коректно реагувати на невалідні чи пошкоджені файли, сповіщати користувача про помилку у зручному форматі, який легко сприймається та гарантувати стабільну роботу, навіть при великих обсягах даних. Додатково, надійність невіддільна від безпеки даних, що полягає у виконанні усіх операцій, без жодної передачі інформації у мережу.

2.2.1 Модульна структура

Модуль завантаження файлів

Модуль призначений для застосовується для ввідкриття документів у форматах TXT, DOCX, PDF. Після завантаження файлу його зміст передається до системи, де буде проходити подальшу обробку.

Для роботи з кожним підтримувальних форматів реалізовані окремі функції:

1. *load_text_file(path)* – читання текстових файлів.

2. *load_pdf_file(path)* – читання PDF-документів за допомогою PDF.

3. *load_docx_file(path)* – читання текстових файлів DOCX.

Модуль має кілька важливих особливостей:

— Модуль виконує лише технічне читання файлів.

— Перевірка формату та повідомлення про помилки реалізовані у графічному інтерфейсі *Main.py*.

— Якщо формат файлу не підтримується, користувач отримує повідомлення «Формат файлу не підтримується».

— У випадку помилки (пошкодження файлу чи некоректний формат) користувач отримує повідомлення «Не вдалося прочитати файл».

Нижче описано, як відбувається процес завантаження файлу, від натискання кнопки до відображення тексту в інтерфейсу:

1. Користувач натискає кнопку «Завантажити файл» у графічному інтерфейсі.

2. Викликається метод *open_file()* у класі *TextSummarizerGUI*.

3. Відкривається діалогове вікно *filedialog.askopenfilename(),* де користувач обирає документ.

4. Метод *open_file()* перевіряє формат файлу:

— .txt виклик *load_text_file(path)*

— .pdf виклик *load_pdf_file(path)*

— .docx виклик *load_docx_file(path)*

— Якщо інші файли виводиться повідомлення про помилку.

5. Вміст файлу передається у змінну *text*.

6. Текст відображається у полі введення програми.

7. Якщо виникає виняток, користувач отримує повідомлення про помилку через *messagebox.showerror()*.

Модуль попередньої обробки тексту

Перед тим як текст потрапить на аналіз, він проходить етап підготовки. Цей модуль відповідає за токенізацію, нормалізацію та видалення слів, які не несуть смислового навантаження і лише заважатимуть точному аналізу. Така

підготовка дозволяє суттєво зменьшити «шум» у даних і забезпечити коректну роботу алгоритмам обробки(TF-IDF, TextRank, Semantic TextRank). Функції модуля:

1. Токенізація речень розбиття тексту на окремі речення за допомогою *nltk.sent_tokenize()*.
2. Видалення стоп-слів використання зовнішнього файлу *stopwords_ua_list.txt* для очищення речень від малозначущих слів.
3. Нормалізація тексту приведення речень до вигляду, придатного для подальшої векторизації та аналізу.

Модуль має кілька практичних особливостей, які роблять його зручним і використанні:

— Список стоп-слів зберігається у зовнішньому файлі, що дозволяє легко його редагувати та адаптувати, для української мови.

— Попередня обробка тексту інтегрована у функції резюмування *(summarize_text, summarize_textrank, summarize_semantic_textrank).*

— Виконується автоматично при виклику алгоритму резюмування через інтерфейс.

Нижче описані покроковий процес, як система обробляє текст від введення до отримання готового результату:

1. Користувач вводить текст або завантажує документ.
2. Викликається метод *make_summazy()* у класі *TextSummarizerGUI.*
3. Текст передається у функцію резюмування *(summarizer.py).*
4. Виконується токенізація речень *(nltk.sent_tokenize).*
5. Застосовується функція *preprocess_sentences()*, яка видаляє стоп-слова на основі завантаженого списку користувача або використовує вбудований список *stopwords_ua_list.txt.*
6. Очищені речення передаються у вибраний алгоритм резюмування (TF-IDF, TextRank, Semantic TextRank).
7. Результат повертається у вигляді скороченого тексту.

Модуль резюмування

Модуль відповідає за формування скороченого тексту на основі вхідного тексту чи документу. Модуль реалізує кілька алгоритмів резюмування, що дозволяє порівнювати їхню ефективність та адаптувати результат під різні потреби. В модулі лежать п’ять ключових функцій:

– *summarize_text()* – резюмування на основі TF-IDF.

– *summarize_textrank()* – резюмування методом TextRank.

– *summarize_semantic_textrank()* – семантичне резюмування на основі *SentenceTransformers (Semantic TextRank).*

– *_select_sentences_adaptive()* – адаптивний вибір речень за вагою та покриття тексту.

– *pagerank_from_similarity()* – обчислення PageRank для матриці схожості речень.

Модуль має кілька особливостей, що роблять його зручним інструментом для роботи з тексту:

– Підтримка вибору довжини резюме (рівень стиснення, за кількістю речення).

– Можливість використання різних алгоритмів для порівняння результатів.

– Інтеграція з модулем попередньої обробки (токенізація, видалення стоп-слів).

– Використання бібліотек NLTK, scikit-learn, SentenceTransformers.

Нижче наведено повний цикл обробки, від введення тексту до отриманого результату:

1. Користувач вводить текст або завантажує документ.

2. Викликається метод *make_summazy()* у класі *TextSummarizerGUI.*

3. Текст передається у функцію резюмування *(summarizer.py).*

4. Виконується токенізація та попередня обробка.

5. Залежно від вибору користувача застосовується один з алгоритмів:

– TF-IDF

– TextRank

– Semantic TextRank.

6. Вибрані речення об’єднуються в скорочений текст.

7. Результат повертається у графічний інтерфейс для відображення.

Модуль інтерфейсу користувача

Забезпечує взаємодію користувача із системою через графічний інтерфейс, реалізований на основі бібліотеки Tkinter. Це дозволяє завантажити документи, вводити текст, обирати алгоритм резюмування, налаштовувати параметри та переглядати результати. В модулі такі основні функції:

1. Завантаження файлів – виклик методу *open_file()*, взаємодія з модулем *file_loader.py*.

2. Введення тексту – поле введення тексту.

3. Вибір алгоритмів резюмування – елемент керування для вибору між TF-IDF, TextRank, Semantic TextRank.

4. Налаштування параметрів – вибір кількості речень, рівня втиснення.

5. Відображення результатів – поле для показу скороченого тексту.

6. Повідомлення про помилку – використання *messagebox.showerror()* для інформування користувача.

Модуль має такі особливості:

1. Побудова інтерфейсу на основі бібліотеці Tkinter, що забезпечує стандартні елементи керування які систем Windows.

2. Усі основні дії винесені у вигляді кнопок («Завантажити файл», «Зробити резюме», «Зберегти результат» тощо).

3. Налаштування параметрів стиснення через повзунок.

4. Перевірка форматів файлів та обробка винятків реалізовані на рівні *GUI*, отримання повідомлення через *messagebox*.

Нижче описаний сценарій взаємодії користувача з програмою, від запуску до збереження результату:

1. Користувач запускає програму.

2. У головному вікні доступні кнопки: «Завантажити файл», «Зробити резюме», «Зберегти результат» та інші.

3. При натисканні «Завантажити файл» викликається метод *open_file()*, який інтегрується з модулем *file_loader.py*.

4. При натисканні «Зробити резюме» викликається метод *make_summary()*, який передає текст у модуль *summarizer.py*.

5. Результат резюмування відображення у полі виводу.

6. При натисканні «Зберегти результат» викликаючи метод *save_summary()*, який зберігає текст у форматах TXT, DOCX, PDF.

7. У випадку помилки користувач отримує повідомлення через messagebox.

Модуль збереження результатів

Забезпечує збереження отриманого резюме у зручних для користувача форматах TXT, DOCX, PDF. Це дає можливість легко зберігати результати роботи програми для подальшого використання чи пошерення.

Для кожного з підтримувальних варіантів реалізований свій підхід:

1. Збереження у TXT – звичайний запис тексту у файл з кодуванням UTF-8.

2. Збереження у PDF – створення PDF-документів за допомогою бібліотеки ReportLab, з підтримки кирилиці.

3. Збереження у DOCX – формування документів Word через python-docx.

Модуль розроблений з урахуванням для зручності користувача:

1. Користувач сам обирає формат у діалоговому вікні *asksaveasfilename()*.

2. Автоматично пропонується ім’я файлу но основі завантаженого документа *filename_resume.txt* .

3. У випадку помилки користувач отримує повідомлення «Не вдалось зберегти файл».

4. Якщо резюме ще не згенеровано, система попереджає користувача повідомленням «Спочатку згенеруйте резюме!».

Нижче описаний процес збереження результату, від натискання кнопки до появи підтверджуючого повідомлення:

1. Користувач натискає кнопку «Зберегти результат» у графічному інтерфейсі.

2. Викликається метод *save_summary()*.

3. Перевіряється, чи є згенероване резюме у полі *summary_output*.

– Якщо текст відсутній виводиться повідомлення «Спочатку згенеруйте резюме!».

4. Відкривається діалогове вікно для вибору місця та формату зберігання.

5. Залежно від вибору користувача буде створено файл у форматах TXT, DOCX чи PDF.

6. Після успішного збереження користувач отримує повідомлення «Файл збережено».

7. У випадку винятку викликається блок *except*, і користувач отримує повідомлення про помилку.

2.2.2 Діаграма класів системи

На рис. 2.2 (нижче) наведено діаграма класів, вона відображає архітектуру системи автоматичного резюмування текстів. Діаграма демонструє взаємодію між основними компонентами: графічним інтерфейсом користувача, модулем завантаження файлів та модулем резюмування.

Клас TextSummarizerGUI реалізує інтерфейс користувача та роботу всієї програми. Він містить змінні для збереження параметрів налаштування алгоритмів (такі як metod_var, coverage-var, max_sent_var та інші), елементи керування (custom_stop_entry та custom_stop_label) та методи для виконання основних функцій:

– *open_file()* – забезпечує процес завантаження файлів;

– *make_summary()* – запускає процес резюмування тектсу;

– save_summary() – реалізує збереження згенерованого резюме;

– copy_summary() – забезпечує копіювання результату в буфер обміну;

– toggle_auto() – керує логікою перемикання між автоматичним визначенням довжини резюме та заданням кількості речень;

– *load_custom_stopwords_file()* та *toggle_stopwords_entry()* – відповідають за налаштування та керуванням списком стоп слів.

Для покращення досвіду користувача під час роботи з додатком, клас TextSummarizerGUI взаємодіє з допоміжним класом ToolTip. Графічний інтерфейс може керувати об’єктами підказки, які використовується для відображення інформації щодо елементів меню, через методи show_tip(event) та hide_tip(event).

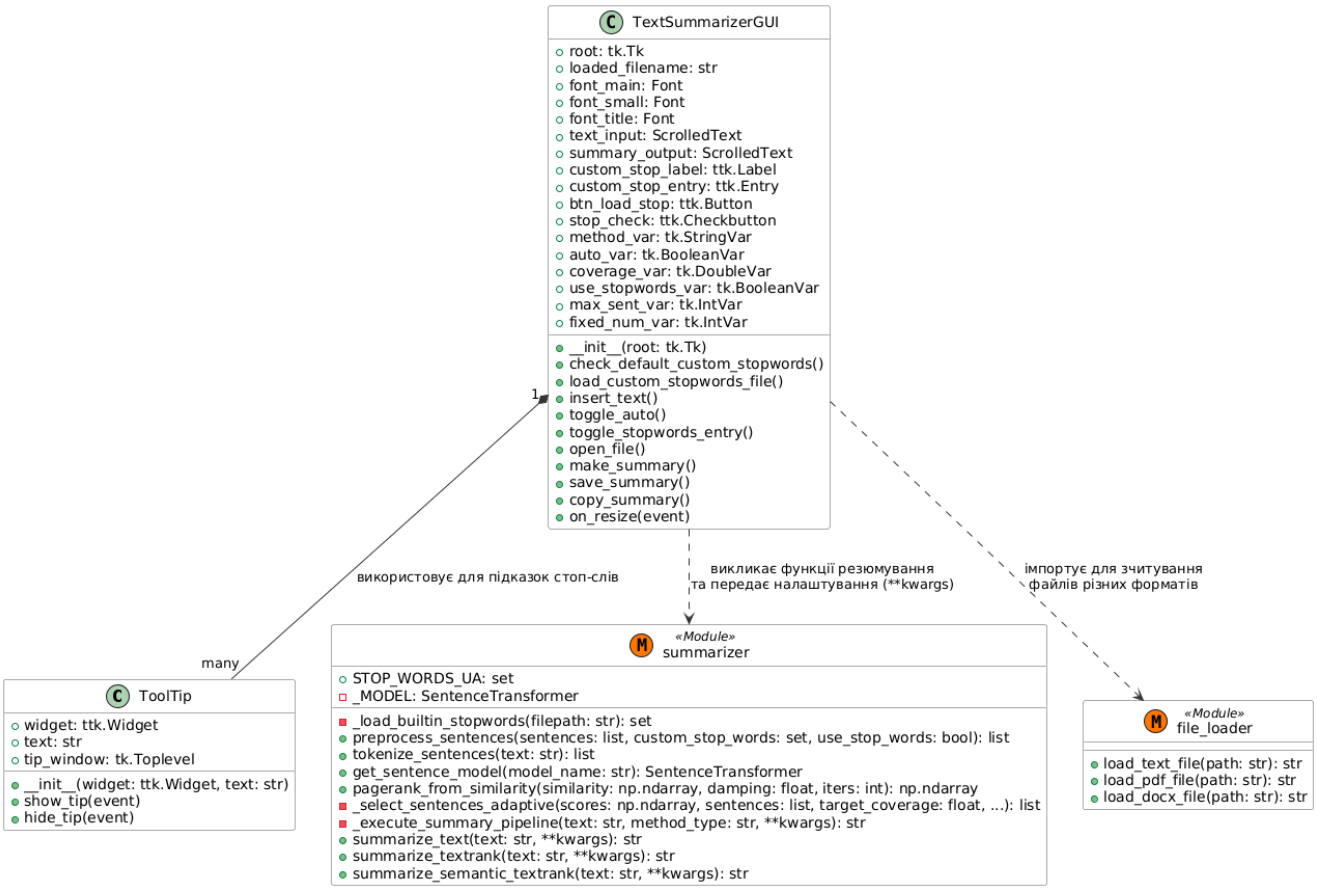


Рис. 2.2. Діаграма класів системи автоматичного резюмування текстів

2.3 Алгоритм роботи застосунку

Робота застосунку автоматичного резюмування текстів побудована на послідовності виконання етапів обробки даних від завантаження документа до формування та збереження скороченого тексту. Такий підхід забезпечує логіку структурованого процесу та узгодження між модулями системи.

На першому етапі користувач вводить текст чи завантажує файл у форматах TXT, DOCX, PDF, далі графічний інтерфейс перевіряє коректність формату та передає шлях до файлу у модуль завантаження:

– Якщо формат не підтримується, система повідомляє про помилку;

– Якщо формат підтримується, система переходить до зчитування вмісту документа.

Наступний етап – обробка та налаштування параметрів стоп-слів. Перед запуском методів резюмування, перевіряється стан «Використовувати стоп-слова»:

– Якщо параметр вимкнений, етап очищення пропускається.

– Якщо використання стоп-слів активоване, система виконує перевірку. наявності користувацького списку. За умови заповнення поля або завантаженння зовнішнього файлу користувача, вбудований словник ігнорується, а видалення відбувається виключно за наданими стоп-словами. У тому випадку, якщо поле залишається порожнім, система за замовчуванням підтягує свій список стоп-слів.

Вибір методу та визначення обсягу резюме. Користувач обирає параметри стиснення, обираючи один з трьох методів. Після визначається режим формування довжини резюме: автоматичний або фіксований.

Виведення та збереження результатів. Після створення скорочений текст відображається у графічному вікні. Користувач може скопіювати текст у буфер обміну або зберегти його у файл формату TXT, DOCX чи PDF.

На рис. 2.3 (нижче) наведено блок-схему алгоритму роботи застосунку, яка демонструє послідовність виконання головних операцій від завантаження до збереження готового резюме.

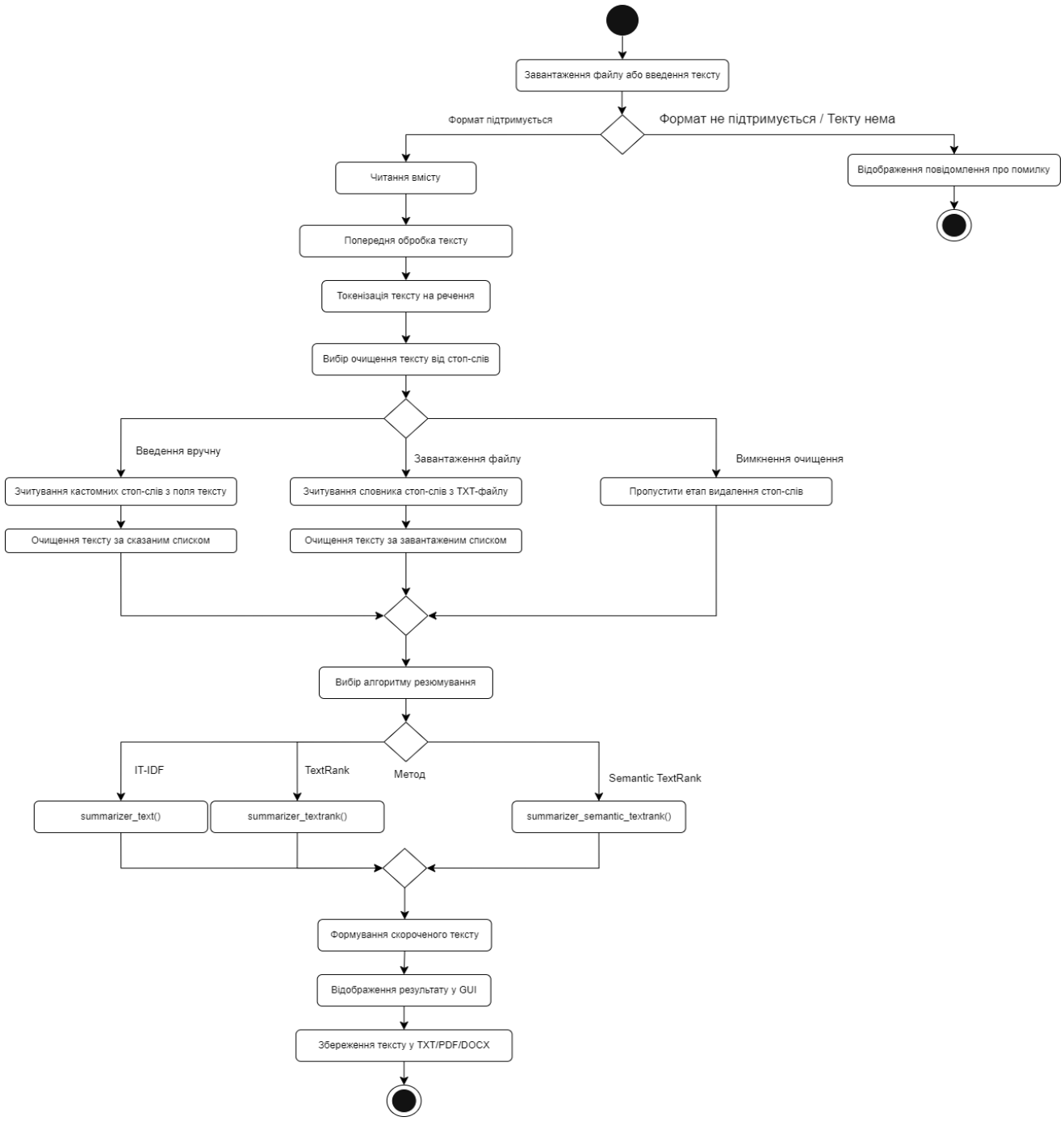


Рис. 2.3. Блок-схема алгоритму роботи застосунку автоматичного резюмування тексту

Висновки розділу

У процесі проєктування застосунку для автоматичного резюмування текстів було визначено функціональні та нефункціональні вимоги до програмного забезпечення. Функціональні вимоги охоплюють основні

можливості системи. Нефункціональні вимоги встановлюють якісні характеристики роботи програми.

Розроблена архітектура застосунку базується на модульному принципі, що забезпечує чітке розмежування функцій між компонентами системи. Описані модулі: завантаження, обробка та налаштування параметрів стоп-слів, резюмування та графічний інтерфейс, що утворює узгоджену структуру та гнучкість.

Діаграма класів і варіанти використання відображають взаємозв’язки між основними елементами системи та демонструють взаємодію між користувачем і алгоритмами резюмування. Алгоритм роботи застосунку представлений у вигляді блок-схеми, яка показує послідовність виконання основних операцій.

Таким чином, результатом проєктування підтверджується, що створений застосунок має логічну впорядковану архітектуру, відповідає визначеним вимогам і забезпечує ефективну реалізацію процесу автоматичного резюмування текстів.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ АНАЛОГІВ

Розділ присвячений практичній реалізації застосунку, а також проведенню аналізу наявних аналогів. Реалізація охоплює головні програмні модулі, використання бібліотек і приклад коду, що демонструє роботу основних компонентів системи. Головна мета продемонструвати реалізацію основних модулів, застосування алгоритмів резюмування та провести аналіз із сучасними аналогами.

3.1 Реалізація ключових модулів

Реалізація застосунку виконана мовою програмування Python, що забезпечує високу гнучкість, простоту інтеграції бібліотек та продуктивність у роботі з текстовими даними. Програмна структура побудована відповідно до модульної архітектури, описаної у розділі 2.2, де кожен компонент виконує свою роль в алгоритмі роботи всієї системи. Використані бібліотеки:

– Tkinter - створення графічного інтерфейсу користувача.

– NKTK - токенізація тексту, видалення стоп-слів.

– Scikit-learn - обчислення ваг термінів TF-IDF.

– SentenceTransformes - реалізація семантичного TextRank.

– PyPDF2, python-docx – робота з файлами PDF та DOCX.

– ReportLab - формування PDF-звіту з результатами резюмування.

Реалізація модуля завантаження та імпорту файлів, на початковому етапі роботи системи відбувається зчитування вхідного тексту. Цю функцію виконує модуль *file_loader.py,* який працює разом з методом *open_file* головного вікна програми. Такий підхід дозволяє розділити логіку обробки файлів та логіку інтерфейсу.

Модуль *file_loader* містить набір функцій для зчитування даних із різними типами даних. Кожна функція спроєктована так, щоб повертати рядки тексту.

```
from docx import Document
import PyPDF2

def load_text_file(path):
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def load_pdf_file(path):
    text = ""
    with open(path, "rb") as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text() + "\n"
    return text

def load_docx_file(path):
    doc = Document(path)
    return "\n".join([p.text for p in doc.paragraphs])
```

Взаємодія через метод *open_file*, даний метод у класі TextSummarizerGUI виступає у ролі контролера. Він відповідає за взаємодію з користувачем через діалогове вікно операційної системи та передачу файлу на обробку функції.

```
    def open_file(self):
        import os
        file_path = filedialog.askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("PDF Files", "*.pdf"), ("Word Files", "*.docx"), ("All files", "*.*")]
        )
        if not file_path:
            return
        self.loaded_filename = os.path.splitext(os.path.basename(file_path))[0]
        try:
            if file_path.endswith(".txt"):
                text = load_text_file(file_path)
            elif file_path.endswith(".pdf"):
                text = load_pdf_file(file_path)
            elif file_path.endswith(".docx"):
                from file_loader import load_docx_file
                text = load_docx_file(file_path)
            else:
                messagebox.showerror("Помилка", "Формат файлу не підтримується")
                return
            self.text_input.delete(1.0, tk.END)
            self.text_input.insert(tk.END, text)
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося прочитати файл:\n{e}")
```

Реалізація модуля резюмування, він є одним з найголовнішим компонентом системи. Тут поєднують методи класичної статистики, графових алгоритмів та сучасної нейромереживої моделі для вилучення найзмістовніших частин тексту.

Метод *make_summary* у головному файлі виконує збір параметрів з інтерфейсу: обраний алгоритм, рівень стиснення, режим автоматичного вибору та передає їх до обчислювального модуля.

```
def make_summary(self):
    text = self.text_input.get(1.0, tk.END).strip()
    if not text:
        messagebox.showwarning("Увага", "Будь ласка, введіть текст або завантажте файл")
        return

    method = self.method_var.get()
    auto = self.auto_var.get()
    target_coverage = float(self.coverage_var.get())
    max_sentences = int(self.max_sent_var.get())
    fixed_num = int(self.fixed_num_var.get())

    use_stop_words = self.use_stopwords_var.get()
    custom_words_raw = self.custom_stop_entry.get().strip()

    if not use_stop_words:
        custom_stop_words = None
    elif custom_words_raw:
        custom_stop_words = {w.strip().lower() for w in custom_words_raw.split(",") if w.strip()}
    else:
        custom_stop_words = None

    num_sentences = None if auto else fixed_num
    kwargs = {}
    if auto:
        kwargs["target_coverage"] = target_coverage
        kwargs["max_sentences"] = max_sentences

    kwargs["use_stop_words"] = use_stop_words
    kwargs["custom_stop_words"] = custom_stop_words

    try:
        if method == "Semantic TextRank":
            self.root.config(cursor="watch")
            self.root.update_idletasks()
            summary = summarize_semantic_textrank(text, num_sentences=num_sentences, **kwargs)
            self.root.config(cursor="")
        elif method == "TextRank":
            summary = summarize_textrank(text, num_sentences=num_sentences, **kwargs)
        else:
            summary = summarize_text(text, num_sentences=num_sentences, **kwargs)
```

```
        except Exception as e:
            try:
                self.root.config(cursor="")
            except:
                pass
            messagebox.showerror("Помилка", f"Помилка при резюмуванні:\n{e}")
            return

        self.summary_output.delete(1.0, tk.END)
        self.summary_output.insert(tk.END, summary)
```

Реалізація методу семантичного резюмування (summarizer.py), він є найбільш інтелектуальним компонентом, функція *summarize_semantic_textrank*. До алгоритму роботи входять наступні етапи:

1. Токенізація та адаптивне видалення стоп-слів.

2. Семантичне кодування. За допомогою багатомовної трансферної моделі paraphrase-multilingual-MiniLM-L12-v2.

3. Побудова матриці для обчислення відстані між парами слів.

4. Визначення впливових речень.

5. Адаптивний підбір, на основі заданого рівня покриття або ліміту речень сформовується тексту.

Нижче невадено лістинг програмного коду, що реалізує логіку семнтичного аналізу:

```
def summarize_text(text: str, **kwargs) -> str:
    return _execute_summary_pipeline(text, "tfidf", **kwargs)

def _execute_summary_pipeline(text: str, method_type: str, num_sentences: int = None,
                              target_coverage: float = 0.30, min_sentences: int = 1,
                              max_sentences: int = 8, max_words: int = None,
                              use_stop_words: bool = True, custom_stop_words: set = None, **kwargs) -> str:

    sentences_orig = tokenize_sentences(text)

    if len(sentences_orig) == 0:
        return "Текст порожній або не містить повних речень."

    if len(sentences_orig) <= (num_sentences or min_sentences):
        return text

    sentences_clean = preprocess_sentences(sentences_orig, custom_stop_words, use_stop_words)
    if method_type == "tfidf":
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(sentences_clean)
        scores = np.array(tfidf_matrix.sum(axis=1)).ravel()
    elif method_type == "textrank":
```

```
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(sentences_clean)
        sim_matrix = cosine_similarity(tfidf_matrix)
        scores = pagerank_from_similarity(sim_matrix, damping=0.85, iters=50)
    elif method_type == "semantic":
        model = get_sentence_model()
        embeddings = []
        for start in range(0, len(sentences_clean), 64):
            batch = sentences_clean[start:start + 64]
            embeddings.append(model.encode(batch, show_progress_bar=False))
        embeddings = np.vstack(embeddings)
        sim_matrix = cosine_similarity(embeddings)
        scores = pagerank_from_similarity(sim_matrix, damping=0.85, iters=40)
    else:
        return "Невідомий метод резюмування."

    if num_sentences is not None:
        top_indices = sorted(np.argsort(scores)[::-1][:num_sentences])
    else:
        top_indices = _select_sentences_adaptive(scores, sentences_clean, target_coverage, min_sentences,
max_sentences, max_words)

    return " ".join([sentences_orig[i] for i in top_indices])
```

Модуль експорту та збереження результатів реалізований через метод *save_summary* у класі *TextSummarizerGUI* забезпечує збереження результатів резюмування. Програма підтримує експорт у форматах TXT, DOCX та PDF.

```
    def save_summary(self):
        summary = self.summary_output.get(1.0, tk.END).strip()
        if not summary:
            messagebox.showwarning("Увага", "Спочатку згенеруйте резюме!")
            return

        file_path = filedialog.asksaveasfilename(
            defaultextension=".txt",
            filetypes=[
                ("Text Files", "*.txt"),
                ("PDF Files", "*.pdf"),
                ("Word Files", "*.docx"),
                ("All Files", "*.*")
            ],
            initialfile=f"{self.loaded_filename}_resume.txt" if self.loaded_filename else "resume.txt"
        )
        if not file_path:
            return

        try:
            if file_path.endswith(".txt"):
                with open(file_path, "w", encoding="utf-8") as f:
                    f.write(summary)
            elif file_path.endswith(".pdf"):
                from reportlab.pdfgen import canvas
                from reportlab.lib.pagesizes import A4
```

```
            from reportlab.pdfbase import pdfmetrics
            from reportlab.pdfbase.ttfonts import TTFont
            try:
                pdfmetrics.registerFont(TTFont("DejaVu","Fonts/dejavu-fonts-ttf/ttf/DejaVuSans.ttf"))
                font_name = "DejaVu"
            except Exception:
                font_name = "Helvetica"
            c = canvas.Canvas(file_path, pagesize=A4)
            c.setFont(font_name, 12)
            width, height = A4
            y = height - 40
            for line in summary.split("\n"):
                c.drawString(40, y, line)
                y -= 18
                if y < 40:
                    c.showPage()
                    c.setFont(font_name, 12)
                    y = height - 40
            c.save()
        elif file_path.endswith(".docx"):
            from docx import Document
            doc = Document()
            for line in summary.split("\n"):
                doc.add_paragraph(line)
            doc.save(file_path)
        messagebox.showinfo("Успіх", f"Файл збережено:\n{file_path}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалося зберегти файл:\n{e}")
```

Модуль графічного інтерфейсу користувача реалізований у головному класі TextSummarizerGUI. Він побудований на принципі візуального розділення зон, щоб забезпечити інтуїтивно зрозумілий процес роботи, а саме: завантаження або введення тексту, налаштування стиснення та резюмування.

Вікно та компонування, для продуктивного використання всього екрану було використано віджет ttk.Panedwindow, який дозволяє розділити робочу частину на дві частини із можливістю зміни їх ширини.

Ліва панель. Спрямована на роботу з вхідними даними. Вона містить поле ScrolledText для великих обсягів тексту та групу кнопок для швидких дій(завантаження, вставити текст, збереження)

Права панель. Містить блок *LabelFrame*, що об’єднують параметри (вибір методу, рівень стиснення) та поле виведення резюмування, також реалізовано можливість редагування тексту, щоб користувач міг внести правки перед копіюванням чи збереженням результату скорочення.

```
# Фрагмент ініціалізації панелей
paned = ttk.Panedwindow(self.root, orient=tk.HORIZONTAL)
paned.pack(fill=tk.BOTH, expand=True, padx=8, pady=6)
# Додавання фреймів з вагами для пропорційного розподілу
left_frame = ttk.Frame(paned)
paned.add(left_frame, weight=3) # 60% ширини

right_frame = ttk.Frame(paned)
paned.add(right_frame, weight=2) # 40% ширини
```

Динамічне налаштування параметрів резюмування, логіка перемикання реалізована через метод *toggle_auto*. Коли активовано режим автоматичного визначення, програма відображає повзунок рівня стиснення та максимальної кількості речень, одночасно приховуючи віджети ручного вводу.

```
def toggle_auto(self):
    #Логіка перемикання між автоматичним та ручним режимом
    if self.auto_var.get():
        # Показуємо налаштування для АВТО
        self.max_sent_label.grid(row=4, column=0, sticky="w", pady=(6, 0))
        self.max_sent_spin.grid(row=4, column=1, sticky="w", padx=6, pady=(6, 0))

        # Ховаємо налаштування для РУЧНОГО вводу
        self.fixed_num_label.grid_remove()
        self.fixed_num_spin.grid_remove()
    else:
        # Ховаємо налаштування для АВТО
        self.max_sent_label.grid_remove()
        self.max_sent_spin.grid_remove()
        # Показуємо налаштування для РУЧНОГО вводу
        self.fixed_num_label.grid(row=5, column=0, sticky="w", pady=(6, 0))
        self.fixed_num_spin.grid(row=5, column=1, sticky="w", padx=6, pady=(6, 0))
```

Реалізація механізму регулювання рівня стиснення це є важливим елементом правої панелі є повзунок, який дозволяє користувачеві керувати рівнем скорочення тексту.

```
#Повзунок рівня стиснення
ttk.Label(settings_frame, text="Рівень стиснення:", font=self.font_small).grid(
row=2,
column=0,
sticky="w")

self.coverage_var = tk.DoubleVar(value=0.30)
self.coverage_scale = ttk.Scale(settings_frame, from_=0.10, to=0.50, variable=self.coverage_var)
self.coverage_scale.grid(row=2, column=1, sticky="ew", padx=6)
#Підпис повзунка
scale_labels_frame = ttk.Frame(settings_frame)
scale_labels_frame.grid(row=3, column=1, sticky="ew", padx=6)
```

```
ttk.Label(scale_labels_frame, text="Мін.", font=self.font_small).pack(side="left")
ttk.Label(scale_labels_frame, text="Сер.", font=self.font_small).pack(side="left", expand=True)
ttk.Label(scale_labels_frame, text="Макс.", font=self.font_small).pack(side="right")
```

3.2 Аналоги та перспективні інструменти

Для оцінки розробленого застосунку проведено аналіз сучасних інструментів автоматичного резюмування текстів. Було розглянуто онлайн-сервіси, які орієнтовані на масове виконання та пропонують широкий спектр функцій. Їх перевага є висока якість скорочення тексту завдяки сучасних моделей машинного навчання та хмарних обчислень. Водночас такі системи мають суттєві обмеження: вони потребуються постійного інтернет-з’єднання, а обробка даних відбувається на стороні сервера це створює ризик витоку інформації.

ChatGPT

ChatGPT реалізує абстарктивне резюмування на основі великих мовних моделей (LLM). Різниця від екстрактивних методів, він не просто вибирає ключові речення з тексту, а формує новий скорочений текст, використовуючи архітектуру GPT. Це дозволяє системі врахувати контекст, перефразовувати та адаптувати стиль скорочення.

Технологічна основа ChatGPT лежить у архітектуру GPT, яка побудована на механізмів уваги. Модель здатна навчатись на величезних масивах інформації з використанням безліччю різних параметрів це дозволяє їй розуміти дуже складні мовні конструкції, утримати контекст у межах великих фрагментів тексту.

Переваги ChatGPT вирізняється високою якістю генерування резюме, оскільки він може не лише його скорочувати, а й переосмислювати його сенс з урахуванням контексту, до основних переваг:

1. Висока якість.
2. Здатність адаптувати стиль.
3. Підтримка багатьох мов.
4. Можливість інтеграцію з API.

Попри переваги, використання ChatGPT має свої певні обмеження, які варто враховувати під час роботи резюмування:

1. Повна залежність від інтернет-з’єднання.

2. Ризики витоку інформації через обробку даних на стороні сервера.

3. Висока потреба в обчислювальних можливостей.

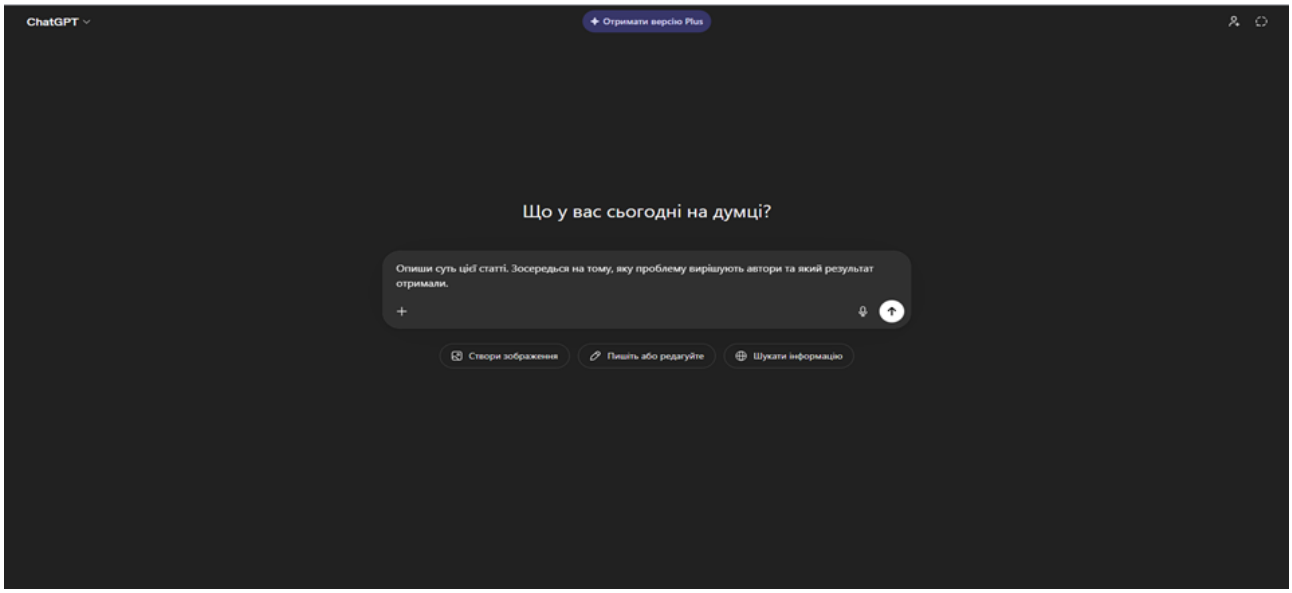


Рис.3.1 Інтерфейс сервісу ChatGPT

Smodin

Smodin реалізує екстрактивний метод резюмування, тобто виділення ключових речень із тексту без формування нового тексту. Алгоритм базується на класичних методах обробки природної мови, що дозволяє швидко вивчити суть документу.

В основі роботи інтрменту лежать математичні та лінгвістичні підходи:

– Використання статистичних моделей для визначення важливості слів та речень.

– Графові алгоритми для оцінки важливості речень у тексту.

– Хмарність

Даний сервіс має низку переваг для швидкої роботи з інормацією:

– Простота використання, сервіс доступний онлайн без необхідності становлення програмного застосунку.

– Швидке отримання результатів

– Підтримка кількох мов

Сервіс має є свої недоліки:

– Середня якість скорочення, речення виглядаються як набір ключових речень без глибокої семантичної обробки.

– Залежність від інтернет-з’єднання.

– Ризики витоку інформації через обробку даних на стороні сервера.

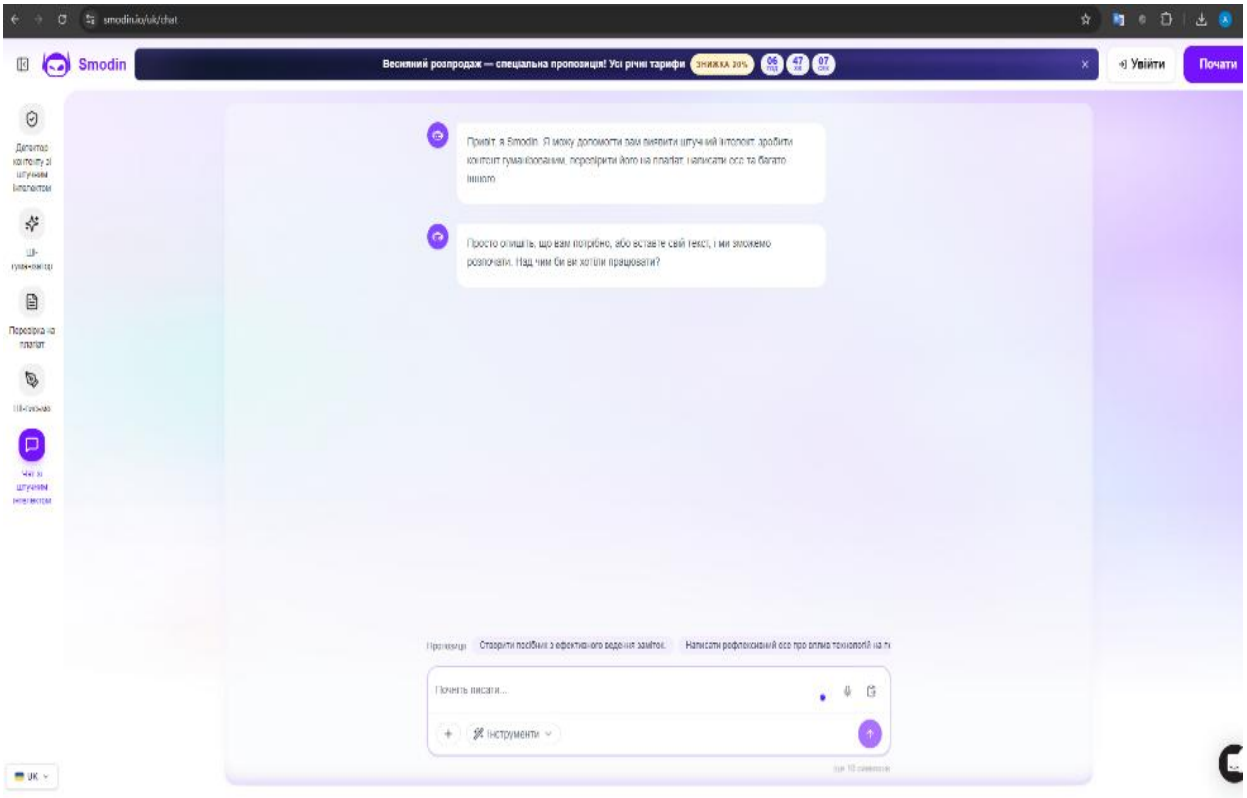


Рис.3.2 Інтерфейс сервісу Smodin

Text Summarizer

Text Summarizer реалізує екстрактивний метод резюмування на основі простих статистичних методів. Алгоритм працює за принципом підрахунки слів та NLP-процедур, виділяючи найуживаніші терміни та відповідні речення. Інструмент сканує текстовий документ, визначає найбільш вагомі слова і на основі цього виділяє оригінальні речення, які мають найбільші бали.

Робота інструмента базується на лінійних механізмах обробках тексту:

– Аналіз тексту за підрахунком слів та речень.

– Використання базових алгоритмів NLP для токенізації та видалення стоп слів.

Даний інструмент має кілька суттєвих переваг завдяки своїй архітектурній простоті:

– Простота використання, сервіс доступний онлайн без необхідності становлення програмного застосунку.

– Швидке отримання результатів.

– Мінімальні вимоги до ресурсів користувача.

Такий спрощений підхід до аналізу тексту, має свою низку технічних недоліків:

– Низька якість узагальнення, речення часто виглядаються як набір ключових речень без глибокого контекстного аналізу.

– Відсутність семантичної обробки.

– Залежність від інтернет-з’єднання.

– Ризики витоку інформації через обробку даних на стороні сервера.

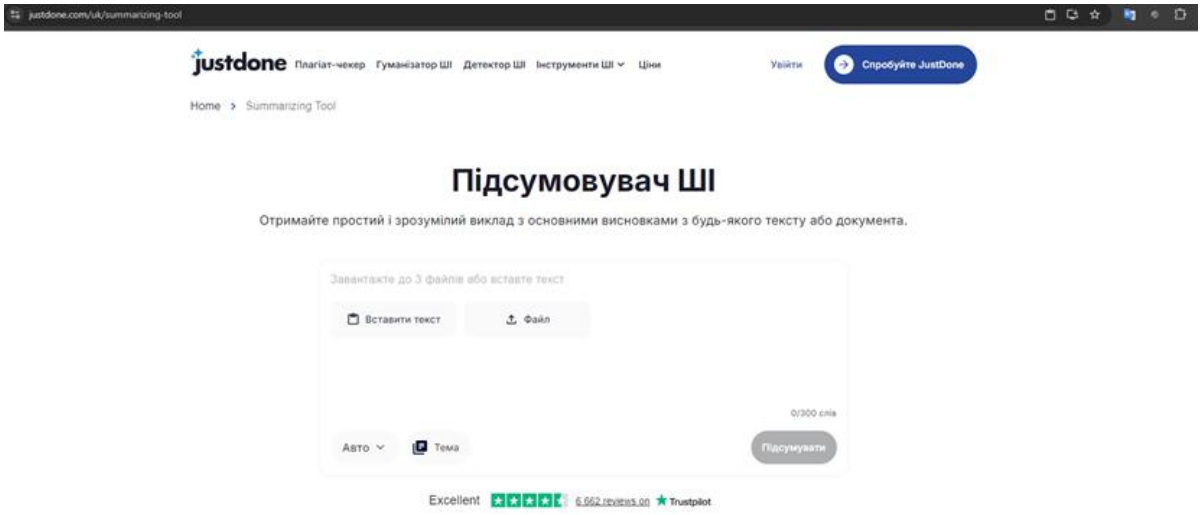


Рис.3.. Інтерфейс сервісу Text Summarizer (Justdone)

3.3 Порівняльна таблиця

Для узагальнення проведеного аналізу сучасних інструментів автоматичного резюмування текстів було складено порівняльну таблицю. Таблиця дозволяє наочно продемонструвати відмінності між глобальними онлайн-сервісами та розробленого застосунка. Онлайн-сервіси забезпечують швидкий доступ і високу якість завдяки хмарним технологіям, проте мають обмеження у сфері конфіденційності та потребують постійного інтернет-з'єднання. А локальний застосунок, навпаки, гарантує автономність роботи, безпеку даних та можливість використання кількох алгоритмів у межах одного середовища.

Таким чином, порівняльна таблиця є узагальненим інструментом оцінки, що дозволяє зробити висновки про конкурентоспроможність розробленого програмного застосунку. Вона підтверджує його переваги у сфері конфіденційності, незалежності від мережі та гнучкості вибору методів резюмування, що робить його рішенням для роботи з великими обсягами текстів.

Таблиця 3.3

Порівняння аналогів

Критерії | Сервіси	ChatGPT	Smodin	Text Summarizer	LocalSummary (власна розробка)
Налаштування етапів попередньої обробки тексту	Відсутнє (модель самостійно вирішує, як обробляти текст)	Відсутнє (налаштування стилю та фокусу впливаються на генерацію, а не на очищення тексту)	Відсутнє (закритий алгоритм без жодних налаштувань)	Передбачено керування токенізацією та вмикати/вимикати фільтрацію
Можливість використати власний словник стоп-слів	Обмежена (можливо через промт або завантаження файлу, але виконання не стабільне	Відсутнє (поля «Стиль письма» та «Фокус» задають тему, але не блокують слова	Відсутнє (інтерфейс передбачає лише вставку тексту)	Реалізовано (ручне введення слів або завантажити файл з стоп словами
Метод резюмування	LLM (глибинне навчання)	Переважно екстрактивний	Екстрактивний	Комбінований NLP (TF-IDF, TextRank, Semantic TextRank)
Якість скороченння за змістом	Висока - формує новий текст із глибоким контекстом.	Середня - вибір ключових речень, але без глибокого узагальнення.	Низька - просте скорочення за частотністю слів.	Достатня - при використанні Semantic TextRank.
Конфіденційність	Обробка на стороні сервера	Обробка на стороні сервера	Обробка на стороні сервера	Локальна обробка
Платформа	Веб / API	Веб	Веб / мобільний додаток	Desktop
Залежність від Інтернет-з'єднання	Потребує підключення	Потребує підключення	Потребує підключення	Не потребує підключення

4. ТЕСТУВАННЯ ТА ЗАТВЕРДЖЕННЯ РЕЗУЛЬТАТІВ

Тестування та затвердження є завершальним етапом розробки програмного забезпечення, що дозволяє підтвердити його працездатність, відповідність функціональним вимогам та оцінити якість отриманих результатів. Для розробленого застосунку було проведено комплексну перевірку, яка охоплює як технічні аспекти роботи модулів, так і практичну частину на реальних текстових даних.

Основна мета тестування полягає у перевірці правильної роботи основних компонентів системи: завантаження файлів різних форматів, виконання алгоритмів резюмування, відображення результатів у лівому полі та збереження скороченого тексту у вибраному форматі. З огляду на те, що програма призначена безпосередньо для споживача, значний фокус було приділено створенню зручного та мінімалістичного інтерфейсу, а також стабільності роботи додатка.

Перевірка відбувалась у кілька етапів. Спочатку проводилось модульне тестування, під час якого перевірялись функції завантаження файлів, алгоритми резюмування та методи експорту. Далі виконувалося інтеграційне тестування, що дозволяло оцінити взаємодію між модулями, від відкриття документа до отримання готового резюме. Завершальним етапом було функціональне тестування, яке проводилось через графічний інтерфейс користувача, імітуючи реальну роботу з програмою.

Затвердженням результатів здійснювалось на прикладі текстів різного обсягу та тематик. Для кожного сценарію фіксувалися очікувані та фактичні результати. Наприклад, при завантаженні файлів у додаток перевірялось коректне зчитування тексту, при виборі алгоритму, формування скороченого тексту з ключовими реченнями.

4.1 Методика тестування

Методика тестування розробленого застосунку була спрямована на перевірку коректності роботи ключових модулів та оцінку очікуваних результатів. Оскільки система має модульну архітектуру, тестування проводилось поетапно

В основу тестування були покладено такі принципи:

– Модульність, кожен космпонент перевіряється окремо.

– Інтеграційність, після перевірки кожного модуля окремо, здійснюється перевірка їх взаємодії у межах як одного процесу.

– Функціональність, тестування проводиться через графічний інтерфейс, імітуючи реальну роботу користувача.

– Практична затвердження, система застосування для реальних текстів різного обсягу та темами для оцінки якості скорочення.

Процес перевірки арограмного продукту було розділено на чотири послідовні етапи:

1. Модульне тестування

– Функцій завантаження

– Алгоритми резюмування

– Методи експорту

2. Інтеграційне тестування

– Перевірка послідовної роботи системи

3. Функціональне тестування

– Робота через графічний інтерфейс

– Оцінка зручності взаємодії та стабільність роботи програми

4. Практична затвердження

– Використання текстів різних обсягів, форматів.

– Оцінка якості скорочення за критеріями

Ефективність та готовність до експлуатації оцінювалися за такими критеріями:

– Коректність роботи модулів: відсутність критичних системних помилок під час використання всіх функцій.

– Якість резюмування: чи зберігається основний зміст документа, після його стиснення вибраним алгоритмом.

– Зручність інтерфейсу: чи зрозуміла навігація в застосунку, наявність підписів, підказок, а також швидкість досягнення результату мінімальною кількістю дій.

– Надійсність експорту: чи коректно зберігаються результати у вибраних форматах.

4.2 Приклади тестових сценаріїв

Для перевірки працездатності системи було продумана певна кількість сценаріїв, які охоплюються основні функції можливості застосунку. Кожен з сценаріїв описує умови виконання, очікуваний результат та фактичний результат, що дозволило оцінити відповідність роботи програми поставленим вимогам.

Сценарії завантаження файлів, цей блок спрямований на коректне зчитування вхійдного тексту з документів із файлів різних форматів, а також виведення їх в інтерфейсі користувача.

Завантаження TXT-файлу

– Умова: користувач обирає текстовий файл у форматі TXT.

– Очікуваний результат: вміст файлу відобразиться у лівому вікні.

– Фактичний результат: текст коректно відображається.

Завантаження PDF-файлу

– Умова: користувач обирає текстовий файл у форматі PDF.

– Очікуваний результат: вміст файлу відобразиться у лівому вікні.

– Фактичний результат: текст коректно відображається.

Завантаження DOCX-файлу

– Умова: користувач обирає текстовий файл у форматі DOCX.

– Очікуваний результат: вміст файлу відобразиться у лівому вікні.

– Фактичний результат: текст коректно відображається.

Сценарії резюмування, цей тип сценаріїв описує перевірку логіки роботи та коректну роботу реалізованих методів скорочення тексту, а аткож якість створених резюме.

Резюмування методом TF-IDF

– Умова: користувач обирає алгоритм TF-IDF.

– Очікуваний результат: формується скорчений текст із ключових речень.

– Фактичний результат: отримано резюме, що має основні ідеї тексту.

Резюмування методом TextRank

– Умова: користувач обирає алгоритм TextRank.

– Очікуваний результат: формується скорочений текст із збереженням змісту.

– Фактичний результат: отримано зв’язний текст, відображає основні ідеї тексту.

Резюмування методом Semantic TextRank

– Умова: користувач обирає алгоритм Semantic TextRank.

– Очікуваний результат: формується скорочений текст із врахуванням семантичної близькості речень.

– Фактичний результат: отримано якісне резюме з високим рівнем інформації.

Сценарії збереження результатів, спрямваний на перевірку виводу обробленого тексту та збереження його в обраний тип файлу.

Експорт у TXT

– Умова: користувач обирає збереження тексту у форматі TXT.

– Очікуваний результат: створюється текстовий файл із резюме

– Фактичний результат: файл збережено, відкривається у стандартному редакторі.

Експорт у PDF

– Умова: користувач обирає збереження тексту у форматі PDF.

– Очікуваний результат: створюється PDF-файл із резюме

– Фактичний результат: файл збережено, відкривається редакторі PDF.

Експорт у DOCX

– Умова: користувач обирає збереження тексту у форматі DOCX.

– Очікуваний результат: створюється DOCX документ із резюме

– Фактичний результат: файл збережено, відкривається редакторі DOCX.

4.3 Аналіз результатів

Проведене тестування показало працездатність усіх модулів системи та правильність їх роботи поставленим вимогам. Завантаження файлів у форматах TXT, PDF та DOCX здійснюється коректно, текст відображається у вікні програми. Алгоритми резюмування виконують скорочення тексту по заданим параметрам, а функція експорту забезпечують збереження результатів у форматах TXT, PDF та DOCX.

Перевірка роботи на практичних прикладах дозволило оцінити якість роботи кожного алгоритму.

TF-IDF виявився дієвим для коротких документів, де головні речення легко відокремити. Однак при роботі з великими текстами цей підхів може губити контекст та не завжди гарантує логіку суміжність.

TextRank продемонстрував більш збалансований результат. Алгоритм створює узагальнений текст, який зберігає головну думку документу, але з часом він іноді долучає менш важливі речення, які не несуть вагомого змісту.

Semantic TextRank показав найвищу якість скорочення. Завдяки застосуванню семантичних векторі та аналізу змістовної спорідненості речень він формує змістовний та логічне резюме, яка найбільше відповідає очікуванню користувача.

Загалом результати підтвердили, що розроблений програмний застосунок демонструє високу продуктивність як засіб автоматичного резюмування текстів. Він спроможний забезпечити якісне скорочення документів, підтримує роботу з різними типами файлів та гарантує збереження конфіденційних документів. Зазначені переваги позиціонують його конкурентоспроможність порівнюючи з хмарними сервісами, котрі не мають можливості опрацювання даних на локальному рівні та не можуть гарантувати належного рівня інформаційної безпеки.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було створено повноцінний програмний застосунок для автоматичного резюмування текстів із використанням методів обробки природної мови. Під час реалізації проєкту було виконано аналіз предметної області, обрано оптимальні технології для розробки, проєктування системи, впроваджено архітектуру системи, функціональні блоки програми, а також проведено тестування системи.

Було проаналізовано предметну галузь автоматичного резюмування текстів, що дозволило визначити ключові проблеми сучасних систем, а саме залежність від інтернет-мережі, ризики витоку інформації та обмеженість алгоритмів у збереженні основного змісту. На основі цього було сформульовано функціональні, так і нефункціональні вимоги до розроблювального застосунку.

Виконаний огляд існуючих інструментів та аналогів (як-от ChatGPT, Smodin, Text Summarizer), було визначено їх переваги та недоліки. Що дало змогу для обґрунтувало необхідність розробки локального рішення, який гарантує автономність роботи та захист даних користувача.

Розроблена архітектура застосунку на основі модульної організації з поділом на блоки: завантаження файлів, резюмування, графічний інтерфейс та експорт результатів. Такий підхід забезпечує гнучкість, здатність до майбутнього розвитку системи.

Реалізовано основну функціональність застосунку: завантаження файлів у форматах TXT, PDF та DOCX; автоматичне резюмування текстів із використанням алгоритмів TF-IDF, TextRank, Semantic TextRank, показ результатів у графічному інтерфейсі; збереження скороченого документу у форматах TXT, PDF та DOCX.

Виконані тестування підтвердили належну роботу всіх складових системи. Алгоритми створюють скорочений текст відповідно до заданих критеріїв, інтерфейс забезпечує зручну взаємодію, а функції експорту дозволяють зберегти

результати у різних форматах. Найкращі результати продемонстрував Semantic TextRank, який здатний врахувати смислову близькість речень, формуючи найбільш інформативне скорочення.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Канарський А.В., Худік Б.О. Розробка програми для автоматичного резюмування текстів із використанням методів обробки природної мови. Матеріли II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. –К.: ДУІКТ, 2025. С.142-144.

2. Канарський А.В., Худік Б.О. Організація безпеки даних у системі резюмування текстів. Матеріали Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

ПЕРЕЛІКИ ПОСИЛАНЬ

1. Cohan et al. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents /*Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, New Orleans, Louisiana. Stroudsburg, PA, USA, 2018. P. 615–619. URL: https://doi.org/10.18653/v1/n18-2097 (date of access: 18.11.2025).

2. Kuznietsov O., Kyselov G. Using and analysis of formal methods for evaluating the relevance of automatically generated summaries of informational texts. *Advanced Information Technology*. 2024. No. 1 (3). P. 32–48. URL: https://doi.org/10.17721/ait.2024.1.04 (date of access: 18.11.2025).

3. Sirohi A. Research Paper on Text to Audio Converter using NLP. *International Journal for Research in Applied Science and Engineering Technology*. 2025. Vol. 13, no. 5. P. 1313–1316. URL: https://doi.org/10.22214/ijraset.2025.70467 (date of access: 18.11.2025).

4. Jurafsky D., Martin J. *Speech and Language Processing*. – 3rd ed. – Draft, 2023. URL: https://web.stanford.edu/~jurafsky/slp3/ (date of access: 18.11.2025).

5. Zhang J., Zhao Y., Saleh M., Peter J. Liu PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization URL:https://arxiv.org/abs/1912.08777 (date of access: 12.05.2025).

6. Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., Zettlemoyer L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension URL: https://arxiv.org/abs/1910.13461 (date of access: 12.05.2025).

7. Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., Zhou Y., Li W., Liu P. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer URL:https://arxiv.org/abs/1910.10683 (date of access: 12.05.2025).

8. ДСТУ EN ISO/IEC 27002:2024. Інформаційна безпека, кібербезпека та захист конфіденційності. Заходи забезпечення інформаційної безпеки (на основі ISO/IEC 27002:2022). URL: https://learn.ztu.edu.ua/pluginfile.php/271472/mod_resource/content/2/ISO%202700 2%202022.pdf (date of access: 18.04.2025).

9. NLTK. Official Documentation. URL https://www.nltk.org (date of access: 12.05.2025).

10. ChatGPT. OpenAI. URL: https://chat.openai.com (date of access: 18.11.2025).

11. Smodin – AI Summarizer Tool. URL: https://smodin.io/summarizer (date of access: 12.05.2025).

12. Text Summarizer (Justdone.com). URL: https://justdone.com/summarizing-tool (date of access: 12.05.2025).

13. Scikit-learn. Official Documentation. URL: https://scikit-learn.org/stable/ (date of access: 12.05.2025).

14. SentenceTransformer. Official Documentation. URL: https://sbert.net/ (date of access: 12.05.2025).

15. SpaCy. Official Documentation. URL: https://spacy.io/ (date of access: 12.05.2025).

16. PyPDF2. Official Documentation. URL: https://pypdf2.readthedocs.io/en/3.x/ (date of access: 12.05.2025).

17. Python-doxc. Official Documentation. URL:https://python-docx.readthedocs.io/en/latest/ (date of access: 12.05.2025).

18. ReportLab. Official Documentation. URL: https://www.reportlab.com/ (date of access: 12.05.2025).

19. Tkinter. Official Documentation. URL: https://docs.python.org/uk/3/library/tkinter.html (date of access: 12.05.2025).

20. Cython. Official Documentation. URL: https://cython.org/ (date of access: 12.05.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ

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

Розробка програмного забезпечення для автоматичного резюмування текстів із використанням методів обробки природної мови

Виконав студент 4 курсу
групи ПД-44
Канарський Артем В'ячеславович

Керівник роботи
доктор філософії (PhD) Худік Богдан Олександрович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: автоматизація резюмування текстів з використанням методів обробки природної мови в умовах обмеженого доступу до мережі інтернет

Об'єкт дослідження: процес резюмування текстів.

Предмет дослідження: методи обробки природньої мови для резюмування текстів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область автоматичного резюмування, дослідити існуючі методи обробки тексту та визначити вимоги.
2. Спроєктувати архітектуру системи та логічну структуру модулів.
3. Реалізувати програмний застосунок із інтерфейсом та функціональною частиною системи.
4. Провести тестування системи, перевірити функціональність та виміряти продуктивність роботи алгоритму.

3

АНАЛІЗ АНАЛОГІВ

Критерії | Сервіси	ChatGPT	Smodin	Text Summarizer	LocalSummary (власна розробка)
Налаштування етапів попередньої обробки тексту	Відсутнє (модель самостійно вирішує, як обробляти текст)	Відсутнє (налаштування стилю та фокусу впливаються на генерацію, а не на очищення тексту)	Відсутнє (закритий алгоритм без жодних налаштувань)	Передбачено керування токенізацією та вмикати/вимикати фільтрацію
Можливість використати власний словник стоп-слів	Обмежена (можливо через промт або завантаження файлу, але виконання не стабільне	Відсутнє (поля «Стиль письма» та «Фокус» задають тему, але не блокують слова	Відсутнє (інтерфейс передбачає лише вставку тексту)	Реалізовано (ручне введення слів або завантажити файл з стоп словами
Метод резюмування	LLM (глибинне навчання)	Переважно екстрактивний	Екстрактивний	Комбінований NLP (TF-IDF, TextRank, Semantic TextRank)
Якість скорочення за змістом	Висока - формує новий текст із глибоким контекстом.	Середня - вибір ключових речень, але без глибокого узагальнення.	Низька - просте скорочення за частотністю слів.	Достатня - при використанні Semantic TextRank.
Конфіденційність	Обробка на стороні сервера	Обробка на стороні сервера	Обробка на стороні сервера	Локальна обробка
Платформа	Веб / API	Веб	Веб / мобільний додаток	Desktop
Залежність від Інтернет-з'єднання	Потребує підключення	Потребує підключення	Потребує підключення	Не потребує підключення

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. **Завантаження тексту**: імпорт у форматах TXT, DOCX, PDF, або вставка через буфер обміну.
2. **Перевірка коректності файлів.**
3. **Керування списком стоп-слів**: можливість введення слів вручну, завантаження з файлу або вимкнення видалення стоп-слів.
4. **Вибір параметрів резюмування**: встановлення рівня стиснення тексту, кількість речень та вибір методів.
5. **Генерація та виведення результату у вікні програми.**
6. **Експорт результатів:** у форматах TXT, DOCX, PDF.

Не функціональні вимоги

1. **Безпека:** Локальна обробка даних у межах ПК, без передачі інформації в мережу та збереження в хмару.
2. **Надійність:** обробка винятків при відкритті файлів (пошкоджені, неправильні формати) та повідомлення користувачу про помилку.
3. **Швидкість обробки:** для статистичних методів (TF-IDF, TextRank) – < 1 сек (до 10000 слів). Для семантичного аналізу (Semantic TextRank) - ~10 сек (для 5000 слів).
4. **Автономність:** робота системи без потреби в інтернет-з'єднанні.

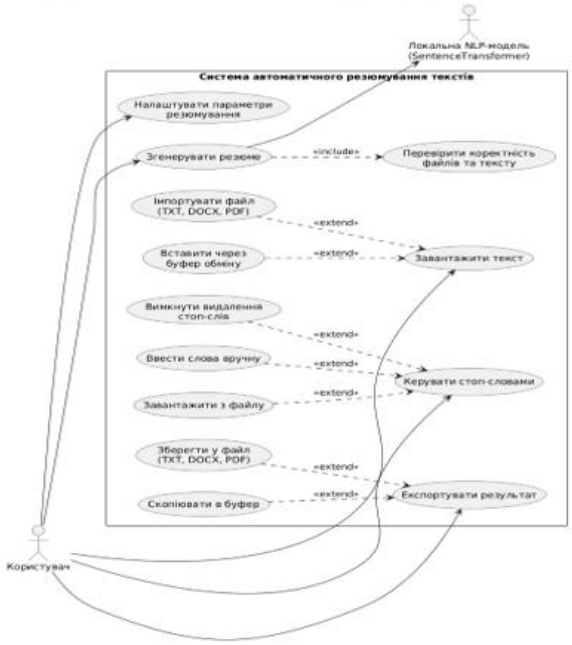
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Python

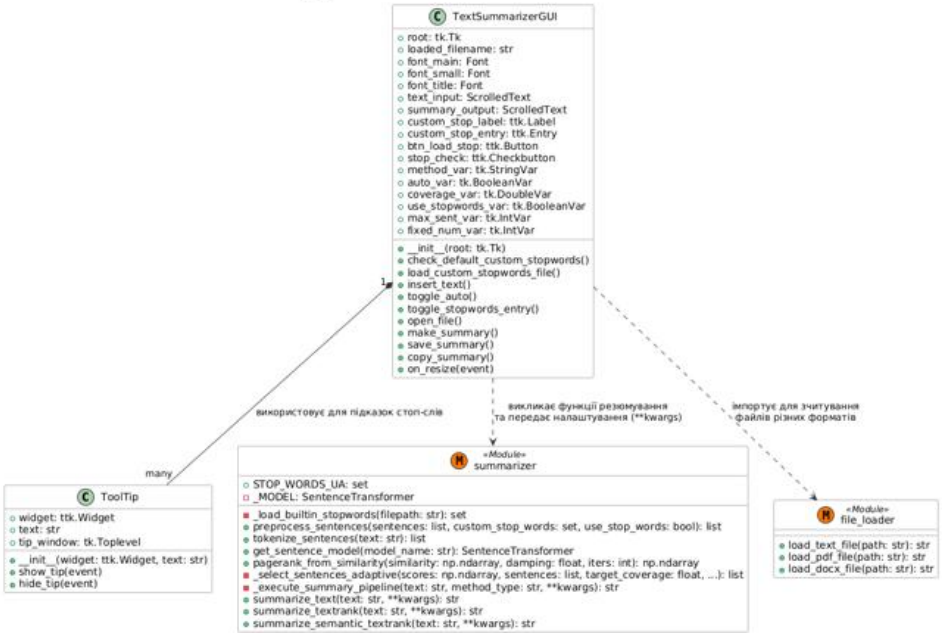
6

ДІАГРАМА ПРЕЦЕНДЕНТІВ



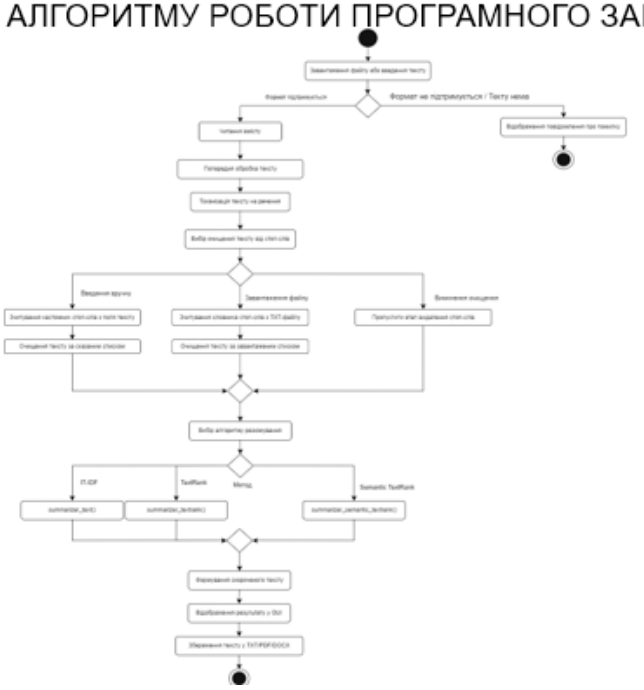
7

ДІАГРАМА КЛАСІВ



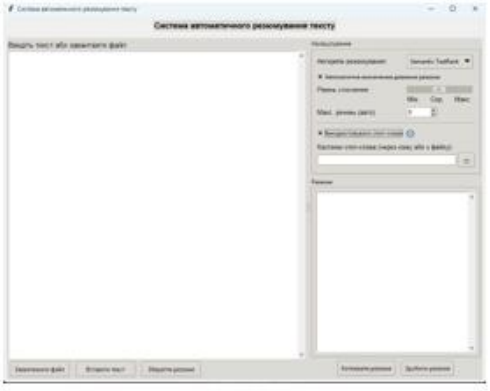
8

БЛОК-СХЕМА АЛГОРИТМУ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



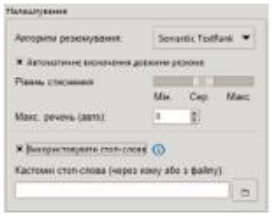
9

ЕКРАННІ ФОРМИ



Головний екран застосунку

Діалогове вікно завантаження файлу



Панель керування параметрами

Реалізація підказки

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Канарський А.В., Худік Б.О. Розробка програми для автоматичного резюмування текстів із використанням методів обробки природної мови. Матеріли II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. –К.: ДУІКТ, 2025. С.142-144.

2. Канарський А.В., Худік Б.О. Організація безпеки даних у системі резюмування текстів. Матеріали Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (*подано до друку*).

ВИСНОВКИ

1. В ході роботи було проаналізовано предметну область автоматичного резюмування текстів та сформовано вимоги до розробленої системи.

2. Спроєктовано архітектуру системи та структуру модулів, що забезпечує поділ функціональних компонентів.

3. Реалізовано програмний застосунок із графічним інтерфейсом та функціональною частино.

4. Проведено тексти запуску системи, у під час якого було перевірено працездатність компонентів та модулів інтерфейсу.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ МОДУЛІВ

```
            # Main.py
import tkinter as tk
import re
from tkinter import filedialog, messagebox, scrolledtext, ttk
import tkinter.font as tkfont

from summarizer import summarize_text,
summarize_textrank, summarize_semantic_textrank
from file_loader import load_text_file, load_pdf_file


class ToolTip:

    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tip_window = None
        self.widget.bind("<Enter>", self.show_tip)
        self.widget.bind("<Leave>", self.hide_tip)

    def show_tip(self, event=None):
        if self.tip_window or not self.text:
            return

        x = self.widget.winfo_rootx() + 25
        y = self.widget.winfo_rooty() + 20
        self.tip_window = tw = tk.Toplevel(self.widget)
        tw.wm_overrideredirect(True)  # Прибираємо
стандартні рамки вікна
        tw.wm_geometry(f"+{x}+{y}")

        label = tk.Label(tw, text=self.text, justify=tk.LEFT,
                         background="#ffffe0", relief=tk.SOLID,
borderwidth=1,
                         font=("Arial", "9", "normal"))
        label.pack(ipadx=4, ipady=2)

    def hide_tip(self, event=None):
        tw = self.tip_window
        self.tip_window = None
        if tw:
            tw.destroy()


class TextSummarizerGUI:
    def __init__(self, root):
        self.root = root
        self.loaded_filename = None
        self.root.title("Система автоматичного резюмування
тексту")
        self.root.geometry("1000x760")
        self.root.minsize(800, 600)

        style = ttk.Style()
        style.theme_use("clam")

        self.font_main = tkfont.Font(family="Arial", size=11)
        self.font_small = tkfont.Font(family="Arial", size=10)
        self.font_title = tkfont.Font(family="Arial", size=13,
weight="bold")

        header = ttk.Label(self.root, text="Система
автоматичного резюмування тексту", font=self.font_title)
        header.pack(pady=8)

        paned = ttk.Panedwindow(self.root,
orient=tk.HORIZONTAL)
        paned.pack(fill=tk.BOTH, expand=True, padx=8,
pady=6)

        left_frame = ttk.Frame(paned)
        paned.add(left_frame, weight=3)

        ttk.Label(left_frame, text="Введіть текст або
завантажте файл:", font=self.font_main).pack(anchor="w",

pady=(4, 2))
        self.text_input = scrolledtext.ScrolledText(left_frame,
wrap=tk.WORD, font=self.font_main)
        self.text_input.pack(fill=tk.BOTH, expand=True,
padx=2, pady=(0, 6))

        btn_frame = ttk.Frame(left_frame)
        btn_frame.pack(fill="x", pady=(0, 6))
        ttk.Button(btn_frame, text="Завантажити файл",
command=self.open_file, width=18).pack(side="left",
padx=6)
        ttk.Button(btn_frame, text="Вставити текст",
command=self.insert_text, width=18).pack(side="left",
padx=6)
        ttk.Button(btn_frame, text="Зберегти резюме",
command=self.save_summary, width=18).pack(side="left",
padx=6)

        right_frame = ttk.Frame(paned)
        paned.add(right_frame, weight=2)

        settings_frame = ttk.LabelFrame(right_frame,
text="Налаштування", padding=(8, 8))
        settings_frame.pack(fill="x", padx=4, pady=(4, 8))

        ttk.Label(settings_frame, text="Алгоритм
резюмування:", font=self.font_small).grid(row=0,
column=0, sticky="w")
        self.method_var = tk.StringVar(value="Semantic
TextRank")
        ttk.OptionMenu(settings_frame, self.method_var,
"Semantic TextRank", "Semantic TextRank", "TextRank",
                       "TF-IDF").grid(row=0, column=1,
sticky="ew", padx=6)

        self.auto_var = tk.BooleanVar(value=True)
        auto_check = ttk.Checkbutton(settings_frame,
text="Автоматичне визначення довжини резюме",
                                     variable=self.auto_var,
command=self.toggle_auto)
        auto_check.grid(row=1, column=0, columnspan=2,
sticky="w", pady=(6, 0))

        ttk.Label(settings_frame, text="Рівень стиснення:",
font=self.font_small).grid(row=2, column=0, sticky="w",

pady=(6, 0))
        self.coverage_var = tk.DoubleVar(value=0.30)
        self.coverage_scale = ttk.Scale(settings_frame,
from_=0.10, to=0.50, variable=self.coverage_var)
        self.coverage_scale.grid(row=2, column=1,
sticky="ew", padx=6, pady=(6, 0))

        scale_labels_frame = ttk.Frame(settings_frame)
```

```
        scale_labels_frame.grid(row=3, column=1, sticky="ew",
padx=6)
        ttk.Label(scale_labels_frame, text="Мін.",
font=self.font_small).pack(side="left")
        ttk.Label(scale_labels_frame, text="Сер.",
font=self.font_small).pack(side="left", expand=True)
        ttk.Label(scale_labels_frame, text="Макс.",
font=self.font_small).pack(side="right")

        self.max_sent_label = ttk.Label(settings_frame,
text="Макс. речень (авто):", font=self.font_small)
        self.max_sent_var = tk.IntVar(value=8)
        self.max_sent_spin = ttk.Spinbox(settings_frame,
from_=1, to=50, textvariable=self.max_sent_var, width=6)

        self.fixed_num_label = ttk.Label(settings_frame,
text="Кількість речень:", font=self.font_small)
        self.fixed_num_var = tk.IntVar(value=3)
        self.fixed_num_spin = ttk.Spinbox(settings_frame,
from_=1, to=100, textvariable=self.fixed_num_var, width=6)

        ttk.Separator(settings_frame,
orient='horizontal').grid(row=6, column=0, columnspan=2,
sticky="ew", pady=10)

        # Чекбокс активації стоп-слів
        self.use_stopwords_var = tk.BooleanVar(value=True)
        self.stop_check = ttk.Checkbutton(
            settings_frame,
            text="Використовувати стоп-слова",
            variable=self.use_stopwords_var,
            command=self.toggle_stopwords_entry
        )
        self.stop_check.grid(row=7, column=0, sticky="w",
pady=(0, 4))

        # Інфо-значок з підказкою ToolTip
        info_label = ttk.Label(settings_frame, text=" ⓘ",
font=("Arial", 11, "bold"), foreground="#0056b3",
                               cursor="hand2")
        info_label.grid(row=7, column=1, sticky="w", padx=(0,
10))
        ToolTip(info_label, "Стандартний список стоп-слів є
узагальненим.\n"
                            "Ви можете вимкнути його, ввести слова
через кому вручну,\n"
                            "або завантажити файл зі своїм списком
термінів.")

        self.custom_stop_label = ttk.Label(settings_frame,
text="Власний список (вбудований буде вимкнено):",
font=self.font_small)
        self.custom_stop_label.grid(row=8, column=0,
columnspan=2, sticky="w", pady=(4, 2))

        stop_input_frame = ttk.Frame(settings_frame)
        stop_input_frame.grid(row=9, column=0,
columnspan=2, sticky="ew", padx=2, pady=(0, 4))

        self.custom_stop_entry = ttk.Entry(stop_input_frame,
font=self.font_small)
        self.custom_stop_entry.pack(side="left", fill="x",
expand=True, padx=(0, 4))

        self.btn_load_stop = ttk.Button(stop_input_frame,
text="📂", width=3,
command=self.load_custom_stopwords_file)
        self.btn_load_stop.pack(side="right")

        settings_frame.columnconfigure(1, weight=1)

        summary_frame = ttk.LabelFrame(right_frame,
text="Резюме", padding=(8, 8))
        summary_frame.pack(fill=tk.BOTH, expand=True,
padx=4, pady=(0, 8))

        self.summary_output =
scrolledtext.ScrolledText(summary_frame, wrap=tk.WORD,
font=self.font_main, height=12)
        self.summary_output.pack(fill=tk.BOTH, expand=True)

        summary_btn_frame = ttk.Frame(right_frame)
        summary_btn_frame.pack(pady=(0, 8))
        ttk.Button(summary_btn_frame, text="Копіювати
резюме", command=self.copy_summary).pack(side="left",
padx=6)
        ttk.Button(summary_btn_frame, text="Зробити
резюме", command=self.make_summary).pack(side="left",
padx=6)

        self.root.bind("<Configure>", self.on_resize)
        self.root.after(100, lambda: paned.sashpos(0,
int(self.root.winfo_width() * 0.62)))

        self.toggle_auto()
        self.toggle_stopwords_entry()

        self.check_default_custom_stopwords()

    def check_default_custom_stopwords(self):
        import os
        default_file = "custom_stopwords.txt"
        if os.path.exists(default_file):
            try:
                with open(default_file, "r", encoding="utf-8") as f:
                    words = [line.strip() for line in f.readlines() if
line.strip()]

                    words_str = ", ".join(words)
                    self.custom_stop_entry.delete(0, tk.END)
                    self.custom_stop_entry.insert(0, words_str)
            except Exception as e:
                print(f"Не вдалося автоматично зчитати
{default_file}: {e}")

    def load_custom_stopwords_file(self):
        file_path = filedialog.askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("All files", "*.*")]
        )
        if not file_path:
            return
        try:
            with open(file_path, "r", encoding="utf-8") as f:
                content = f.read()

                words_in_quotes = re.findall(r"['\"]([^'\"]+)['\"]",
content)

                if words_in_quotes:
                    words = [w.strip() for w in words_in_quotes if
w.strip()]
                elif "," in content:
                    words = [w.strip() for w in content.split(",") if
w.strip()]
                else:
                    words = [line.strip() for line in
content.splitlines() if line.strip()]
```

```
            words_str = ", ".join(words)
            self.custom_stop_entry.config(state="normal")
            self.custom_stop_entry.delete(0, tk.END)
            self.custom_stop_entry.insert(0, words_str)

            self.toggle_stopwords_entry()

            messagebox.showinfo("Успіх",
                                f"Успішно завантажено власний
список!\nКількість слів: {len(words)}\nВбудований
список тепер не використовуватиметься.")
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
прочитати файл стоп-слів:\n{e}")

    def insert_text(self):
        try:
            text = self.root.clipboard_get()
            self.text_input.delete(1.0, tk.END)
            self.text_input.insert(tk.END, text)
        except tk.TclError:
            messagebox.showwarning("Увага", "Буфер обміну
порожній або недоступний")

    def toggle_auto(self):
        if self.auto_var.get():
            self.max_sent_label.grid(row=4, column=0,
sticky="w", pady=(6, 0))
            self.max_sent_spin.grid(row=4, column=1,
sticky="w", padx=6, pady=(6, 0))
            self.fixed_num_label.grid_remove()
            self.fixed_num_spin.grid_remove()
        else:
            self.max_sent_label.grid_remove()
            self.max_sent_spin.grid_remove()
            self.fixed_num_label.grid(row=5, column=0,
sticky="w", pady=(6, 0))
            self.fixed_num_spin.grid(row=5, column=1,
sticky="w", padx=6, pady=(6, 0))

    def toggle_stopwords_entry(self):
        if self.use_stopwords_var.get():
            self.custom_stop_entry.config(state="normal")
            self.btn_load_stop.config(state="normal")
            self.custom_stop_label.config(foreground="")
        else:
            self.custom_stop_entry.config(state="disabled")
            self.btn_load_stop.config(state="disabled")
            self.custom_stop_label.config(foreground="gray")

    def open_file(self):
        import os
        file_path = filedialog.askopenfilename(
            filetypes=[("Text Files", "*.txt"), ("PDF Files",
"*.pdf"), ("Word Files", "*.docx"), ("All files", "*.*")]
        )
        if not file_path:
            return
        self.loaded_filename =
os.path.splitext(os.path.basename(file_path))[0]
        try:
            if file_path.endswith(".txt"):
                text = load_text_file(file_path)
            elif file_path.endswith(".pdf"):
                text = load_pdf_file(file_path)
            elif file_path.endswith(".docx"):
                from file_loader import load_docx_file
                text = load_docx_file(file_path)
            else:
                messagebox.showerror("Помилка", "Формат
файлу не підтримується")
                return
            self.text_input.delete(1.0, tk.END)
            self.text_input.insert(tk.END, text)
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
прочитати файл:\n{e}")

    def make_summary(self):
        text = self.text_input.get(1.0, tk.END).strip()
        if not text:
            messagebox.showwarning("Увага", "Будь ласка,
введіть текст або завантажте файл")
            return

        method = self.method_var.get()
        auto = self.auto_var.get()
        target_coverage = float(self.coverage_var.get())
        max_sentences = int(self.max_sent_var.get())
        fixed_num = int(self.fixed_num_var.get())

        use_stop_words = self.use_stopwords_var.get()
        custom_words_raw =
self.custom_stop_entry.get().strip()

        (custom_stop_words=None)
        if not use_stop_words:
            custom_stop_words = None
        elif custom_words_raw:
            custom_stop_words = {w.strip().lower() for w in
custom_words_raw.split(",") if w.strip()}
        else:
            custom_stop_words = None

        num_sentences = None if auto else fixed_num
        kwargs = {}
        if auto:
            kwargs["target_coverage"] = target_coverage
            kwargs["max_sentences"] = max_sentences

            kwargs["use_stop_words"] = use_stop_words
        kwargs["custom_stop_words"] = custom_stop_words

        try:
            if method == "Semantic TextRank":
                self.root.config(cursor="watch")
                self.root.update_idletasks()
                summary = summarize_semantic_textrank(text,
num_sentences=num_sentences, **kwargs)
                self.root.config(cursor="")
            elif method == "TextRank":
                summary = summarize_textrank(text,
num_sentences=num_sentences, **kwargs)
            else:
                summary = summarize_text(text,
num_sentences=num_sentences, **kwargs)
        except Exception as e:
            try:
                self.root.config(cursor="")
            except:
                pass
            messagebox.showerror("Помилка", f"Помилка при
резюмуванні:\n{e}")
            return

        self.summary_output.delete(1.0, tk.END)
        self.summary_output.insert(tk.END, summary)
```

```
    def save_summary(self):
        summary = self.summary_output.get(1.0,
tk.END).strip()
        if not summary:
            messagebox.showwarning("Увага", "Спочатку
згенеруйте резюме!")
            return

        file_path = filedialog.asksaveasfilename(
            defaultextension=".txt",
            filetypes=[
                ("Text Files", "*.txt"),
                ("PDF Files", "*.pdf"),
                ("Word Files", "*.docx"),
                ("All Files", "*.*")
            ],
            initialfile=f"{self.loaded_filename}_resume.txt" if
self.loaded_filename else "resume.txt"
        )
        if not file_path:
            return

        try:
            if file_path.endswith(".txt"):
                with open(file_path, "w", encoding="utf-8") as f:
                    f.write(summary)
            elif file_path.endswith(".pdf"):
                from reportlab.pdfgen import canvas
                from reportlab.lib.pagesizes import A4
                from reportlab.pdfbase import pdfmetrics
                from reportlab.pdfbase.ttfonts import TTFont
                try:
                    pdfmetrics.registerFont(TTFont("DejaVu",
"Fonts/dejavu-fonts-ttf/ttf/DejaVuSans.ttf"))
                    font_name = "DejaVu"
                except Exception:
                    font_name = "Helvetica"
                c = canvas.Canvas(file_path, pagesize=A4)
                c.setFont(font_name, 12)
                width, height = A4
                y = height - 40
                for line in summary.split("\n"):
                    c.drawString(40, y, line)
                    y -= 18
                    if y < 40:
                        c.showPage()
                        c.setFont(font_name, 12)
                        y = height - 40
                c.save()
            elif file_path.endswith(".docx"):
                from docx import Document
                doc = Document()
                for line in summary.split("\n"):
                    doc.add_paragraph(line)
                doc.save(file_path)
            messagebox.showinfo("Успіх", f"Файл
збережено:\n{file_path}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
зберегти файл:\n{e}")

    def copy_summary(self):
        summary = self.summary_output.get(1.0,
tk.END).strip()
        if not summary:
            messagebox.showwarning("Увага", "Немає тексту
для копіювання!")
            return
        self.root.clipboard_clear()
        self.root.clipboard_append(summary)
        messagebox.showinfo("Готово", "Резюме скопійовано
в буфер обміну!")

    def on_resize(self, event):
        try:
            w = max(800, self.root.winfo_width())
            base = max(10, int(w / 90))
            self.font_main.configure(size=base)
            self.font_small.configure(size=max(9, base - 1))
            self.font_title.configure(size=max(12, base + 2))
        except Exception:
            pass


if __name__ == "__main__":
    root = tk.Tk()
    app = TextSummarizerGUI(root)
    root.mainloop()

#file-loader.py
from docx import Document

def load_text_file(path):
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def load_pdf_file(path):
    import PyPDF2
    text = ""
    with open(path, "rb") as file:
        reader = PyPDF2.PdfReader(file)
        for page in reader.pages:
            text += page.extract_text() + "\n"
    return text

def load_docx_file(path):
    doc = Document(path)
    text = "\n".join([p.text for p in doc.paragraphs])
    return text

import numpy as np
import nltk
import ast
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

_MODEL = None

def _load_builtin_stopwords(filepath: str =
"stopwords_ua_list.txt") -> set:
    try:
        with open(filepath, "r", encoding="utf-8") as f:
            content = f.read()

            words = re.findall(r"['\"]([^'\"]+)['\"]", content)
            return set(words)
    except FileNotFoundError:
        print(f"[Попередження] Файл '{filepath}' не знайдено.
Вбудований список буде порожнім.")
        return set()


STOP_WORDS_UA = _load_builtin_stopwords()


def preprocess_sentences(sentences,
custom_stop_words=None, use_stop_words=True):
```

```
    filtered = []

    if not use_stop_words:
        full_stop_words = set()
    elif custom_stop_words is not None and
len(custom_stop_words) > 0:
        full_stop_words = set(custom_stop_words)
else:
        full_stop_words = set(STOP_WORDS_UA)
    for sent in sentences:
        words = re.findall(r'\b\w+\b', sent.lower())
        words_filtered = [w for w in words if w not in
full_stop_words]
        filtered.append(" ".join(words_filtered))
    return filtered

def tokenize_sentences(text: str) -> list:
        if not text or not isinstance(text, str):
        return []

    text = text.replace('\x00', '')

    try:
        from nltk.tokenize import PunktSentenceTokenizer
        tokenizer = PunktSentenceTokenizer()
        raw_sentences = tokenizer.tokenize(text)
    except Exception:
        raw_sentences = re.split(r'(?<=[.!?])\s+', text)

    valid = [s.strip() for s in raw_sentences if
len(re.findall(r'\b\w+\b', s)) >= 2]

    if len(valid) == 0:
        valid = [s.strip() for s in text.split('\n') if len(s.strip()) >
5]

    return valid


def get_sentence_model(model_name: str = "paraphrase-
multilingual-MiniLM-L12-v2"):
    global _MODEL
    if _MODEL is None:
        from sentence_transformers import
SentenceTransformer
        _MODEL = SentenceTransformer(model_name)
    return _MODEL


def pagerank_from_similarity(similarity: np.ndarray,
damping: float = 0.85, iters: int = 50) -> np.ndarray:
    sim = np.array(similarity, dtype=float)
    row_sums = sim.sum(axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1.0
    P = sim / row_sums
    n = P.shape[0]
    scores = np.ones(n) / n
    for _ in range(iters):
        scores = (1 - damping) / n + damping * P.dot(scores)
    return scores


def _select_sentences_adaptive(scores: np.ndarray,
sentences: list, target_coverage: float = 0.30,
                               min_sentences: int = 1, max_sentences:
int = 8, max_words: int = None) -> list:
    n = len(sentences)
    if n == 0: return []
    total_weight = float(scores.sum())
    if total_weight == 0: return list(range(min(min_sentences,
n)))

    ranked_idx = np.argsort(scores)[::-1]
    selected = []
    cum_weight = 0.0

    for idx in ranked_idx:
        selected.append(idx)
        cum_weight += scores[idx]
        if max_words is not None:
            words_count = sum(len(sentences[i].split()) for i in
selected)
            if words_count > max_words:
                selected.pop()
                break

        coverage = cum_weight / total_weight
        if (coverage >= target_coverage and len(selected) >=
min_sentences) or len(selected) >= max_sentences:
            break

    if len(selected) < min_sentences:
        needed = min_sentences - len(selected)
        remaining = [i for i in ranked_idx if i not in selected]
        selected.extend(remaining[:needed])

    return sorted(set(selected))


def _execute_summary_pipeline(text: str, method_type: str,
num_sentences: int = None,
                              target_coverage: float = 0.30,
min_sentences: int = 1,
                              max_sentences: int = 8, max_words: int =
None,
                              use_stop_words: bool = True,
custom_stop_words: set = None, **kwargs) -> str:

    sentences_orig = tokenize_sentences(text)

    if len(sentences_orig) == 0:
        return "Текст порожній або не містить повних
речень."

    if len(sentences_orig) <= (num_sentences or
min_sentences):
        return text

    sentences_clean = preprocess_sentences(sentences_orig,
custom_stop_words, use_stop_words)

    if method_type == "tfidf":
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(sentences_clean)
        scores = np.array(tfidf_matrix.sum(axis=1)).ravel()
    elif method_type == "textrank":
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(sentences_clean)
        sim_matrix = cosine_similarity(tfidf_matrix)
        scores = pagerank_from_similarity(sim_matrix,
damping=0.85, iters=50)
    elif method_type == "semantic":
        model = get_sentence_model()
        embeddings = []
        for start in range(0, len(sentences_clean), 64):
            batch = sentences_clean[start:start + 64]
            embeddings.append(model.encode(batch,
show_progress_bar=False))
```

```
        embeddings = np.vstack(embeddings)
        sim_matrix = cosine_similarity(embeddings)
        scores = pagerank_from_similarity(sim_matrix,
damping=0.85, iters=40)
    else:
        return "Невідомий метод резюмування."

    if num_sentences is not None:
        top_indices = sorted(np.argsort(scores)[::-
1][:num_sentences])
    else:
        top_indices = _select_sentences_adaptive(scores,
sentences_clean, target_coverage, min_sentences,
max_sentences,
                                         max_words)

    return " ".join([sentences_orig[i] for i in top_indices])

def summarize_text(text: str, **kwargs) -> str:
    return _execute_summary_pipeline(text, "tfidf", **kwargs)

def summarize_textrank(text: str, **kwargs) -> str:
    return _execute_summary_pipeline(text, "textrank",
**kwargs)

def summarize_semantic_textrank(text: str, **kwargs) -> str:
    return _execute_summary_pipeline(text, "semantic",
**kwargs)
```