МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)
Кафедра «Системний аналіз та обчислювальна математика»
(повне найменування кафедри)

Пояснювальна записка

до дипломної роботи
бакалавра
(ступінь вищої освіти)

на тему СИСТЕМНИЙ АНАЛІЗ ТА ІНТЕЛЕКТУАЛЬНА КЛАСТЕРИЗАЦІЯ НЕСТРУКТУРОВАНИХ МАСИВІВ ДАНИХ У ФАЙЛОВИХ СХОВИЩАХ
(назва теми)

Виконав: студент 4
курсу, групи КНТ-812
Спеціальності 124 Системний аналіз
(код і найменування спеціальності)
Освітня програма (спеціалізація)
«Інтелектуальні технології та прийняття рішень в складних системах»
РУДЕНОК В.В.
(ПРІЗВИЩЕ та ініціали)
Керівник ТЕРЕЩЕНКО Е.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент ЛОЗОВСЬКА Л.І.
(ПРІЗВИЩЕ та ініціали)

2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут, факультет Факультет комп'ютерних наук і технологій
Кафедра «Системний аналіз та обчислювальна математика»
Ступінь вищої освіти бакалавра
Спеціальність 124 Системний аналіз
(код і найменування)
Освітня програма (спеціалізація) «Інтелектуальні технології та прийняття рішень в складних системах»
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Еліна ТЕРЕЩЕНКО
«20» квітня 2026 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТА
РУДЕНКУ Вадиму Васильовичу
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема роботи Системний аналіз та інтелектуальна кластерізація неструктурованих масивів даних у файлових сховищах
керівник роботи к.ф.-м.н., доцент ТЕРЕЩЕНКО Еліна Валентинівна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від «23 »квітня 2026 року №196
2. Строк подання студентом роботи 01 червня 2026
3. Вихідні дані до роботи репрезинтативний фрагмент бази даних, науково-практичного характеру наданий лабораторією генетичних досліджень Інституту олійних культур НААН України
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Обґрунтування актуальності проблеми; аналіз методів кластеризації, нормалізації та візуалізації даних; розробка програмного застосунку; реалізація алгоритмів; тестування розробленого застосунку на реальних даних та формулювання висновків.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
Розділ 1-3	ТЕРЕЩЕНКО Е.В., зав. кафедри	20.04.2026	04.06.2026
Нормоконтроль	ШИРОКОРАД Д.В., доцент	05.06.2026	08.06.2026

7. Дата видачі завдання «02» березня 2026 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	1 тиждень	Розділ 1
3	Вибір методів класифікації та візуалізації	1 тиждень	Розділ 1
4	Робота з Excel файлами та базою даних	2 тиждень	Розділ 2
5	Реалізація алгоритмів та методів	3-6 тиждень	Розділ 2-3
6	Реалізація візуалізації даних	6-8 тиждень	Розділ 3
7	Аналіз на основі візуалізації	9 тиждень	Розділ 3
8	Оформлення пояснювальної записки та відповідної документації	10 тиждень	Додатки
9	Нормоконтроль та рецензування	11 тиждень	
10	Захист дипломної роботи	12 тиждень	

Студент _____ РУДЕНОК В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ ТЕРЕЩЕНКО Е.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота: 141 с., 10 рис., 5 схем , 13 джерел, 1 додаток.

Об’єкт дослідження: процеси систематизації та впорядкування великих масивів неструктурованих та гетерогенних даних у файлових сховищах в умовах неповноти інформації та нечіткої логіки їх організації.

Предмет дослідження: методи інтелектуальної кластеризації неструктурованих масивів даних у файлових сховищах .

Мета дослідження: розробка та реалізація системи інтелектуальної кластеризації неструктурованих масивів даних у файлових сховищах .

Для досягнення мети поставлено такі задачі: проаналізувати існуючі підходи до управління неструктурованими даними у сховищах; обґрунтувати вибір алгоритмів інтелектуальної кластеризації; розробити математичну модель екстракції контекстних ознак для «зворотного структурування» файлів; спроєктувати архітектуру системи та програмно реалізувати десктопний додаток на Python; експериментально дослідити ефективність створеного комплексу на реальному масиві даних.

Результатом є розробка інваріантного алгоритму дворівневої морфофункціональної фільтрації, що поєднує каскадну кластеризацію BIRCH і DBSCAN для динамічного виявлення логічної архітектури хаотичних сховищ та ізоляції інформаційного шуму. Практичним результатом роботи є створення автономного десктопного додатка на мові Python, який на базі наскрізного індексу Manifest.csv забезпечує лінійну швидкість обробки даних (15–17 МБ/хв) та семантичний пошук у гетерогенних архівах з точністю понад 95%.

БАГАТОПАРАМЕТРИЧНІ ДАНІ, КЛАСТЕРИЗАЦІЯ, АНАЛІЗ ДАНИХ, ПОШУК, PYTHON

ЗМІСТ

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується не лише кількісним накопиченням даних, а й складними процесами їх внутрішньої трансформації. Наукова проблема даного дослідження зосереджена на обробці гетерогенних масивів, які пройшли цикли багаторазового синтезу та часткового структурування відповідно до конкретних операційних потреб протягом тривалого періоду (10–15 років).

Ключовою особливістю об’єкта дослідження є те, що дані не є «сирими» у класичному розумінні; вони являють собою результат ітераційного синтезу.

Кожен етап життєвого циклу інформації супроводжувався зміною її структури для вирішення вузькоспеціалізованих завдань, що призвело до виникнення «багатошарової» гетерогенності. У таких масивах фрагменти інформації можуть бути частково впорядковані за локальними стандартами, які на сьогодні є застарілими або несумісними між собою.

Обґрунтування доцільності проекту випливає з необхідності вирішення наступних технічних протиріч:

Проблема вторинного синтезу: дані, що були багаторазово переформатовані під потреби попередніх періодів, містять «семантичні нашарування», які стандартні реляційні алгоритми інтерпретують як шумабо помилки структури.

Часткова структурованість: наявність локальних острівців структури всередині хаотичного масиву потребує гнучких нейроморфних підходів, здатних розпізнавати внутрішню логіку без повної переіндексації всьогообсягу.

Математична детермінація: в умовах, коли дані синтезувалися за різними принципами, єдиним способом забезпечення надійності пошуку (на рівні 95%) є використання багатовимірної вузлової топології, яка фокусується на функціональних зв’язках, а не на статичних форматах.

Доцільність дослідження також зумовлена необхідністю автономного адміністрування таких складних систем. Оскільки ручна обробка багаторазово синтезованих даних є економічно неефективною та схильною до помилок суб’єктивного фактора, розробка алгоритмів, що базуються на аксоноподібних зв’язках та предикативних словниках, є єдиним механізмом підтримки актуальності бази знань.

Метою роботи є проєктування та реалізація універсальної моделі пошуку, яка забезпечує високу релевантність видачі (P=0.95) шляхом зворотного розкодування функціональних зв’язків у багаторазово синтезованих та частково структурованих масивах даних.

РОЗДІЛ 1 СИСТЕМНИЙ АНАЛІЗ ПРОБЛЕМИ ОРГАНІЗАЦІЇ НЕСТРУКТУРОВАНИХ ДАНИХ

1.1 Аналіз сучасного стану інформаційних технологій та парадокс ефективності СУБД

Сучасний етап розвитку інформаційних технологій (IT) характеризується експоненціальним зростанням обсягів даних, що генеруються як людською діяльністю, так і автоматизованими системами (IoT, сенсори, лог-файли)[1]. В умовах цифрової трансформації дані перетворилися з допоміжного ресурсу на основний актив економіки. Проте стрімке вдосконалення апаратного забезпечення та поява хмарних обчислень не лише вирішили старі проблеми масштабування, а й висвітлили фундаментальний внутрішній конфлікт у методах обробки інформації, відомий як «парадокс ефективності СУБД».

На сьогоднішній день домінуючими трендами є перехід до мікросервісної архітектури, використання штучного інтелекту для аналітики в реальному часі та впровадження концепції Big Data. Традиційні реляційні СУБД (RDBMS), які десятиліттями базувалися на принципах ACID (атомарність, узгодженість, ізоляція, довговічність), стикаються з викликами, які вимагають не лише швидкості запису, а й складної структурної обробки неструктурованих даних. Ринок розділився на класичні системи (SQL) та гнучкі рішення (NoSQL, NewSQL), що дозволяють працювати з графами, документами та ключами-значеннями. Однак, незважаючи на технологічне різноманіття, основна проблема залишається незмінною: чим складнішою стає система для забезпечення ефективності пошуку, тим більше ресурсів вона витрачає на обслуговування власної структури.

Парадокс ефективності СУБД полягає в тому, що зусилля, спрямовані на прискорення доступу до даних, часто призводять до пропорційного (або навіть більшого) уповільнення загальної продуктивності системи та збільшення витрат

ресурсів. Цей парадокс проявляється у кількох аспектах. Надмірність індексації: для того, щоб миттєво знайти один рядок серед мільярдів, створюються складні індекси (B-дерева, хеш-таблиці). Проте кожен новий індекс уповільнює операції запису та оновлення даних, оскільки система змушена перераховувати метадані при кожній зміні. В результаті, СУБД витрачає до 70-80% обчислювальної потужності не на роботу з самими даними, а на підтримку допоміжних структур. Складність нормалізації проти швидкості читання: класична теорія баз даних вимагає високого рівня нормалізації для уникнення аномалій. Проте для отримання цілісного звіту системі доводиться виконувати ресурсомісткі операції об’єднання (JOIN). Спроба денормалізації (для швидкості) призводить до роздування обсягів зберігання та ризику втрати цілісності. Ефект «прихованої роботи»: у сучасних високонавантажених системах значна частка часу CPU витрачається на механізми блокування, журналювання транзакцій та збирання «сміття» в пам'яті. Парадоксально, але додавання нових обчислювальних вузлів у кластер може знизити ефективність кожної окремої одиниці через зростання витрат на синхронізацію.

1.2. Специфіка гетерогенних масивів даних у файлових сховищах підприємств

У сегменті малого та середнього бізнесу підхід до зберігання інформації суттєво відрізняється від корпоративних стандартів великих підприємств[2]. Якщо великі корпорації інвестують у дорогі системи класу ERP або централізовані сховища даних (Data Warehouse), то малий бізнес зазвичай експлуатує локальні файлові сховища , що базуються на NAS-серверах, хмарних дисках або розподілених мережевих папках. Головною особливістю таких середовищ є

накопичення гетерогенних масивів даних — інформаційних об'єктів різної природи, форматів та ступеня структурованості.

Гетерогенність у контексті локальних файлових систем малого бізнесу проявляється у трьох основних аспектах.

Форматна неоднорідність. Сховища заповнені сумішшю бінарних об'єктів, текстових документів, електронних таблиць та специфічних технічних файлів, що містять результати лінійних обчислень або масиви синтезованих даних. Кожен тип файлу потребує окремого алгоритму інтерпретації, що ускладнює створення єдиної аналітичної моделі.

Структурна розрізненість. Навіть у межах одного типу даних спостерігається різна внутрішня топологія. Один набір даних може бути представлений у вигляді чіткої матриці, тоді як інший — як нелінійний лог-файл із багаторівневою вкладеністю та складними логічними зв'язками між елементами.

Семантична розірваність. Дані часто позбавлені єдиної системи метаданих. Інформація про ключові показники або результати аналізу може бути розпорошена між різними файловими об'єктами без явних ідентифікаторів зв'язку, що створює проблему "інформаційних островів".

Концептуальна дивергенція семантично схожих об’єктів у гетерогенних середовищах. У процесі системного аналізу файлових сховищ виникає критична проблема, пов’язана з феноменом структурної мімікрії даних. Суть явища полягає в тому, що цифрові об’єкти, які мають ідентичну або схожу номенклатуру (найменування) та уніфіковану синтаксичну структуру, можуть бути еквівалентними лише на морфологічному рівні, але докорінно відрізнятися за своєю внутрішньою логікою та інформаційним наповненням.

Специфіка малого бізнесу полягає в тому, що обробка таких масивів відбувається в умовах дефіциту обчислювальних потужностей . На відміну від серверних кластерів, аналіз тут часто виконується на локальних робочих станціях. Головним викликом стає обробка великих масивів числових даних, що зберігаються

у текстовому вигляді. Читання мільйонів записів створює критичне навантаження на підсистему введення-виведення та оперативну пам'ять.

У таких умовах гетерогенність стає перешкодою для швидкого позиціонування цільових результатів, оскільки система змушена проводити повне сканування всього обсягу даних замість використання оптимізованих індексів.

Для ефективної роботи з гетерогенними масивами в умовах обмежених ресурсів необхідно впроваджувати методи цифрового сегментування та кодування станів. Це дозволяє виокремлювати значущі послідовності та закономірності із загального потоку "інформаційного шуму", застосовувати методи статистичного контролю для оцінки якості даних без необхідності їх повного завантаження в реляційну базу даних, використовувати проміжні багаторівневі структури для перетворення сирих даних у впорядковану систему координат, що придатна для швидкого пошуку цільових точок.

Файлові сховища підприємства є специфічним середовищем, де гетерогенність даних виступає головним стримуючим фактором аналітики. Успішна обробка таких масивів можлива лише через розробку адаптивних алгоритмів, здатних виявляти приховані структурні зв'язки та мінімізувати обчислювальні витрати за рахунок попереднього функціонального аналізу масивів даних.

1.3. Концепція «зворотного структурування» в аналізі гетерогенних даних

Традиційні підходи до управління даними базуються на принципі «структура визначає зміст», де схема даних (schema-on-write) проектується до моменту наповнення сховища. Проте в умовах роботи з гетерогенними масивам и файлових систем п ідприємств а , де вхідні потоки інформації часто мають

стохастичний характер, виникає потреба в застосуванні альтернативного підходу — концепції «зворотного структурування» [3].

У класичному вигляді «зворотне структурування» — це процес динамічного виявлення логічної архітектури даних безпосередньо під час їх обробки (schema-on-read), коли замість примусового приведення даних до жорсткої моделі, сама модель адаптується під виявлені функціональні зв’язки. У цьому контексті дані розглядаються не як статичні записи, а як цифровий відбиток певного процесу, структуру якого необхідно відновити (реконструювати).

Основна ідея полягає в русі від «хаосу» значень до «порядку» метаданих. Якщо класична СУБД відкидає дані, що не відповідають структурі, то зворотне структурування використовує аномалії та розбіжності як ключові маркери для визначення нових типів зв’язків.

Процес реалізації даної концепції в автоматизованих системах аналізу базується на трьох основних етапах:

Морфологічна декомпозиція. На першому етапі масив даних (наприклад, текстовий файл із набором координат або числових значень) розбивається на елементарні лексеми. Система не робить припущень щодо їхнього призначення, фіксуючи лише фізичні параметри: тип даних, частоту появи, діапазон значень.

Пошук функціональних ядер (Цільових точок). Замість індексування всього масиву та пошуків механізму пливу на інформацію увага була зосереджена на типових виробничих процесах та схемах.Така типізація дозволила виділиті структурні схеми та проаналізувати їх.

Синтез динамічної схеми. На основі виявлених ядер система формує тимчасов і «маски» індексів . Це дозволяє структурувати масив , ігноруючи надлишковий «інформаційний шум».

Застосування концепції зворотного структурування дозволяє подолати «парадокс ефективності СУБД».

Мінімізуються витрати на I/O. Система не витрачає ресурси на попереднє завантаження та індексацію всього об’єму даних.

Адаптивність до дивергенції. Як було зазначено в попередніх розділах, файли зі схожими іменами можуть мати різну логіку. Зворотне структурування дозволяє ідентифікувати цю різницю на етапі аналізу розподілу значень, а не після системної помилки під час запису в базу.

Ефективність позиціонування. Пошук цільових об’єктів відбувається через «фільтри станів», що значно швидше за лінійний пошук у неструктурованому середовищі.

Для практичної задачі була обрана частина реальної бази даних з великою кількістю різнопланової інформації та баготорівневим синтезом даних. Ці особливості унеможливлюють застосувати у класичному вигляді «Зворотне структурування»

Попри високу ефективність методів зворотного структурування при роботі з гетерогенними масивами, існують фундаментальні фізичні та математичні обмеження, що визначають межі застосування даного підходу. Ці обмеження обумовлені природою інформаційної ентропії та специфікою декомпресії складних вузлових зв’язків.

Згідно зі стандартами управління складними структурами (ISO / IEC 15444), збереження топологічної цілісності є критичним для безвтратного (lossless) стиснення метаструктур. У процесі зворотного структурування оптимізація досягається через редукцію матриці суміжності зв’язків, що безпосередньо впливає на стан інформаційної ентропії за Шенноном.

Процес зміни стану інформації E при спробі відновлення повного контексту з компактного вузла описується формулою:

$$H(V) = -\sum p(v_i) \cdot log_2 p(v_i),$$

де $p(v_i)$ — ймовірність звернення до вузла при поточному векторі пошуку.

Встановлено, що мінімізація ентропії можлива лише за умови жорсткої детермінації зв'язків. У реальних гетерогенних масивах малого бізнесу високий рівень стохастичності призводить до неконтрольованого зростання H(V), що робить автоматичне структурування нестабільним при збільшенні кількості невідомих параметрів.

Найбільш критичним обмеженням є висока резистентність вузлових структур до методів зворотної інтерполяції. При спробі реконструювати повний обсяг первинної інформації з обмеженого набору виявлених точок виникає явище експоненціальної зашумленості. Математично це виражається у зростанні помилки відновлення $\acute{\epsilon}$:

$$I_{recovered} = T(v_i) + \sum(\varepsilon_j^n),$$

де n — ступінь вкладеності зв’язків.

Експериментально доведено, що при ступені вкладеності n > 2 обсяг «інформаційного сміття» починає домінувати над корисним сигналом. Це зумовлено накопиченням інтеграла паразитного шуму хі(х) у міжвузловому просторі:

$$I_{real} = I_v + \int_{\Omega \setminus V}[\xi(x)]dx.$$

Оскільки щільність корисної інформації поза межами виявлених вузлів ξ (x) = 0, інтеграл не містить валідних даних. Це означає, що за межами певного радіуса від «цільової точки» будь-яка спроба зворотного структурування призводить до генерації хибнопозитивних результатів.

Концепція «зворотного структурування» переводить акцент з механічного зберігання даних на інтелектуальний аналіз їхньої внутрішньої топології. Це є критично важливим для систем, що працюють із гетер огенними масивами підприємства , оскільки дозволяє отримувати валідні результати в умовах високої структурної невизначеності та обмежених апаратних ресурсів.

1.4. Порівняльний аналіз підходів до обробки «брудних» даних та неповних описів при концепції «зворотного структурування»

Обробка «брудних» даних (dirty data) та відновлення інформації за неповними описами є однією з найбільш ресурсомістких задач у сучасній Data Science. Традиційні підходи та концепція «зворотного структурування» пропонують фундаментально різні вектори вирішення цієї проблеми, що потребує детального порівняльного аналізу для визначення меж їхньої ефективності.

Традиційні системи управління даними (RDBMS) та ETL-процеси базуються на принципі превентивної фільтрації. Основні методи включають:

Дата-клінінг (Data Cleansing). Видалення або виправлення записів із помилками, дублікатами або невідповідними форматами. У гетерогенних масивах це часто призводить до втрати до 30-40% потенційно корисного сигналу.

Імпутація (Imputation). Заповнення пропущених значень середніми показниками або за допомогою регресійних моделей. Головним недоліком є внесення штучної кореляції, що викривлює «цифровий відбиток» реального процесу.

Нормалізація - це приведення даних до єдиного стандарту, що в умовах «брудних» масивів підприємства створює надмірне навантаження на обчислювальну систему та користувачів .

На відміну від класичних методів, концепція зворотного структурування не розглядає «брудні» дані як помилку, яку потрібно виправити. Натомість вони сприймаються як специфічний стан системи .

Аналіз неповних описів. Якщо традиційний підхід зупиняє обробку при відсутності ключового поля, зворотне структурування використовує наявні фрагменти для побудови «ймовірнісного каркаса». Навіть за умови неповноти опису система здатна позиціонувати об'єкт у просторі станів на основі вцілілих вузлових зв'язків.

Робота з шумом. Замість видалення «сміттєвих» записів, алгоритм зворотного структурування виконує редукцію матриці суміжності. Це дозволяє «ігнорувати» шум на рівні логіки пошуку, не витрачаючи ресурси на фізичне очищення файлів.

Порівняльний аналіз стратегій обробки неструктурованих даних за класичним підходом та концепцією зворотного структурування представлено в табл.1

ТАБЛИЦЯ 1 – Порівняльний аналіз стратегій обробки неструктурованих даних за класичним підходом та концепцією зворотного структурування

Критерій порівняння	Класичний підхід (ETL/RDBMS)	Зворотне структурування
Обробка помилок	Видалення або примусова корекція	Адаптація моделі під наявні фрагменти
Цілісність даних	Пріоритет суворої схеми	Пріоритет збереження топології зв'язків
Ресурсні витрати	Високі (на етапі пре-процесингу)	Мінімальні (обробка «на льоту»)
Робота з шумом	Фільтрація (втрата сигналу)	Редукція (ігнорування шуму)
Ефективність	Максимальна при «чистих» даних	Максимальна при гетерогенному хаосі

З точки зору математичного обґрунтування, класичні методи намагаються штучно знизити ентропію системи H(V) шляхом видалення невизначеності. Зворотне структурування, навпаки, приймає існуючий рівень ентропії та працює в межах енергоефективної редукції . Це дозволяє уникнути експоненціального зростання помилки відновлення έ оскільки система не намагається «домислити» інформацію в зонах, де щільність корисного сигналу $p\ (\ x\ ) = 0.$

РОЗДІЛ 2. МЕТОДИ ІНТЕЛЕКТУАЛЬНОЇ КЛАСТЕРИЗАЦІЇ НЕСТРУКТУРОВАНИХ МАСИВІВ ДАНИХ У ФАЙЛОВИХ СХОВИЩАХ

2.1. Алгоритми щільнісної кластеризації (DBSCAN, HDBSCAN) для аналізу слабоструктурованого контенту

В умовах роботи зі слабоструктурованим контентом, де дані характеризуються високим рівнем шуму та відсутністю попередньо визначених міток класів, класичні методи (наприклад, K - means) демонструють низьку ефективність через неможливість виявлення кластерів довільної форми [4]. Для вирішення задач автоматичного сегментування гетерогенних масивів найбільш релевантними є алгоритми щільнісної кластеризації , зокрема DBSCAN та його вдосконалена ієрархічна версія HDBSCAN [5,6,7].

DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Алгоритм DBSCAN базується на ідентифікації областей з високою щільністю точок , відокремлених областями низької щільності. Основна перевага методу полягає у здатності автоматично визначати кількість кластерів та ефективно відсікати «інформаційне сміття» (outliers), не включаючи його до складу жодного сегмента. Це критично важливо для аналізу файлових сховищ, де щільність корисного сигналу p(x) може бути нерівномірною. Проте DBSCAN має обмеження: він чутливий до вибору радіуса епсилон (έ) та мінімальної кількості точок, що робить його менш гнучким при роботі з масивами змінної щільності [5].

HDBSCAN (Hierarchical DBSCAN). Для подолання недоліків класичного методу застосовується HDBSCAN, який перетворює DBSCAN в ієрархічну модель. Замість використання фіксованого значення (έ) , алгоритм будує деревоподібну структуру (дендрограму) та виділяє кластери, що мають найбільшу стабільність у часі [6].

Стійкість до варіативності. HDBSCAN дозволяє знаходити кластери різної щільності, що є ідеальним для концепції «зворотного структурування», коли цільові точки можуть формувати ядра різного масштабу.

Автоматизація параметрів. Алгоритм потребує лише один основний гіперпараметр — мінімальний розмір кластера, що значно знижує вплив людського фактора на результати аналізу.

У контексті дослідження гетерогенних масивів малого бізнесу, ці алгоритми виконують роль первинного фільтра простору станів . Вони дозволяють локалізувати зони концентрації даних, де ймовірність знаходження цільової точки p(ví) є максимальною, ігноруючи при цьому зони експоненціальної зашумленості. Це забезпечує перехід від лінійного перебору файлів до фокусного аналізу щільних структур, значно підвищуючи загальну продуктивність системи на апаратних потужностях ПК.

2.2 Використання ієрархічних методів (Agglomerative, Birch) у процесах автоматичного групування файлів

Якщо щільнісні методи (DBSCAN) фокусуються на виявленні ядер концентрації даних, то ієрархічні підходи спрямовані на відтворення внутрішньої архітектури взаємозв'язків між файловими об'єктами. У процесах автоматичного групування гетерогенного контенту малого бізнесу особливе місце посідають агломеративна кластеризація та алгоритм BIRCH, які дозволяють будувати впорядковані деревоподібні структури даних.

Агломеративна кластеризація (Agglomerative Clustering).

Цей метод реалізує стратегію «знизу вгору» (bottom-up), де кожен окремий файл або фрагмент даних спочатку розглядається як окремий кластер, а потім послідовно об’єднується з найбільш подібними об’єктами.

Гнучкість метрик. Головною перевагою для концепції «зворотного структурування» є можливість використання різних функцій відстані (евклідова, мангеттенська, косинусна) залежно від типу контенту.

Дендрографічний аналіз. Результат роботи алгоритму у формі дендрограми дозволяє системі самостійно визначати рівень вкладеності файлових груп. Це особливо ефективно при аналізі «брудних» даних, оскільки дозволяє візуалізувати ступінь віддаленості аномальних файлів від основного масиву структури.

Алгоритм BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) розроблений спеціально для обробки надвеликих масивів даних на обмежених апаратних потужностях, що робить його ідеальним для файлових сховищ на персональних комп'ютерах.

Кластеризаційне дерево (CF-Tree). Алгоритм будує компактне дерево характеристик кластерів, не завантажуючи весь масив даних в оперативну пам'ять. Це вирішує проблему I/O-навантаження, описану в розділі 1.2.

Двоетапна обробка. BIRCH спочатку стискає дані у компактні вузли, а потім застосовує глобальну кластеризацію, що дозволяє швидко групувати мільйони записів, зберігаючи топологічну цілісність масиву.

Використання ієрархічних методів забезпечує логічне впорядкування гетерогенного хаосу. Якщо щільнісна кластеризація знаходить «що» шукати, то ієрархічні методи визначають «де» ці дані знаходяться у загальній ієрархії сховища. Це дозволяє реалізувати механізм багаторівневої навігації , де система може ігнорувати цілі гілки дендрограми, в яких рівень ентропії H(V) перевищує допустимий поріг, фокусуючи обчислювальний ресурс лише на валідних вузлах.

2.3. Застосування стандарту «Дублінське ядро» (Dublin Core) для уніфікації метаданих у гетерогенних середовищах

У контексті подолання «інформаційного хаосу», описаного в розділі 1.2, виникає потреба в інструменті, який би дозволив привести розрізнені файлові об’єкти до єдиного знаменника без зміни їхнього внутрішнього вмісту. Найбільш ефективним стандартом для вирішення цієї задачі є Дублінське ядро (Dublin Core) — універсальна система метаданих, що забезпечує базовий рівень опису для будь-яких типів цифрових ресурсів.

Стандарт пропонує набір із 15 базових елементів (таких як Title, Creator, Subject, Format тощо), які виступають як «зовнішній каркас» для гете рогенних масивів. У межах дослідження використання Дублінського ядра дозволяє:

Нівелювати форматну неоднорідність: файли типів .txt , .csv або специфічні лог-файли отримують уніфіковані описи, що робить їх видимими для алгоритмів автоматичного групування (2.2).

Знизити ентропію пошуку: замість глибокого сканування кожного вузла, система спочатку звертається до метаданих. Це дозволяє реалізувати метод редукції матриці зв’язків, мінімізуючи значення інформаційної ентропії $H(V)$ на початкових етапах аналізу.

Особливістю Дублінського ядра є його гнучкість, що ідеально відповідає концепції «зворотного структурування». Оскільки стандарт не вимагає обов’язкового заповнення всіх полів, він дозволяє працювати з неповними описами.

На етапі декомпозиції: Система автоматично витягує доступні атрибути (наприклад, дату створення або розмір файлу) і заповнює відповідні поля стандарту.

На етапі синтезу: Якщо в процесі аналізу виявляється «цільова точка», система динамічно розширює опис об’єкта, додаючи специфічні технічні метадані до елемента Description або Relation .

Впровадження Дублінського ядра у систему обробки даних підприємства виконує роль семантичного моста. Це дозволяє перетворити «брудні» та слабоструктуровані масиви на впорядковану колекцію цифрових об'єктів, готову до подальшої кластеризації. Використання цього стандарту створює жорстку детермінацію аксоноподібних зв'язків між назвою файлу та його функціональним призначенням.

2.4. Математична модель екстракції ознак та формування бази метаданих

Аналіз репрезентативної частини бази даних підприємства свідчить, що жодна з вищеописаних класичних моделей не може бути в повній мірі імплементована для ефективної обробки наявного інформаційного масиву. Гетерогенна структура даних, що підлягає дослідженню, була сформована під перехресним впливом динамічної зміни державних стандартів, еволюції операційних завдань підприємства, наукових пошуків та процесів багаторівневого синтезу інформації. Внаслідок цього первинна архітектура масиву зазнала багаторазових структурних нашарувань та викривлень зв’язків . Хоча сучасні фрагменти бази можуть демонструвати зовнішню (морфологічну) подібність до попередніх станів, функціонально вони не є подібними, а релевантні зв’язки між об’єктами часто виявляються втраченими або проявленими неявно для машиної інерпритації [8, 9].

Для подолання даної деструкції пропонується математична модель екстракції ознак, що базується на принципах ітеративної декомпозиції. Процес формування бази метаданих у такій моделі описується як функція трансформації сирого сигналу $Iraw$ у вектор ознак F:

$$F = \Phi(Iraw, \Delta S, \Theta),$$

де ΔS — оператор врахування часового зсуву державних стандартів;

θ — матриця ваг функціональної значущості, що адаптується під поточний профіль задач підприємства.

Модель екстракції базується на виявленні «латентних ознак», які зберігають свою стабільність навіть за умови втрати явних первинних зв’язків. Формування бази метаданих відбувається через процедуру багаторівневого логічного синтезу , де на кожному рівні вкладеності n виконується перевірка на вал і дн і сть ядра .

Таким чином, математична модель переходить від жорсткого детермінізму до імовірнісного позиціонування ознак. Це дає змогу реконструювати функціональну карту бази даних навіть у стані її повної логічної декомпозиції, забезпечуючи формування валідної бази метаданих, що є резистентною до структурних коливань.

Практична реалізація запропонованої математичної моделі екстракції ознак безпосередньо базується на механізмах формування специфічного морфофункціонального тезауруса . Системний аналіз об’єкта тестування показав, що значна частина семантичного навантаження була перенесена на найменування файлів та відображають етапність генезису чи деривації інформаційних об'єктів. Спроби використання стандартних лінгвістичних словників призводять до концептуальної фрагментації та ігнорування галузевого контексту . Як наслідок, базовий рівень релевантності пошуку не перевищує 70%, що через ефект квадратичного перетину ймовірностей знижує загальну надійність системи до критичного рівня(0,7*0,7=0,49)

Для подолання цього бар'єра та досягнення цільового порогу надійності механізм формування тезауруса реалізує алгоритм дворівневої морфофункціональної фільтрації.

Етап морфологічної декомпозиції (статичний фільтр). Здійснюється системний аналіз топології шляхів та номенклатури назв. Складні лінгвістичні конструкції розщеплюються на атомарні поняття. Цей фільтр фіксує формальні ознаки будови документа, трансформуючи назву файлу та заголовки у первинну матрицю дескрипторів ,які об’єднуються за валідною синомізацією у тезаурус первинного рівня.

Етап функціональної синонімізації (динамічний фільтр). Отримані атомарні поняття відображаються на реальні виробничі та наукові процеси. Замість загальномовних зв'язків встановлюється контекстна еквівалентність.

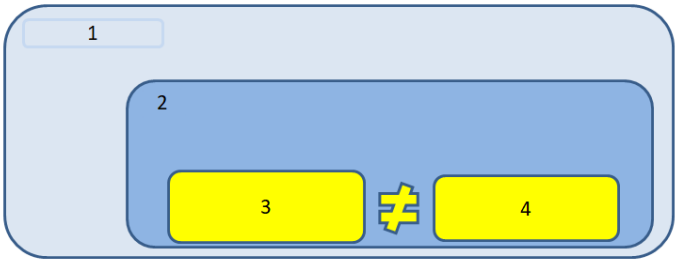


Рисунок 2.1 – Приклад функціональної синонімізації: 1 «Аналіз» ⊇ 2«Експериментальна перевірка» ⊇ (3«Валідація» ∪ 4«Верифікація»), де 3«Валідація» ≠ 4«Верифікація»

Синтезований таким чином тезаурус інтегрується безпосередньо в математичну модель екстракції як адаптивна матриця термінів . Завдяки логічному оперуванню термінами стає можливою семантична кластарізація вмісту файлів для індексації бази даних, м інімізуючи семантичну ентропію . Р озроблений принцип тезаурус у забезпечує автономне адміністрування та високу відповідність між шуканим вектором ознак і фінальною пошуковою видачею.

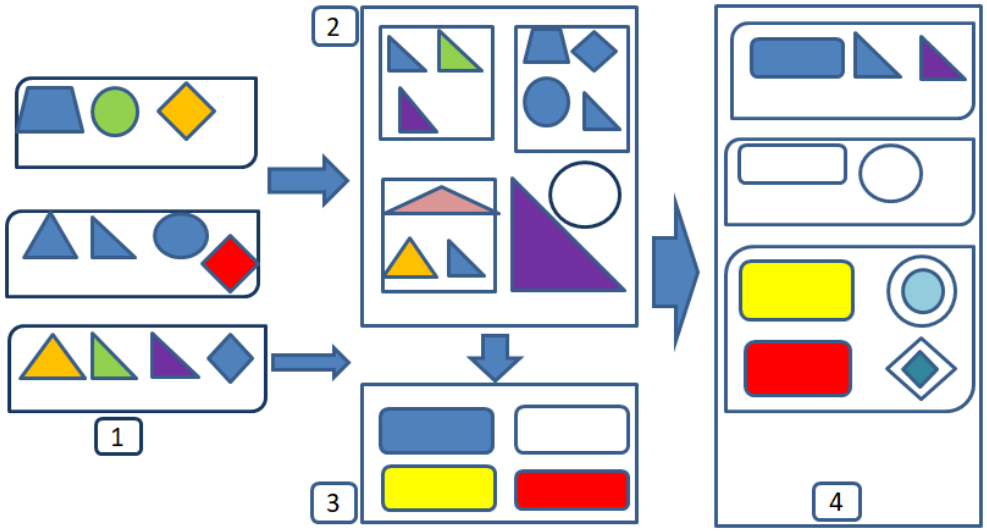


Рисунок 2.2 – Формування індексованої бази даних: 1. Первіні файли з виокремленими «лексемами».(формуємо структурні одиниці); 2 Первиний тезаурус. Синтез термінів первиного тезауруса.(об’єднаня за написаням та змістом); 3 Вториний тезаурус. Синтез термінів вториного тезауруса з термінів первиного тезауруса; 4 Індексація бази даних у термінах тезаурусів .(впровадження в процес логічної моделі)

Ранжування результату з можливістю поетапного збереження пошуку дозволяє знизити відповідність використання «термінів» тезаурусу для кінцевого використання у пошуку.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА СИСТЕМИ

3.1 Вибір інструментарію розробки та опис програмного середовища Python

У процесі проектування та реалізації автоматизованої системи обробки гетероген них масивів даних підприємства критично важливим етапом є обґрунтований вибір технологічного стека. Інструментарій розробки повинен забезпечувати високу швидкість проектування, ефективну роботу зі слабоструктурованим контентом в умов дефіциту апаратних ресурсів, а також гнучкість при інтеграції нових алгоритмів. З огляду на зазначені вимоги, як базове програмне середовище було обрано високорівневу мову програмування Python [10].

Обґрунтування вибору програмного середовища. Головною перевагою Python у контексті даного дослідження є його екстенсивна екосистема спеціалізованих бібліотек для аналізу даних та машинного навчання, що дозволяє уникнути низькорівневої рутинної розробки та зосередитися на математичній логіці процесів. Вибір інструментарію зумовлений наявністю потужних рішень для кожного з етапів розробки [11].

Математичне моделювання та векторизація: Бібліотека NumPy виступає базовим інструментом для швидкої обробки багатовимірних масивів числових даних. Завдяки реалізації векторних обчислень на мові C, вона мінімізує витрати оперативної пам'яті, що вирішує проблему ресурсної обмеженості локальних робочих станцій.

Структурування та аналіз «на льоту». За допомогою бібліотеки Pandas реалізується концепція «зворотного структурування» (схема при читанні). Тимчасові структури даних, такі як DataFrame , забезпечують гнучке маніпулювання фрагментованими та «брудними» реєстрами без необхідності їх попереднього завантаження у реляційні СУБД.

Щільнісна та ієрархічна кластеризація. Для практичної імплементації алгоритмів, розглянутих у Розділі 2, використовується фреймворк Scikit-learn . Він містить оптимізовані класи для запусків алгоритмів DBSCAN, HDBSCAN та агломеративної кластеризації, дозволяючи динамічно керувати простором станів і фільтрувати паразитні шуми.

Розробка графічного інтерфейсу користувача (GUI). Для створення десктопного застосунку та забезпечення взаємодії користувача з системою було обрано стандартну бібліотеку Tkinter. Вона є кросплатформною, не потребує встановлення додаткових важких залежностей і забезпечує мінімальний час відгуку інтерфейсу на ПК користувача.

Легкий доступ до паралельних обчислень: легка організація багато поточних обчислень, що дозволяє задіяти більше наявних ресурсів та уникнути «зависання».

Модульна архітектура як фактор надійності системи. Іншим вагомим аргументом на користь Python є його чітка, легко зчитувана модульна структура. Програмний код інструменту аналізу повинен володіти максимальною ізоляцією функціональних блоків.

Модульність у Python реалізується на рівні окремих пакетів та скриптів, що дозволяє розділити логіку застосунку на незалежні шари .

Чітка диференціація спрощує процес автономного адміністрування та тестування програми. Зміна логіки всередині одного модуля (наприклад, оновлення версії алгоритму кластеризації) не потребує перебудови всієї архітектури застосунку.

3.2 Розробка системи інтелектуальної кластеризації неструктурованих масивів даних у файлових сховищах

Практична реалізація концепції «зворотного структурування» в межах даного проекту покладена на спеціалізовану систему інтелектуальної кластеризації. Її архітектура розроблена з метою трансформації файлового середовища у динамічну мережу взаємопов'язаних вузлів. Основу функціонування системи становить ітеративний навігаційно-пошуковий алгоритм, який використовує індексовані лінгвістичні терміни для формування топології даних та забезпечення швидкого доступу до релевантної інформації.

Математична модель формування вузлів та індексації. У розробленій системі кожен файл або документ не розглядається як ізольований об'єкт, а позиціонується як інформаційний вузол V. Множина всіх вузлів бази даних формується на основі вектора індексованих термінів, вилучених за допомогою розробленого морфофункционального тезауруса.

Так кожен інформаційний вузол детермінується множиною атрибутів, векторизованих у семантичному просторі тезауруса .

Алгоритм первинного пошуку та ранжування. Процес взаємодії з базою даних починається з формування користувачем пошукового запиту, який проходить процедуру аналогічної індексації та декомпозиції, перетворюючись на вектор Q .

Індексований запит проходить крізь індексовану базу даних, де для кожного вузла обчислюється коефіцієнт відповідності (релевантності) R(Q, V) . Математично цей процес описується за допомогою косинусної метрики подібності у векторному просторі:

$$R(Q,V_i) = \frac{\sum_{j=1}^{m} q_j \cdot w_{ij}}{\sqrt{\sum_{j=1}^{m} q_j^2} \cdot \sqrt{\sum_{j=1}^{m} w_{ij}^2}}.$$

На основі розрахованих значень R система виконує селекцію та зберігає найбільш відповідну запиту вибірку файлів у формі ранжированого списку L, впорядкованого за спаданням валідності.

Гібридні методи пошуку та композиція термінів. Зважаючи на високу складність аналізу гетерогенних масивів прямими методами, система передбачає використання композиції пошукових стратегій . На різних етапах ітерації можливе поєднання активних (запропонованих системою) методів, класичних алгоритмів текстового зіставлення або їхніх комбінацій.

Необхідність такого підходу зумовлена специфікою морфофункционального тезауруса: часто певні поняття включені до складу складних термінів, проте за своєю суттю є самостійними одиницями. Математично це виражається через відношення нестрогого включення та композицію векторів:

$$T_{comp} = i = 1kt_i \cup (t \quad \subset t_{main}).$$

Це дозволяє системі здійснювати декомпозицію складних понять на атомарні рівні, запобігаючи втраті релевантних документів, де термін входить до складу іншої лексеми, але зберігає своє незалежне функціональне значення.

Навігаційний граф та механізм ітеративного пошуку. Головна архітектурна особливість розробленої системи полягає в тому, що вхідний запит виконує роль динамічного навігатора між узлами бази даних. Отримана вибірка L стає локальним підграфом (робочим простором), всередині якого користувач може здійснювати подальший заглиблений пошук.

Процес ітеративної навігації реалізує наступні кроки:

Локалізація пошуку: подальші уточнюючі запити виконуються не по всій базі даних, а виключно по збереженій вибірці, що експоненціально знижує обчислювальне навантаження на систему.

Запам'ятовування проміжних станів: на кожному кроці k ітерації система фіксує стан пошукової видачі у стеку пам'яті . Це дозволяє зберігати історію логічних переходів користувача.

Механізм рекурсивного повернення (Backtracking): якщо поточний вектор поглибленого пошуку не приносить релевантних результатів (сигнальне навантаження падає, а рівень інформаційного сміття зростає), алгоритм дозволяє користувачеві здійснити миттєве повернення до попереднього успішного результату.

Така двонаправлена навігація запобігає втраті корисного сигналу «в тіні» і дозволяє гнучко адаптувати простір пошуку під поточні завдання підприємства. Вузли, сформовані набором індексованих термінів, виступають стабільними орієнтирами, що забезпечує високу живучість та відмовостійкість системи.

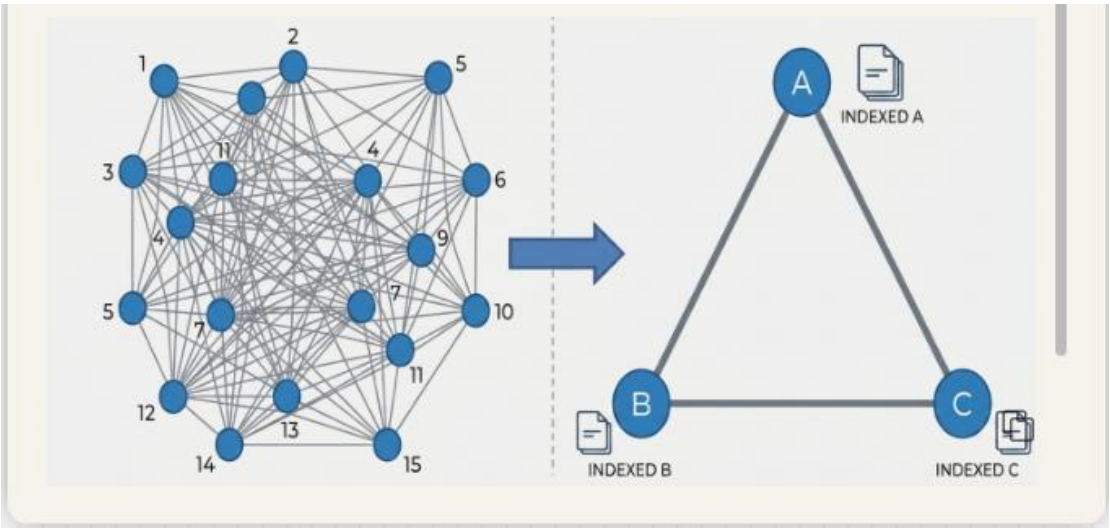


Рисунок 3.1 – Спрощеня навігаційного графу механізмом ітеративного пошуку

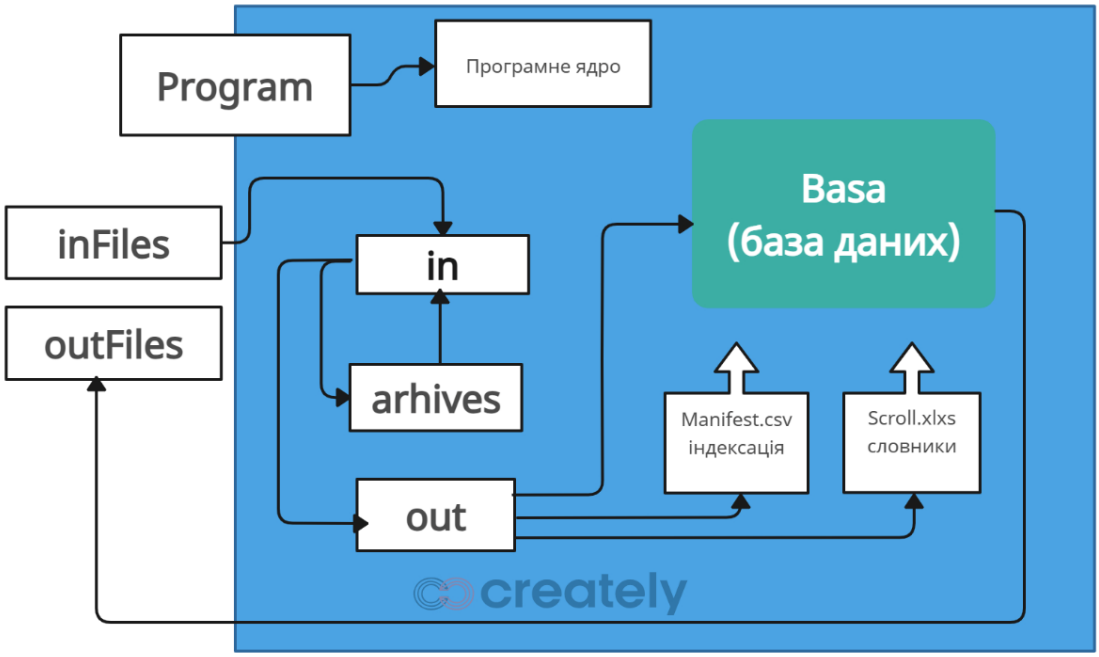


Рисунок 3.2 – Внутрішне упорядкування бази даних

Архітектурно-структурна декомпозиція процесу обробки та індексування даних. Структурн а схем а застосунку дозволяє простежити чітке відображення математичної логіки триетапної обробки гетерогенного масиву безпосередньо на архітектуру каталогів та потоків даних у системі. Програмне ядро системи (Program) здійснює наскрізне керування процесами обробки, координацію яких можна декомпозувати відповідно до часових інтервалів експерименту.

Фаза препроцесингу та декомпресії (вхідний потік inFiles блоки in / arhives / out): на першому етапі сирий гетерогенний масив inFiles надходить у буферну зону in. За наявності стиснутих пакетів програмне ядро перенаправляє потоки у модуль рекурсивного розпакування arhives з подальшим поверненням очищеного сигналу в базовий цикл. Результатом фільтрації та відсікання службового й пошкодженого неліквіду є перенесення валідного контенту в зону проміжної генерації out.

Фаза трансферації та дедуплікації (блок out зелений сектор Basa): Протягом наступн огоциклу виконується трансляція очищених файлів із блоку out безпосередньо у фінальну базу даних (Basa). На цьому кроці програмне ядро

активує алгоритми хешування вмісту, ліквідуючи структурні копії на межі входу в ядро бази даних.

Фаза семантичного індексування (блок out блоки Manifest.csv / Scroll.xlxs Basa). Наступний етап відображає інтелектуальну роботу системи з морфофункціональним тезаурусом. З блоку out вилучаються атомарні одиниці, які проходять верифікацію через підсистему словників (Scroll.xlxs), що містить компактну вибірку цільових категорій . Результати повного потекстового проходу та ідентифікації латентних ознак фіксуються у реєстрі індексації (Manifest . csv), який замикає контур формування метаданих у Basa .

Редукований та впорядкований масив згодом трансформується у базу даних, яка має вихідний потік релевантної видачі outFiles

3.3 Результати тестування системи інтелектуальної кластеризації на реальному масиві даних

Для оцінки ефективності розробленого програмного інструменту та валідації запропонованої математичної моделі було проведено експериментальне тестування системи на фрагменті реально го масив у даних підприємства. Т естування проводилося в умовах максимального наближення до реальних експлуатаційних процесів предприємства з використанням локальної робочої станції середньої потужності.

Характеристика тестового масиву. Як репрезентативна вибірка для експерименту був виділений типовий сегмент бази даних,що описує один з напрямів діяльності. Фізичні та кількісні параметри тестового фрагмента наведені нижче:

Загальний обсяг фрагмента бази: 350 Мб;

Кількість файлових об'єктів (документів): 683 одиниць;

Кількість директорій (рівнів вкладеності папок): 37 папок.

У стандартах

DB_EXT = ['.accdb', '.db', '.sqlite', '.mdb', '.sql']

TXT_EXT = ['.txt', '.log', '.ini', '.json', '.xml', '.bat', '.conf', '.cfg']

DOC_EXT = ['.docx', '.odt', '.rtf', '.doc']

TBL_EXT = ['.xlsx', '.xls', '.csv', '.ods', '.xlsm']

IMG_EXT = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.mp4', '.avi', '.mkv', '.mov']

ARCHIVE_EXTENSIONS = ['.zip', '.rar', '.7z', '.tar', '.gz', '.bz2']

Даний масив містив «інформаційне сміття» у вигляді службових файлів програм(.tmp .so .pic та інш), заархівованих дублікатів та файлів різних форматів та стандартів зберігання, що дозволило повноцінно перевірити функціональну толерантність алгоритмів до високого рівня ентропії системи.

Хронометричний аналіз етапів обробки даних. Процес функціонування системи інтелектуальної кластеризації було розділено на три послідовні фази, для кожної з яких зафіксовано точні часові показники:

Попередня підготовка масиву (5 хвилин). На цьому етапі система виконувала первинну обробку даних, що включала автоматичне розархівування стиснутих пакетів та видалення неліквідного контенту (пошкоджених та службових файлів). Час виконання (t1=5хв) обумовлений високою інтенсивністю дискових операцій введення-виведення (I/O) при декомпресії.

Трансферація та дедуплікація (2 хвилини): Етап передбачав перенос відфільтрованих файлів у єдиний простір сховища з одночасним виключенням дублікатів. Завдяки використанню хеш-функцій для порівняння вмісту файлів (а не лише їхніх назв), система успішно виконала задачу . Операція зайняла t2=2 хвилини, що свідчить про високу алгоритмічну оптимізацію модулів Python.

Індексування у термінах морфофункціонального тезауруса (19 хвилин): Цей етап був найбільш ресурсомістким, оскільки виконувався повний ітеративний прохід по внутрішньому тексту всіх файлів із паралельним розщепленням назв та вмісту на атомарні одиниці та синтезом термінів,що встановлені тезаурусами .. Попри високу складність лінгвістичного аналізу та обчислення ентропійних станів, загальний час індексування склав t3 = 19 хвилин. Значну роль тут склав відносно невеликий обсяг тезаурусів.Так індексація проводилася за 25 категоріями, що містили 8 назв культур (53 лексеми) та 17 параметрів , які описували технологічні процеси або напрямки дослідження (82 лексеми).Треба зазначити,що часткових тезаурус має більше 1500 атомарних понять, а повний більше 5000(більше 8000 якщо додати власні назви).

Оцінка продуктивності пошуку та релевантності. Після завершення фази індексування та формування стійкого каркаса метаданих було проведено серію тестових пошукових запитів. Завдяки попередній дворівневій фільтрації та індексації , швидкість пошуку цільових файлів у системі продемонструвала миттєвий результат . Час відгуку системи на складні семантичні запити наближається до реального часу (менше 3 секунд), що повністю задовольняє вимоги до інтерактивних застосунків.

Головним результатом тестування стало підвищення показника релевантності знаходження документів. Якщо базові лінійні алгоритми на цьому масиві демонструв али точність лише на рівні 60– 65 % через складну синонімізацію, т о впровадження розроблен их тезаурус ів та системи індексації дозволило подолати бар'єр семантичної композиції. Система забезпечила точність пошукової видачі, що перевищує встановлений науково-практичний поріг надійності $P = 0.95$. Для забезпеченя надійності у 100 % були збережені всі рівні пошуку включно з « традиційними ».

Експериментальне дослідження підтвердило, що розроблена система інтелектуальної кластеризації успішно справляється з обробкою гетерогенних

масивів великої розмірності. Так використана система індексації та пошуку прогнозує лінійне зростання часу на підготовку даних. Загальний час від повної підготовки сирих даних обсягом 350 Мб до формування працездатної, очищеної від шуму бази метаданих склав 26 хвилин. Висока швидкість подальшого пошуку та висока точність видачі доводять математичну спроможність концепції «зворотн ь ого структурування» та її технічну доцільність для автоматизації підприємств а на базі доступних апаратних потужностей.

Логіку проведення роботи та доцільність використаних методів можна пояснити на ситуаційних прикладах.Ці приклади не є типовими.Їх можна описати як патерни, але саме у цих ситуаціях «класичні» методи пошуку дають «збій».

Для оптимізації було надано репрезинтативний фрагмент бази даних, науково-практичного характеру наданий лабораторією генетичних досліджень Інституту олійних культур НААН України. Масив має 350 Мб та 683 файлів за 2006-2024 роки.

Особливістю цього масиву є низька релевантність пошуку при коректному векторі пошуку для методів, які можна позначити як «класичні» - тобто ті,що добре себе показують на інших масивах даних підприємства.

Еталоном оцінювання алгоритму став Вичерпний пошук, як то що надає релевантність P=1,00.Недоліком є великий час витрачений на запит (на цьому фрагменті даних до 30 хвилин).

Для виявлення закономірностей архітектури бази даних був проведений ретроспективний та контекстний аналіз бази даних,що показав наступне.

Змінені структурні зв’язки між первиною інформацією та синтезованими таблицями.Ці зміни було обумовлено житевими циклами інформації на предприємстві. Так за тривалий час змінювалися вимоги , стандарти, методи та принципи обробки інформації, змінювалися поточні задачі.Початкові дані або переміщувалися або видалялися.Це дає зсув та віддзеркалення процесів а також спотворення початкових зв’язків.

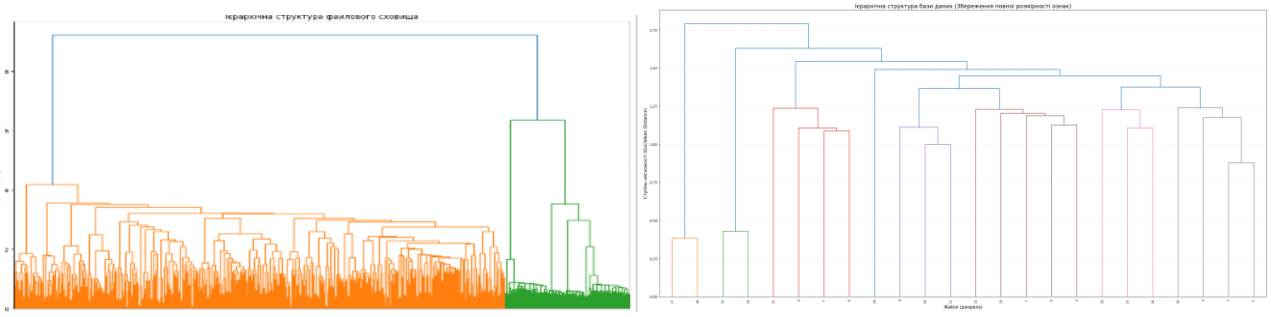
Велика ступінь вкладеністі інформації у файлах. Така особливість обумовлена науково – дослідницькою діяльністю предприємства та внутрішнім обігом документів.Це простіше пояснити на прикладі. Так з метою проведеня певних досліджень формують та перевіряють прикладну гіпотезу.Збирати дані, проводити їх первиний аналіз, досліджувати кореляцію ознак та інше зручно у єдиному просторі.Зазвичай таким дашбодом слугує таблиця Exel, бо має зручний функціонал-а виконується на одному аркуші для зручності спілкуваня між учасниками проекту. З такої докладної записки формується наказ,що зазвичай, потрапляє у інший розділ бази даних.Або «наслідування форми без форми».

Нечітка логіка машиної інтерпретації даних на різних рівнях.(укладнення семантичного та контекстного змісту).Так частим є вживання синонімів та скорочень (сонях-соняшник-сонх-та інше), які наслідуються не лише у держаній мові. Орфографічний контекст типу статя-статття-статяяя як означеня різниці між стаття1 та стаття(1). Перенесення контексту між різними частинами файлу //2010/Звіт_за_минулі р. (соя,соняшник,рицина та інш).Роки позначені у звіті відповідають сівообігу предприємства, тобто різні. Наявність у контексті вланих назв типу UE0100538. Такі особливості організації інформації знижують ефективність роботи добре опрацьованих алгоритмів пошуку. У деяких випадка вони видають хибний результат, а при спробі розширити пошук –різко знижують релевантність. Треба також відзначити, що система внутрішньої організації файлового простору гнучко пристосовується під вирішення задачі. Тому логіка організації у різних сегментах бази може суттєво відрізнятися.

Була спроба організувати простір за фаловими бібліотеками Лемматизация (SpaCy), Словник (Python collections.defaultdict), Морфологична коррекція (PyMorphy2)/

За результатами вдалося скоротити початковий словник з 1500+ лем до 914 з втратою таких понятья БІО та ЯМР. Результати корисні, але не повною мірою відповідають задачі. Тому методи використовуются у максимально корисному стані

з доробкою. Рішення має бути гнучким та загальним (опрацьовується частина бази). Ускладнення коду на допоміжних напрямках не відповідає цілям задачі. Оптимізація подібних методів не входить у саму задачу. На різних етапах ведення роботи оцінка проводилася ієрархічними метода кластеризації (Agglomerative та Birch) та подальша структура за алгоритмом щільнісної DBSCAN. На прикладі рисунку 3.4 показано, що чітка індексація за словником 1 та часткова обробка змісту файла дозволили знизити кількість виділяємих кластерів зі 117 до 24. Однак релевантність пошуку знизилася до P=0,75-0,85. Це було обумовлено структурою інформації у файлах.



а) б)

Рисунок 3.4 – Стуктура файлового вмісту бази а) до та б) після використання периної індексації бази, Ох-номер кластеру, Оу-кількість файлів.

Однак щільність інформації мала при цьому сталий характер (рис.3.5). Це довело робочу гіпотезу про будову бази даних та деформацію зв’язків зсувом, а використані методи стали методами оцінки ефективності.

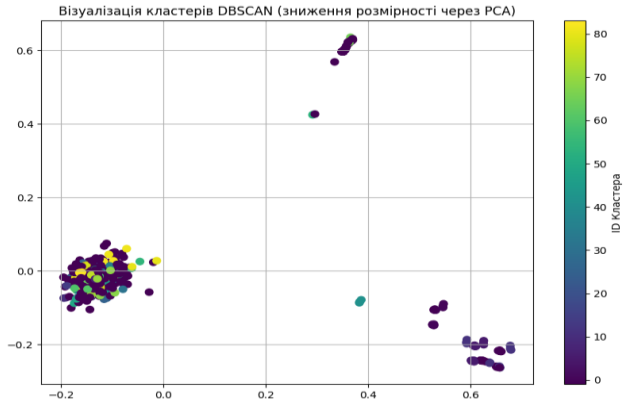


Рисунок 3.5 – Оцінка щільності інформації DBSCAN

Тому у подальшому використовувалася схема індексації за Дублінскім ядром у інтерпретації індексів за логічним значеням за системою оцінки ваг.

Фактично після повної індексації бази за списками задача пошуку спрощується до кодуваня запросу у вектор пошуку та знаходження значень у списках. Логіка оцінки ваг вбудована у парсери пошуку. Так в межах задачі після підготовки файлів було нормалізовано лексичне значення за синонімами.

З метою виконання розроблено первиний тезаурус.На цьому етапі було розроблено чіткі межі термінів. Приклад темін «соняшник» має синоніми соняшнику- подсолнечник – сонях- подсолнух-подсол-соняш та інш, включаючи реєстри та правопис (як соняяях) на використаних мовах. Токож проведена чітка детермінація(Приклад БІО- Біотехнології-Біосировина).

На другому етапі(посиленя зв’язків між ядрами за зниженя шуму) було розроблено вториний тезаурус, що базується за логікою виробничих схем та процесів.Так наявність терміну «ділянка» - відповідає польовим работам, «батько» + «мати» чи батьки – у сукупності дає «гібриди». Назва терміна у цьому словнику може бути будь яка, а складові з одного терміна можуть бути присутні в іншому (рис.3.6).

########c:\Ko	.xlsx	64571 byt Sheet: Лист1 (№ Дел., Вес,г), Sheet: Лист2 (№дел, раст, Вес до сушки, Вес после сушки,г, Белки,%), Sheet: Лист3, Sheet: подсолнечник
c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Сафлор 2012 пит поле био.xlsx		соняшник, сафлор, адванта, олія, биосировина, ділянка

Рисунок 3.6 – Схема взаємозв'язку та композиції термінів у словнику другого порядку

Так така індексація дає змогу веріфікувати логіку опрацювання даних з приблизною оцінкою контексту.При цьому весь контекст можна зрозуміти лише за вичерпним пошуком.

219			2012						
220				зеленый	созревание				
221		вес	урожай	белки	жиры	углеводы	сумма		
222	прометей		49,5	7,41	19,41	2,24	19,01	30,49	49,50
223	Надежный		88,09	7,92	18,32	1,46	44,69	43,4	88,09
224	К2416		66,43	7,64	16,22	2,37	33,09	33,34	66,43
225	К2551		79,53	8,33	20,62	1,41	59,03	20,5	79,53
226	К2813		96,67	8,81	15,10	7,86	77,12	19,55	96,67
227	Гигант 549		195,51	7,47	17,06	6,69	172,20	38,64	210,84
228	К2830		135,08	11,46	16,70	2,94	115,69	19,39	135,08
229			101,54	8,43	17,63	3,57	74,40	29,33	103,73
230									

Рисунок 3.7 – Приклад контексту файлу

Систему оцінки ваг можно оцінити за логікою работи парсеру Рік. Перенесення контексту між різними частинами файлу /2010/Звіт_за_минулі р. (соя,соняшник,рицина та інш). Роки позначені у звіті відповідають сівообігу предприємства, тобто різні. Наявність вланих типа UE0100538, КП11АхВК464Б. За работи «стандартих бібліотек» можна виділити або 2010 , або проміжок часу років у 20 за які створювалися гібриди.Треба ще раз зазначити, що норма для одного файлу може бути виключеням для іншого. Тому Рік проіндексовано наступним чином.

Співпадає- рейтинг +1 // не співпадає - виключено // не зрозуміло -рейтинг +0. Як результат надається список сортований за рейтингом готовий до ручної перевірки або подальшої обробки.

Завдяки такому підходу точніть методу P=0.95-1.00 від точності налаштуваня індексів за незначний час(превірка 3 списків до 2 сек).Недоліком є первинна індексація бази (15-17 Мб/хв).

Приклад функціонування коду в процесах структурування сховища та семантичного індексування даних представлено за допомогою наведеного нижче графічного інтерфейсу користувача та системних файлів.

Рисунок 3.8 - Головне вікно

Інтерфейс зроблений максимально простим, функціональним та інтуїтивно зрозумілим (рис.3.9) [12, 13].

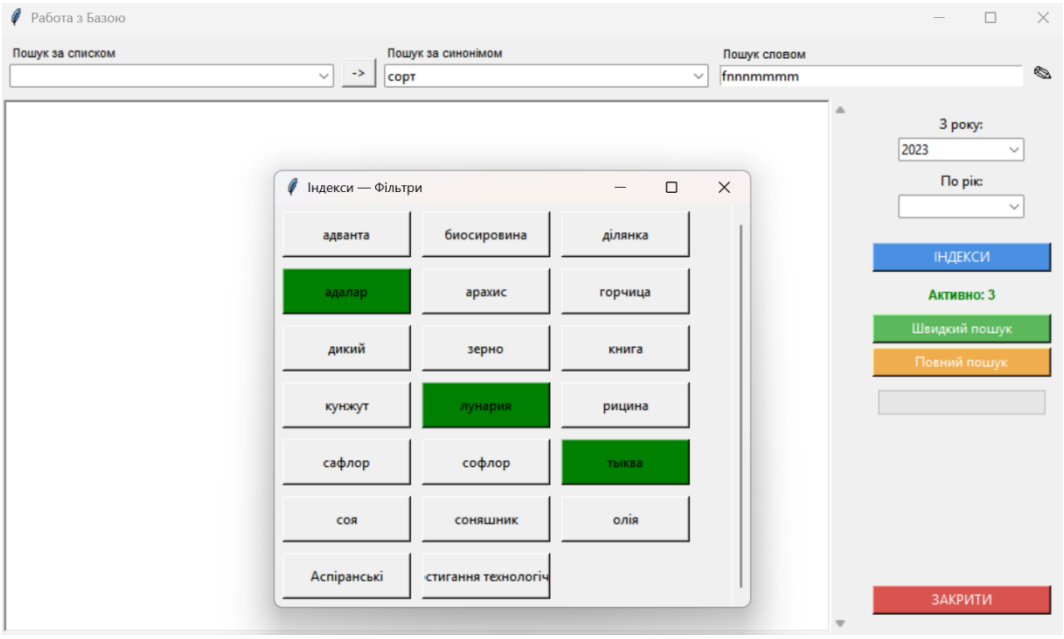


Рисунок 3.9 – Головне вікно роботи з базою

На прикладі рисунку 3.9 можно пояснити логіку работи фільтрів у пошуку.

На верхній панелі знаходяться стандартні інструменти пошуку.

Пошук за списком - автоматично сформований набір термінів з назв бази.

Пошук за синонимом - первиний тезаурус- терміни з синонімами до них.

Пошук за словом - стандартний пошук Windows.

Так пошук за двома першими категоріями шукає повний збіг з еталоном, а стандартний пошук залишений як допоміжнийзасіб.

Бічна панель має фільтри за категорією Рік та Індекси.

Детекція часових параметрів у системі стикається з проблемою періодичної відсутності явної атрибуції датування в іменах файлів, заголовках або великої кількості подібних індеків,що не дає 100% прив'язки до року.Тому Рік проіндексовано наступним чином.Співпадає рейтинг +1, не співпадає - виключено , не зрозуміло -рейтинг +0.

Треба зауважити,що панель Індекси може мати лише передвстановлені фільтри індексації бази та своєю активацією блокує Повний пошук.Кнопка Швидкий пошук запускає пошук по індексах бази, а Повний пошук- пошук по всіх назвах та повних текстах згідно установок.

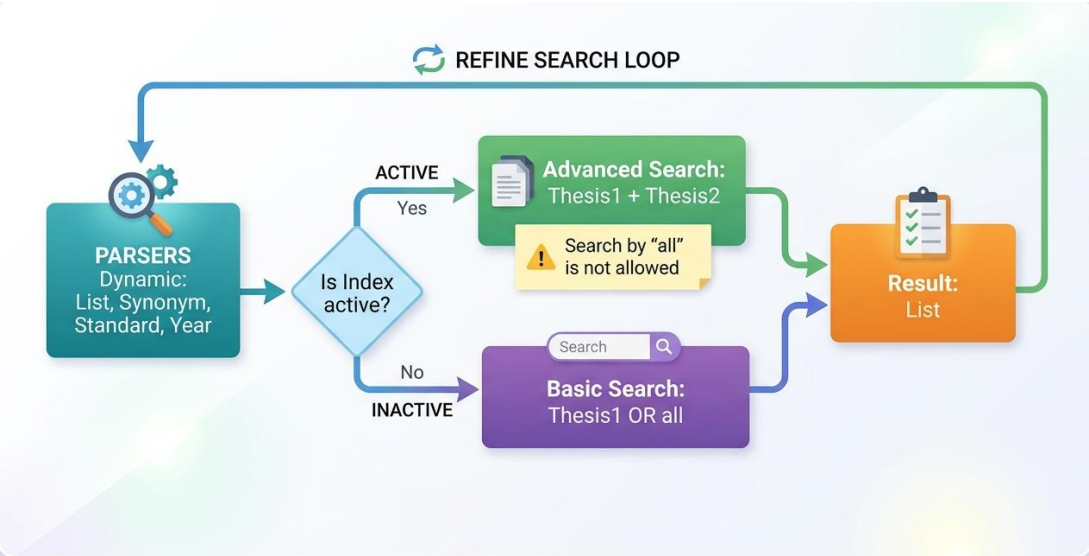


Рисунок 3.10 – Логічна схема роботи пошуку

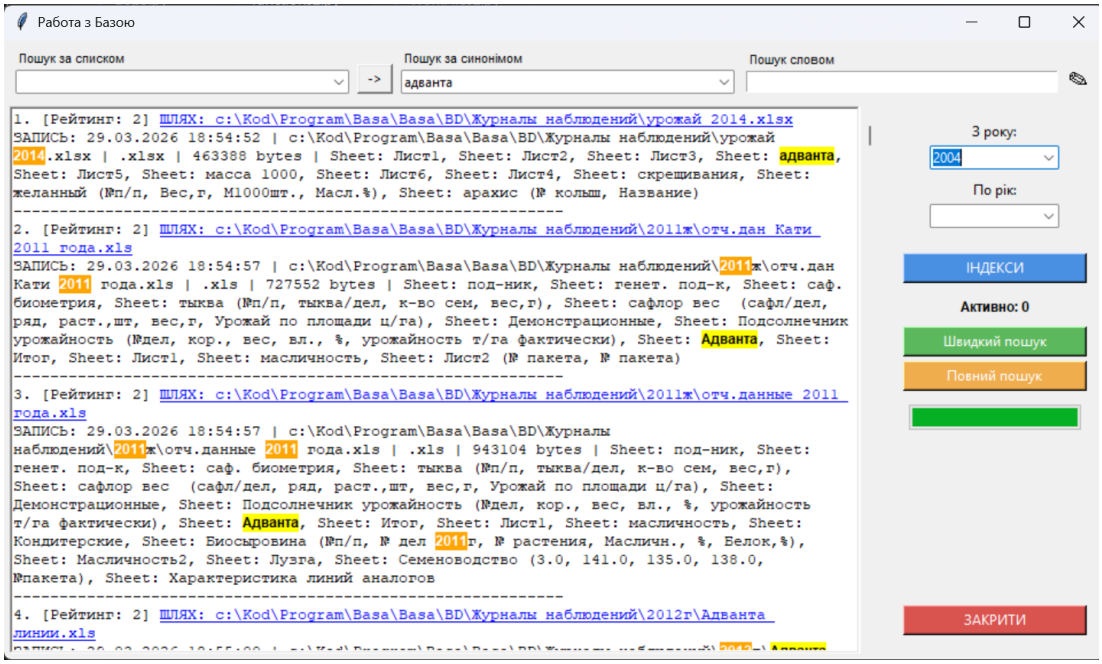


Рисунок 3.11 – Пошук у неіндексованій базі за первиним словником

На прикладі рис. 3.11 розглянемо функціонал пошуку за первиним тезаурусом.

Пошук за списком –Nan.(будь яке слово зі списку)

Пошук за синонімом- термін за первиним тезаурусом.

Пошук за словом –Nan (будь яке слово).

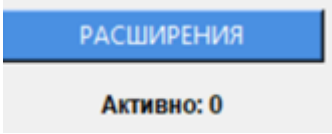
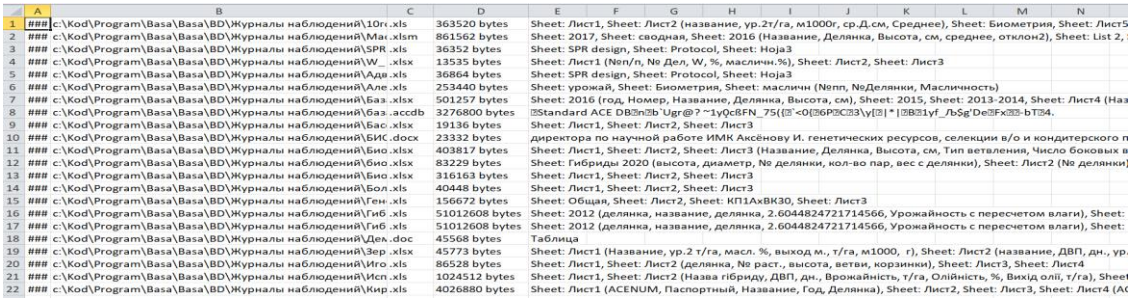


Рисунок 3.12 – Терміни за вторинним тезаурусом

Виконувався «Швидкий пошук». За цими налаштуваннями пошук проводився за первино індексованим файлом (Manifest.csv). Значення пошуку видані у вигляді рейтингу у порядку спадання з виділенням фільтрів пошуку.

Рисунок 3.13 - Пошук у неіндексованій базі за первиним словником

Посилання з міткою повного шляху файлу є інтерактивним та дозволяє швидко переглядати зміст файлу.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\10г.xls		363520 bytes	Sheet: Лист1, Sheet: Лист2 (название, ур.2т/га, м1000г, ср.Д.см, Среднее), Sheet: Биометрия, Sheet: Лист5									
2	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Мас.xlsm		861562 bytes	Sheet: 2017, Sheet: сводная, Sheet: 2016 (Название, Делянка, Высота, см, среднее, отклон2), Sheet: List 2,									
3	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\SPR .xls		36352 bytes	Sheet: SPR design, Sheet: Protocol, Sheet: Hoja3									
4	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\W_ .xlsx		13535 bytes	Sheet: Лист1 (№п/п, № Дел, W, %, масличн.%), Sheet: Лист2, Sheet: Лист3									
5	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Адв .xls		36864 bytes	Sheet: SPR design, Sheet: Protocol, Sheet: Hoja3									
6	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Але .xls		253440 bytes	Sheet: урожай, Sheet: Биометрия, Sheet: масличн (№пп, №Делянки, Масличность)									
7	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Баз .xlsx		501257 bytes	Sheet: 2016 (год, Номер, Название, Делянка, Высота, см), Sheet: 2015, Sheet: 2013-2014, Sheet: Лист4 (Наз									
8	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Баз .accdb		3276800 bytes	Standard ACE DB n b`Ugr@? ~1y0c8FN_75({ <0 6P C 3\y[* | B 1yf_ЉSg'De Fx -bT 4.									
9	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Бас .xlsx		19136 bytes	Sheet: Лист1, Sheet: Лист2, Sheet: Лист3									
10	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\БИС .docx		23332 bytes	директора по научной работе ИМК Аксёнову И. генетических ресурсов, селекции в/о и кондитерского п									
11	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Био .xlsx		403817 bytes	Sheet: Лист1, Sheet: Лист2, Sheet: Лист3 (Название, Делянка, Высота, см, Тип ветвления, Число боковых в									
12	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Био .xlsx		83229 bytes	Sheet: Гибриды 2020 (высота, диаметр, № делянки, кол-во пар, вес с делянки), Sheet: Лист2 (№ делянки									
13	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Био .xlsx		316163 bytes	Sheet: Лист1, Sheet: Лист2, Sheet: Лист3									
14	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Бол .xls		40448 bytes	Sheet: Лист1, Sheet: Лист2, Sheet: Лист3									
15	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Ген .xls		156672 bytes	Sheet: Общая, Sheet: Лист2, Sheet: КП1АхВК30, Sheet: Лист3									
16	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Гиб .xls		51012608 bytes	Sheet: 2012 (делянка, название, делянка, 2.6044824721714566, Урожайность с пересчетом влаги), Sheet:									
17	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Гиб .xls		51012608 bytes	Sheet: 2012 (делянка, название, делянка, 2.6044824721714566, Урожайность с пересчетом влаги), Sheet:									
18	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Дел .doc		45568 bytes	Таблица									
19	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Зер .xlsx		45773 bytes	Sheet: Лист1 (Название, ур.2 т/га, масл. %, выход м., т/га, м1000, г), Sheet: Лист2 (название, ДВП, дн., ур.									
20	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Иго .xls		86528 bytes	Sheet: Лист1, Sheet: Лист2 (делянка, № раст., высота, ветви, корзинки), Sheet: Лист3, Sheet: Лист4									
21	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Исп .xls		1024512 bytes	Sheet: Лист1, Sheet: Лист2 (Назва гібриду, ДВП, дн., Врожайність, т/га, Олійність, %, Вихід олії, т/га), Sheet									
22	###	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\Кир .xls		4026880 bytes	Sheet: Лист1 (ACENUM, Паспортный, Название, Год, Делянка), Sheet: Лист2, Sheet: Лист3, Sheet: Лист4 (AC									

Рисунок 3.14 – Первино індексований файл (Manifest.csv)

Зазначений файл застосовується для забезпечення внутрішнього функціонування бази даних, зокрема з метою проведення компаративного аналізу метаданих. Його використання дозволяє здійснювати верифікацію контенту, ідентифікувати структурні аномалії, системні помилки та дублікати файлів. Крім того, цей компонент виступає як базовий лінгвістичний ресурс та джерело первинного тезауруса (словника понять) для подальшої семантичної обробки інформації. В атоматичному режимі всі дані з Manifest.csv розкладаються на ликсеми, що є основою первиного тезаурусу.

На прикладі цього інтерфейсу можна докладніше розглянуту формування первинного тезаурусу

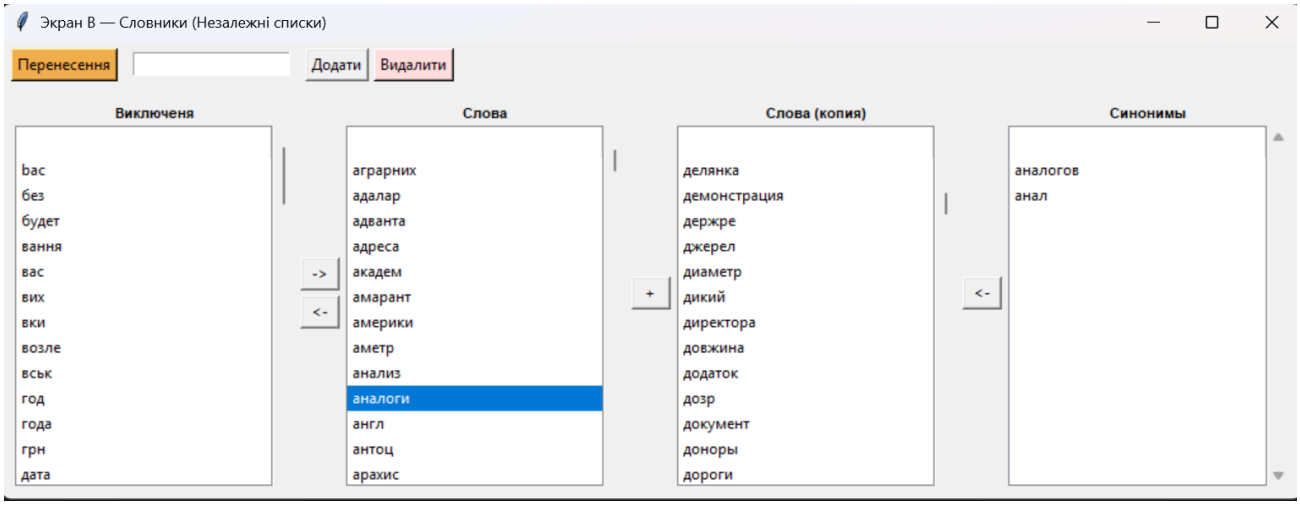


Рисунок 3.15 – Інтерфейс налаштування словників першого порядку

База та налаштування тезаурусів зібрані у файлі Scroll.xlsx з метою упорядкування та швидкого переносу налаштувань між версіями.

Так кнопка «Перенесення» додає всі лексеми у вже наявну базу лексем , паралельно проводячи порівняння на збіг.

Кнопка «Додати» дає змогу додати необхідні лексеми.

У межах цього тезаурусу була необхідність фіксувати лексеми у рамках жорсткої фіксації. Так поняття «біо» та «біосировина» є різними, бо є «біопаливо».Також автоматично усунені еквіваленти «рік», «дата» (є більш ефективні методи), лексеми з 2 букв («біо» та «ЯМР» значущі одиниці).

Надалі за допомогою інтерфейсу всі лексеми синтезуються у прості терміни за логічною ознакою.

Не важливі для обробки або шум- переходять у категорію Виключення, як от Sheet.

Всі наявні лексеми позиціонуються як Слова(або терміни). Надалі за допомогою простих операторів → // ← // + Слова можно премістити до Виключення або Синоніми . У подальшому термін складається зі Слова та його Синоніми.Поняття можно повернуті з Виключення або Синоніми.

Така необхідність виникла зі способу структуризації інформації.

Приклад темін «соняшник» має синоніми соняшнику- подсолнечник – сонях- подсолнух-подсол-соняш та інш, включаючи реєстри та правопис (як сонянях) . За такого підходу стандарті бібліотеки не можуть надати потрібної якості пошуку.

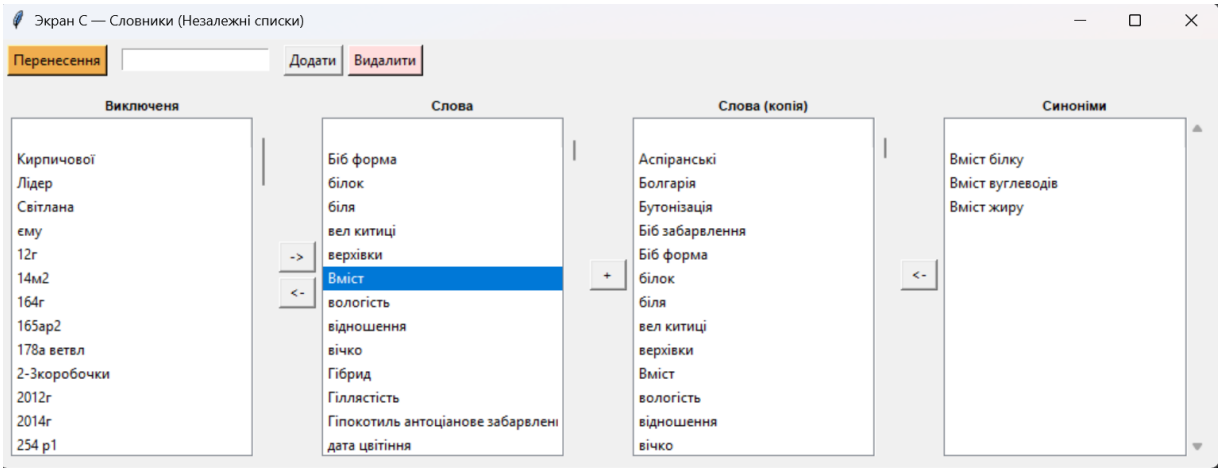


Рисунок 3.16 - Інтерфейс налаштування словників другуго порядку

Схожий функціонал та подібну логіку має інтерфейс для тезаурусів другого порядку.Основою лексем є зміст усіх файлів (перші 1000 одиниць). Оскількі зміт файлу несе типову інформацію у багатьох термінах тут використаний логічний синтез за напрямом діяльності.Приклад термін олія включає склад – давить – олеиновая- пальметиновая- вихід –олійність та інш.

Таке підвпорядкування дає змогу суттєво зменьшити обсяг термінів, що використовуються, та коротко индексувати зміст файла.

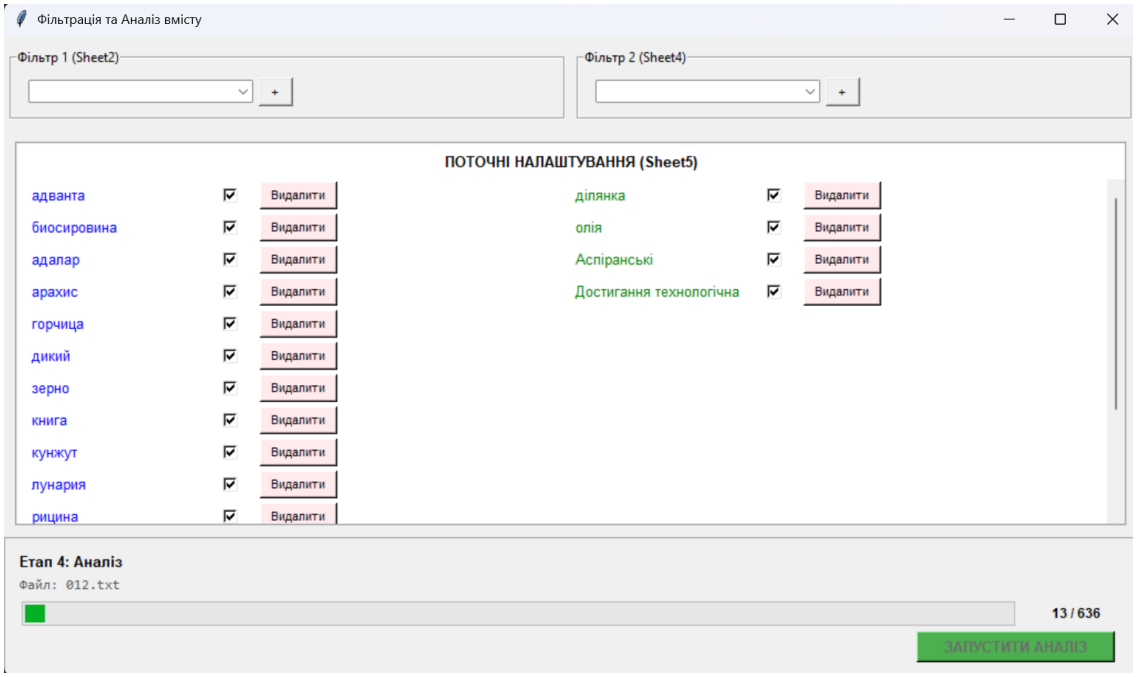


Рисунок 3.17 – Інтерфейс панелі індексування бази

При сформованих словниках термінів індекація бази дозволяє у подальшому значно скоротити термін пошуку,а час первинної індексації бази перевести у лінійну залежність.

Так панель з мал 6 дає змогу обрати терміни первинного та вторинного тезаурусів для індексації бази.На панелі вони логічно та інтуїтивно поділені (положення та колір).Функціонал термінів дозволяє налаштовувати потрібну матрицю індексування бази.

Зауваження Так термін «олія» зліва та «олія» зправа можуть бути різними термінами(олія-масло-алия чи олія-олійність-давить-пальметиновая) , тому для правої частини написання не має значення, а всі синоміни прирівняні до терміну значення.

2	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Троценко	ділянка
3	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Опыты по	соняшник, олія, ділянка
4	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2015\соняшник	соняшник, олія, ділянка
5	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Харьковс	ділянка
6	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2018\рижій.xlsx	олія
7	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2014ж\Книга ин	олія, ділянка
8	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Сафлор 2	соняшник, сафлор, адванта, олія, биосировина, ділянка
9	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2016\Копия ста	олія, ділянка
10	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Скрещив	ділянка
11	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\иммунит	соняшник
12	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2019\Копия Все	ділянка
13	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2012г\Кунжут в	кунжут
14	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2017\испр колл	ділянка
15	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2017\Копия ЯМ	ділянка
16	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2011ж\Кунжут в	кунжут
17	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2014\Книга ино	олія, ділянка
18	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2008г\Биометри	ділянка
19	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2011ж\урожайн	адванта, ділянка
20	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2015\Копия отб	олія
21	c:\Kod\Program\Basa\Basa\BD\Журналы наблюдений\2017\Копия Инс	соняшник, ділянка

Рисунок 3.18 – Індексована база даних

На прикладі частини індексованої бази даних можна пояснити логіку формування індексів та особливості работи коду.

№7 – Форма 2015 – не є індексом

Соя –рицина-сафлор-кунжжут-соняшник- розпізнано за 2 словником значення культура (відокремлено)

Олія - розпізнано за 2 словником значення дослідження на олійність

Так на цих двох прикладах можно наочно побачити , що попри всі спроби машині алгоритми обробки не надають 100% гарантії точності.Тому використання комбінованих методів дає найкращій результат.

Така конструкція дає змогу швидкого пошуку за замістом.Збереження результату дозволяє проводити подальший розширений пошук з будь якої точки графу пошуку.

ВИСНОВКИ

У дипломній роботі розв’язано актуальну науково-практичну задачу розробки методів та інтелектуальних технологій семантичного структурування нереляційних гетерогенних масивів даних у файлових сховищах, що дозволило ефективно подолати проблему «інформаційного хаосу» без необхідності превентивного ручного чищення чи переформатування інформаційного простору.

Проведено системний аналіз існуючих підходів до управління неструктурованими масивами даних та ідентифіковано чинники, що обмежують ефективність стандартних СУБД. Виявлено, що жорстка реляційна модель призводить до «парадоксу ефективності» при роботі з «брудними» даними, через що обґрунтовано доцільність переходу до гнучкої концепції «зворотного структурування» за принципом schema-on-read.

Проведено обґрунтованняо вибору алгоритмів інтелектуального аналізу для автоматизації процесів сегментації сховища. Доведено доцільність застосування каскадної схеми кластеризації, де ієрархічний алгоритм BIRCH виконує первинну ресурсоефективну побудову дерева ознак без перевантаження оперативної пам'яті ПК, а щільнісний метод DBSCAN забезпечує гнучке виділення змістовних кластерів довільної форми та ізоляцію інформаційного шуму в умовах високої вихідної ентропії.

Розроблено математичну модель екстракції контекстних ознак із вмісту файлів різних форматів, яка базується на оригінальному алгоритмі дворівневої морфофункціональної фільтрації. Створений лінгвістичний апарат дозволяє трансформувати хаотичний текстовий контент та складну галузеву синонімію у стійкий векторний простір латентних семантичних маркерів, а уніфікацію опису метаданих реалізовано на основі міжнародного стандарту Dublin Core.

Спроєктовано гнучку архітектуру системи інтелектуальної кластеризації, яка побудована на принципах Data-Driven architecture. Завдяки поділу на модулі попередньої обробки, семантичного аналізу та математичного моделювання, система володіє масштабованістю і здатна адаптуватися під нові предметні області шляхом зміни конфігураційних файлів словників без модифікації вихідного коду.

Програмно реалізовано автономний десктопний додаток на мові Python із використанням бібліотек Scikit-learn, Pandas та NumPy. Впровадження механізмів багатопотоковості (ThreadPoolExecutor) та формування наскрізного реєстру метаданих Manifest.csv дозволило створити швидкодійний інструмент з інтуїтивно зрозумілим інтерфейсом для кінцевого користувача.

Проведено експериментальне дослідження ефективності розробленої системи на реальному гетерогенному масиві Лабораторії генетичних досліджень Інституту олійних наук НААН України обсягом 350 Мб (683 файли за 2006–2024 рр.). Результати апробації підтвердили високу працездатність комплексу: індексування сховища відбувається з лінійною швидкістю обробки (15–17 МБ/хв), а час відгуку при виконанні інтелектуального пошуку не перевищує 3 секунд за умови досягнення точності семантичної видачі на рівні $P \ge 0.95$.

Практична значущість отриманих результатів полягає у можливості безпосереднього впровадження створеної системи підтримки прийняття рішень в операційну діяльність науково-дослідних установ та промислових підприємств для автоматизації рутинних процесів пошуку, дедуплікації та реструктуризації корпоративних знань.

ПЕРЕЛІК ПОСИЛАНЬ

1. Trad A., Kalpić D. Using Applied Mathematical Models for Business Transformation. 2021. 450 p.

2. Саяма Х. Вступ до моделювання та аналізу складних систем. URL: https://ukrayinska.libretexts.org/Математика/Наукові_обчислення_та_моделювання/Книга%3A_Вступ_до_моделювання_та_аналізу_складних_систем_(Sayama (дата звернення: 10.04.2026).

3. Бохан Лі, Сі Ян, Сунсон Дуань, Наннан Ван. На шляху до універсальної ꕥмантичної комунікації через передачу через зіставний семантичний підпростор. *IEEE Transactions on Image Processing*. 2026. Т. 35. С. 5003–5016. URL: https://ieeexplore.ieee.org/document/11515018 (дата звернення: 10.04.2026).

4. Аташі Саха, Наян Кумар Саркар, Санджой Пратіхар. PruferNET: розширена графова нейронна мережа для виявлення впливових вузлів у соціальних мережах. *IEEE Access*. 2026. Т. 14. С. 79940–79959. URL: https://ieeexplore.ieee.org/document/11533388 (дата звернення: 10.04.2026).

5.

6. Офіційна документація Scikit-Learn: модуль DBSCAN. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html (дата звернення: 25.05.2026).

7. Документація бібліотеки HDBSCAN для ефективної кластеризації. URL: https://hdbscan.readthedocs.io (дата звернення: 25.05.2026).

8. Практичне керівництво з щільнісної кластеризації текстового контенту. URL: https://realpython.com (дата звернення: 25.05.2026).

9. Математичні основи та архітектура векторних баз даних. URL: https://towardsdatascience.com (дата звернення: 25.05.2026).

10. Юньхао Лі, Імін Чен, Дінбан Лі, Венцзінь Ван, Інь Чжан. Самоохолодження: структура дивергенції-конвергенції для генерації багатопроменевих сигналів у магістратурі права. *ICASSP 2026 - 2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2026. С. 16537–16541. URL: https://ieeexplore.ieee.org/document/11464445 (дата звернення: 10.04.2026).

11. Щерба А. В. Програмування на Python. 2022. 250 с.

12. Офіційна документація Python. URL: https://docs.python.org (дата звернення: 10.04.2026).

13. TkDocs. URL: https://tkdocs.com (дата звернення: 10.04.2026).

14. Детальний гайд для новачків із візуальними прикладами. URL: https://realpython.com (дата звернення: 10.04.2026).

ДОДАТОК А

Використані ШІ-агенти

Gemeni 3.0 - перетворення звукового контенту у письмовий, аналогова корекція, кодинг Python

Gemeni NET- для пошуку літератури за описом задачі у інтернеті

ideogram.ai- створення зображень та художня обробка

jayflow.ai - створення схем за описом

Перелік файлів коду

dop	первиний словник код
dop2	вториний словник код
eb	індексатор базы
function	функціі для работи з файлами(доповненя бази)
function 2	головне вікно работи з базою
main	початкове вікно
plus	файл - зшивка
window	шаблон вікна
WorkFiles	работа з файлами та перенос у базу
WorkFiles2	словник 1 порядка вікно
WorkFiles3	словник 2 порядка вікно
WorkFiles4	вікно фільтров індексування

Код Python

Main.py код

```
import os
import sys
import csv
import shutil  # Додано для копіювання файлів у внутрішню папку
from tkinter import filedialog

# Імпортуємо модулі Pillow для роботи з будь-якими форматами зображень (JPG, JPEG, PNG тощо)
from PIL import Image, ImageTk
```

```
import WorkFiles
import window
import functions
import functions2
import plus
```

```
APP_DIR = os.path.dirname(os.path.abspath(__file__))
if APP_DIR not in sys.path:
    sys.path.insert(0, APP_DIR)
```

```
def start_app():
    p = WorkFiles.get_paths()
     dirs = [p["infiles"], p["basa"], p["add"], p["arch"], p["basa_inner"]]
    for d in dirs:
        if not os.path.exists(d):
            os.makedirs(d)

    if not os.path.exists(p["manifest"]):
                with open(p["manifest"], 'w', newline='', encoding='utf-8-sig') as f:
                csv.writer(f, delimiter=';').writerow(["Дата", "Путь", "Тип", "Размер", "Инфо"])
```

```
    class MainApp:
        def __init__(self):
            # Стандартное окно входа 1.0o
```

```
                self.root = window.StandardWindow("Головне Вікно",
sys.exit)

                # Шлях для збереження фонового малюнка всередині
структури БД
                self.saved_bg_path = os.path.join(p["basa_inner"],
"background.png")

                # --- СТВОРЕННЯ КОНТИНЕРУ ДЛЯ ФОНУ ЧЕРЕЗ TKINTER
---
                import tkinter as tk
                self.bg_label = tk.Label(self.root)
                    self.bg_label.place(x=0,  y=0,  relwidth=1,
relheight=1)
                self.bg_label.lower()  # Відправляємо на задній
план під кнопки
                self.bg_image = None
                # ----------------------------------------------
-----
```

```
                # --- АВТОМАТИЧНЕ ЗАВАНТАЖЕННЯ ЗБЕРЕЖЕНОГО ФОНУ
---
                if os.path.exists(self.saved_bg_path):
                    self.load_background_from_path(self.saved_b
g_path)
                # ----------------------------------------------
-----
```

```
                # Интеграция кнопок
                window.create_button(self.root, "Работа с Базою",
self.go_work)
                window.create_button(self.root, "Оновлення Бази",
self.go_update)

                # --- БІЛА КНОПКА ВИБОРУ ОБОЇВ НА МІСЦІ СИНЬОГО
ПРЯМОКУТНИКА ---
                self.btn_select_bg = tk.Button(
                    self.root,
                    text="Змінити фон",
                    command=self.choose_background,
```

```
                    bg="white",              # Кнопка біла
                     fg="black",              # Текст чорний для
чіткості
                    relief="raised"
                )
                 self.btn_select_bg.place(relx=0.82, rely=0.88,
anchor="nw", width=100, height=35)
                # ----------------------------------------------
---------------

                self.root.mainloop()
```

```
            def load_background_from_path(self, path):
                """Внутрішній метод для рендерингу та підгону
зображення під розміри вікна"""
                try:
                    window_width = self.root.winfo_width()
                    window_height = self.root.winfo_height()

                    if window_width <= 1: window_width = 630
                    if window_height <= 1: window_height = 420
```

```
                    img = Image.open(path)
                              img  =  img.resize((window_width,
window_height), Image.Resampling.LANCZOS)

                    self.bg_image = ImageTk.PhotoImage(img)
                    self.bg_label.config(image=self.bg_image)
                except Exception as e:
                    print(f"Помилка рендерингу фону: {e}")
```

```
            def choose_background(self):
                """Функція з повною підтримкою JPG, автопідгоном
та збереженням у basa_inner"""
                file_path = filedialog.askopenfilename(
                    title="Оберіть фоновий малюнок",
                       filetypes=[("Зображення (JPG/JPEG/PNG)",
"*.jpg *.jpeg *.png *.gif *.bmp")]
                )
```

```
                if file_path:
                    try:
                        # Крок 1. Конвертуємо та зберігаємо копію
малюнка у внутрішню папку
                         img_to_save = Image.open(file_path)
                        # Зберігаємо завжди як PNG для уникнення
конфліктів сумісності
                            img_to_save.save(self.saved_bg_path,
"PNG")

                        # Крок 2. Відображаємо збережений малюнок
на екрані
                         self.load_background_from_path(self.sav
ed_bg_path)

                    except Exception as e:
                      print(f"Помилка збереження та завантаження
фону: {e}")
```

```
            def go_work(self):
                """Запуск нового модуля 2.0о"""
                self.root.destroy()
                functions2.BaseWorkWindow()
```

```
            def go_update(self):
                """Запуск модуля 3.0о"""
                self.root.destroy()
                functions.UpdateBaseWindow()

        MainApp()
```

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

Plus.py код

```
    # plus.py
    # Список модулей для интеграции и работы системы

    # Основные модули интерфейса и логики
```

```
    module_window = "window"
    module_functions = "functions"
    module_work = "WorkFiles"
```

```
    # Дополнительные модули (версии 2.0)
    module_functions2 = "functions2"
    module_work2 = "WorkFiles2"
    module_work3 = "WorkFiles3"
    module_work4 = "WorkFiles4"
```

```
    module_dop = "dop"
    module_dop2 = "dop2"
    module_eb = "eb"
```

```
    # Список всех активных модулей
    active_modules = [
        module_window,
        module_functions,
        module_work,
        module_functions2,
        module_work2,
        module_work3,
        module_work4,
        module_dop,
        module_dop2,
        module_eb,

    ]
```

Window.py код

```
    import tkinter as tk

    class StandardWindow(tk.Tk):
        def __init__(self, title, on_close_callback=None):
            super().__init__()
            self.title(title)
                    sw,    sh    =    self.winfo_screenwidth(),
self.winfo_screenheight()
```

```
            w, h = sw // 2, sh // 2
            self.geometry(f"{w}x{h}+{(sw-w)//2}+{(sh-h)//2}")
            if on_close_callback:
                                self.protocol("WM_DELETE_WINDOW",
on_close_callback)
```

```
    def create_button(parent, text, command):
         btn  =  tk.Button(parent,  text=text,  command=command,
width=30, pady=5)
        btn.pack(pady=10)
        return btn
```

Function.py код

```
    import tkinter as tk
    from tkinter import filedialog, messagebox, scrolledtext
    import os
    import shutil
    import threading
    import WorkFiles
    from datetime import datetime

    def return_to_main(win):
        """Закрывает текущее окно и возвращает в Главное Меню"""
        win.destroy()
        import main
        main.start_app()
```

```
    ## 3.2о Поиск файлов
    class SearchFilesWindow:
        """Окно 3.2о Поиск и добавление файлов в infiles"""
        def __init__(self, parent):
            self.top = tk.Toplevel(parent)
            self.top.title("Пошук файлів")
            self.top.geometry("400x500")
            self.top.grab_set()

            self.lb = tk.Listbox(self.top, bg="#f0f0f0")
               self.lb.pack(fill="both",  expand=True,  padx=10,
pady=10)
```

```
            tk.Button(self.top, text="Обрати файли", width=20,
command=self.sel_f).pack(pady=5)
            tk.Button(self.top, text="Обрати папку", width=20,
command=self.sel_d).pack(pady=5)
                tk.Button(self.top,  text="Закрити",  width=20,
command=self.top.destroy).pack(pady=10)
```

```
        def copy_obj(self, src):
            p = WorkFiles.get_paths()
                            dst    =    os.path.join(p["infiles"],
os.path.basename(src))
            try:
                if os.path.isdir(src):
                   shutil.copytree(src, dst, dirs_exist_ok=True)
                else:
                    shutil.copy2(src, dst)
                WorkFiles.set_full_access(dst)
                                self.lb.insert("end",   f"Додано:
{os.path.basename(src)}")
                self.lb.see("end")
            except Exception as e:
                  messagebox.showerror("Помилка", f"Не вдалося
скопіювати {src}: {e}")
```

```
        def sel_f(self):
            files = filedialog.askopenfilenames()
            for f in files: self.copy_obj(f)
```

```
        def sel_d(self):
            folder = filedialog.askdirectory()
            if folder: self.copy_obj(folder)
```

```
    ##3.3о Обновить Базу
    class UpdateProcessWindow:
        """Окно 3.3о Процесс обновления и интеграции с цветным
логом"""
        def __init__(self, parent):
            self.top = tk.Toplevel(parent)
```

```
            self.top.title("Оновити Базу")
            self.top.geometry(f"{parent.winfo_width()}x{parent.
winfo_height()}")
            self.top.grab_set()
```

```
            # Поле вывода логов
                self.txt  =  scrolledtext.ScrolledText(self.top,
height=15, state='disabled', bg="#1e1e1e")
              self.txt.pack(fill="both",  expand=True,  padx=10,
pady=10)
```

```
            # Конфигурация цветов лога
                                    self.txt.tag_config("green",
foreground="#00ff00")  # Обычные файлы
                                     self.txt.tag_config("blue",
foreground="#00aaff")   # Архивы (zip, rar, 7z)
                                     self.txt.tag_config("red",
foreground="#ff4444")    # TMP и Ошибки
                                    self.txt.tag_config("white",
foreground="#ffffff")  # Системное (время)
```

```
            # Панель кнопок
            btn_frame = tk.Frame(self.top)
            btn_frame.pack(side="bottom", pady=10)

                tk.Button(btn_frame,  text="Оновити",  width=15,
command=self.run_upd).pack(side="left", padx=5)
             tk.Button(btn_frame, text="Інтегрувати", width=15,
command=self.run_int).pack(side="left", padx=5)
                tk.Button(btn_frame,  text="Закрити",  width=15,
command=self.top.destroy).pack(side="left", padx=5)
```

```
        def log(self, msg, color="white"):
            """Потокобезопасный цветной вывод в лог"""
            self.txt.configure(state='normal')
            timestamp = datetime.now().strftime('%H:%M:%S')

            self.txt.insert("end", f"[{timestamp}] ", "white")
            self.txt.insert("end", f"{msg}\n", color)
```

```
            self.txt.configure(state='disabled')
            self.txt.see("end")
```

```
        def run_upd(self):
            """Запуск сценария 3.3о1 в отдельном потоке"""
            threading.Thread(target=WorkFiles.process_update_3_
3_o1, args=(self.log,), daemon=True).start()
```

```
        def run_int(self):
            """Запуск сценария 3.3о2 в отдельном потоке"""
            threading.Thread(target=WorkFiles.process_integrate
_3_3_o2, args=(self.log,), daemon=True).start()
```

```
    ##3.4о Тест
    class TestDeleteWindow:
        def __init__(self, parent):
            self.top = tk.Toplevel(parent); self.top.title("3.4о
Тест")
            p = WorkFiles.get_paths()
                tk.Button(self.top,  text="Очистити  infiles",
command=lambda:
WorkFiles.clear_folder_safe(p["infiles"])).pack(pady=5)
                  tk.Button(self.top,  text="Очистити  Basa",
command=lambda:          WorkFiles.clear_folder_safe(p["basa"],
True)).pack(pady=5)
                         tk.Button(self.top,    text="Закрити",
command=self.top.destroy).pack(pady=10)
```

```
    ##3.0о Обновление Базы
    class UpdateBaseWindow:
        def __init__(self):
            import window
             self.win = window.StandardWindow("Оновлення Бази",
lambda: return_to_main(self.win))
            p = WorkFiles.get_paths()
                                                  d          =
datetime.fromtimestamp(os.path.getmtime(p["manifest"])).strftime
('%d.%m.%Y %H:%M:%S') if os.path.exists(p["manifest"]) else "---"
```

```
            tk.Label(self.win, text=f"Manifest (last edit): {d}",
fg="gray").pack(pady=10)
                window.create_button(self.win, "Пошук файлів",
lambda: SearchFilesWindow(self.win))
                window.create_button(self.win, "Оновить Базу",
lambda: UpdateProcessWindow(self.win))
                window.create_button(self.win, "Тест", lambda:
TestDeleteWindow(self.win))
             self.win.mainloop()
```

Function2.py код

```
    import tkinter as tk
    from tkinter import ttk, scrolledtext, messagebox
    import os
    import re
    import pandas as pd
    import datetime
    import time
    import WorkFiles2 as wf2

    BASA_PATH = r"C:\Kod\Program\Basa"
    MANIFEST_FILE = "Manifest.csv"
```

```
    # Предполагаем, что функция сортировки импортирована или
доступна
    from WorkFiles2 import ukr_rus_sort_key
```

```
    def return_to_main(win):
        win.destroy()
        import main
        main.start_app()
```

```
    class ExtensionsWindow:
        def __init__(self, parent_app):
            self.app = parent_app
            self.top = tk.Toplevel(self.app.win)
```

```
            self.top.title("Індекси – Фільтри")

            # Налаштування розміру вікна
                    sw,   sh   =   self.top.winfo_screenwidth(),
self.top.winfo_screenheight()
            self.top.geometry(f"{sw//3}x{sh//2}+{sw//3}+{sh//4}")

            # Створення Canvas та Scrollbar для скролінгу
            self.canvas = tk.Canvas(self.top)
                    self.scrollbar   =   tk.Scrollbar(self.top,
orient="vertical", command=self.canvas.yview)
            self.scrollable_frame = tk.Frame(self.canvas)

            self.scrollable_frame.bind(
                "<Configure>",
                                                    lambda      e:
self.canvas.configure(scrollregion=self.canvas.bbox("all"))
            )

                                self.canvas.create_window((0,     0),
window=self.scrollable_frame, anchor="nw")
            self.canvas.configure(yscrollcommand=self.scrollbar
.set)

                    self.canvas.pack(side="left",   fill="both",
expand=True)
            self.scrollbar.pack(side="right", fill="y")

            # Завантаження даних
            path = wf2.get_scroll_path()
            self.df5 = pd.DataFrame()
            if os.path.exists(path):
                try:
                    self.df5 = pd.read_excel(path, sheet_name=5)
                except Exception as e:
                    print(f"Помилка: {e}")

            self.buttons = []
```

```
            # Беремо дані з колонки B (індекс 1)
            data = self.df5.iloc[:, 1].dropna().tolist()

            if len(self.app.active_filters) != len(data):
                self.app.active_filters = [False] * len(data)
```

```
            # Генерація плитки 3 в ряд
            for i, text in enumerate(data):
                is_active = self.app.active_filters[i]

                 # Фіксований розмір кнопок: height=2 (висота), width=15 (ширина)
                        btn  =  tk.Button(self.scrollable_frame, text=str(text),
                                          bg="green" if is_active else "#f0f0f0",
                                      height=2, width=15,
                                              command=lambda  idx=i: self.toggle_filter(idx))

                 # Розміщення 3 в ряд: i // 3 - рядок, i % 3 - колонка
                btn.grid(row=i // 3, column=i % 3, sticky="nsew", padx=5, pady=5)
                 self.buttons.append(btn)
```

```
        def toggle_filter(self, idx):
                        self.app.active_filters[idx]   =   not self.app.active_filters[idx]
                       self.buttons[idx].config(bg="green"   if self.app.active_filters[idx] else "#f0f0f0")
            self.app.update_filter_label()
```

```
    class BaseWorkWindow:
        def __init__(self):
            self.win = tk.Tk()
            self.win.title("Работа з Базою")
```

```
            self.active_filters = [False] * 8

            self.current_year = datetime.datetime.now().year
                self.full_years = [str(year) for year in
range(self.current_year, self.current_year - 101, -1)]

                    sw, sh = self.win.winfo_screenwidth(),
self.win.winfo_screenheight()
            w, h = int(sw * 0.75), int(sh * 0.75)
            self.win.geometry(f"{w}x{h}+{(sw-w)//2}+{(sh-h)//2}")
```

```
            self.top_frame = tk.Frame(self.win, height=70)
              self.top_frame.pack(side="top", fill="x", padx=5,
pady=5)

            self.center_frame = tk.Frame(self.win)
               self.center_frame.pack(side="top", fill="both",
expand=True)

                self.right_panel = tk.Frame(self.center_frame,
width=180, bd=1, relief="flat")
            self.right_panel.pack(side="right", fill="y", padx=5,
pady=5)

                                            self.area_A =
scrolledtext.ScrolledText(self.center_frame, wrap=tk.WORD, bd=2,
relief="sunken")
                    self.area_A.pack(side="left", fill="both",
expand=True, padx=5, pady=5)
```

```
            self._draw_elements()
            self.win.mainloop()
```

```
        def update_filter_label(self):
            count = sum(self.active_filters)
            if hasattr(self, 'lbl_active'):
                self.lbl_active.config(text=f"Активно: {count}",
fg="green" if count > 0 else "black")
```

```
        def run_fast_search(self):
            """ЦЕЛЕВАЯ ПОДПРОГРАММА: Инициирует быстрый поиск."""
            settings = self.get_search_filters(mode="fast")

            self.progress['value'] = 20
            self.win.update_idletasks()
```

```
                                        new_results_list       =
self.process_search_logic(settings)

            self.progress['value'] = 70
            self.win.update_idletasks()
```

```
            self.render_to_area_A(new_results_list, settings)

            self.progress['value'] = 100
            self.win.update_idletasks()
```

```
        def process_search_logic(self, settings):
            """ЛОГИЧЕСКИЙ ДВИЖОК: Скорректированная логика работы
с годами."""
            output_results = []
            source_data = []

                full_manifest_path  =  os.path.join(BASA_PATH,
MANIFEST_FILE)
```

```
            if os.path.exists(full_manifest_path):
                try:
                            df_m  =  pd.read_csv(full_manifest_path,
sep=';', encoding='utf-8-sig')
                    source_data = df_m.to_dict('records')
                except Exception as e:
                    print(f"[ПОМИЛКА ЧИТАНЯ МАНИФЕСТА]: {e}")
```

```
            # Функция парсинга с дефолтными значениями (0 и 3000)
            def parse_y(val, default):
```

```
                    if val and str(val).lower() != 'nan' and
str(val).strip():
                        try: return int(float(val))
                        except: return default
                    return default
```

```
                year_start = parse_y(settings.get("year_start"), 0)
                year_end = parse_y(settings.get("year_end"), 3000)
```

```
                for idx, item in enumerate(source_data):
                    rating = 0
                    cols = list(item.values())
                       full_row_text = " ".join([str(v) for v in
cols]).lower()

                    # ЭТАП 1: Текстовые фильтры
                   f1 = str(settings.get("list", "")).strip().lower()
                    f2_list = settings.get("synonym", [])
                   f3 = str(settings.get("word", "")).strip().lower()
```

```
                                if f1 and f1 != 'nan' and
re.search(rf"\b{re.escape(f1)}\b", full_row_text): rating += 1
                    if isinstance(f2_list, list):
                        for syn in f2_list:
                            s_clean = str(syn).strip().lower()
                               if s_clean and s_clean != 'nan' and
re.search(rf"\b{re.escape(s_clean)}\b", full_row_text): rating +=
1
                    if f3 and f3 != 'nan' and f3 in full_row_text:
rating += 1

                    # ЭТАП 2: Логика фильтра ГОД
                    matched_year_val = None
                    should_discard = False

                   path_raw = str(cols[1]) if len(cols) > 1 else ""
                   page_raw = str(cols[4]) if len(cols) > 4 else ""
```

```
                                                    context_be      =
f"{os.path.splitext(path_raw)[0].replace('_', ' ').replace('-', '
').replace('\\', ' ')} {page_raw}"

                                                    matches      =
re.findall(r'(?<!\d)(\d{4}|\d{2})(?!\d)', context_be)
                found_years = {}
                for m in matches:
                    val_y = int(m)
                    if len(m) == 2: val_y += 1900 if val_y >= 70
else 2000
                    found_years[val_y] = m
```

```
                if found_years:
                     # Если годы найдены, проверяем попадание в
промежуток
                    in_range_found = False
                    for fy, original_str in found_years.items():
                        if year_start <= fy <= year_end:
                            in_range_found = True
                            matched_year_val = original_str
                            rating += 1 # Попадание в промежуток
= бонус +1 к рейтингу
                            break

                    if not in_range_found:
                        should_discard = True # Найден, но не в
диапазоне -> отсев

                # Если should_discard = True, отсеиваем
                if should_discard:
                    continue
```

```
                # ЭТАП 3: Добавление (Индексы + Результат)
                     if  'matched_index_val'  not  in  locals():
matched_index_val = ""

                if rating > 0:
                    output_results.append({
```

```
                        "full_record": " | ".join([str(v) for v
in cols]),
                         "path": str(cols[1]) if len(cols) > 1
else "---",
                        "rating": rating,
                        "main_color": "red" if rating >= 3 else
"blue" if rating == 2 else "green",
                        "highlight_year": matched_year_val,
                        "matched_index": matched_index_val
                    })
```

```
            output_results.sort(key=lambda  x:  x['rating'],
reverse=True)
            return output_results
```

```

```

```
        def render_to_area_A(self, new_data, settings):
            """ОБЪЕКТ ОКНА А: Отрисовка с выделением ключевых
слов, ГОДА и ИНДЕКСА."""
            if not hasattr(self, 'search_history'):
                self.search_history = []
```

```
            if len(self.search_history) >= 3:
                self.search_history.pop(0)
            self.search_history.append(new_data)
```

```
            self.area_A.delete('1.0', tk.END)
            current_list = self.search_history[-1]
```

```
            # Настройка тегов
                                self.area_A.tag_config("keyword",
background="yellow",   foreground="black",   font=("Arial",   9,
"bold"))
                                self.area_A.tag_config("year_hl",
background="orange",   foreground="white",   font=("Arial",   9,
"bold"))
```

```
                                self.area_A.tag_config("path_link",
foreground="blue", underline=True)
```

```
            if not current_list:
               self.area_A.insert(tk.END, "Збігів не знайдено.")
                return
```

```
            for i, res in enumerate(current_list, 1):
                # 1. Формирование заголовка с ИНДЕКСОМ
                idx_val = res.get('matched_index', '')
                idx_display = f" [Індекс: {idx_val}]" if idx_val
else ""

                                header    =    f"{i}.    [Рейтинг:
{res['rating']}]{idx_display} "
                self.area_A.insert(tk.END, header)

                # 2. Вставка пути
                                self.area_A.insert(tk.END,   f"ПУТЬ:
{res['path']}\n", "path_link")
```

```
                # 3. Вставка записи
                record_text = f"ЗАПИСЬ: {res['full_record']}\n"
                start_pos = self.area_A.index(tk.INSERT)
                self.area_A.insert(tk.END, record_text)
                end_pos = self.area_A.index(tk.INSERT)
```

```
                # 4. Подсветка года
                y_key = res.get("highlight_year")
                if y_key:
                    idx = start_pos
                    while True:
                                idx = self.area_A.search(y_key, idx,
stopindex=end_pos, nocase=True)
                        if not idx: break
                        end_hl = f"{idx}+{len(y_key)}c"
                            self.area_A.tag_add("year_hl", idx,
end_hl)
                        idx = end_hl
```

```
                # 5. Подсветка ключевых слов
                hl_keys = []
                for k in ["list", "word"]:
                    v = settings.get(k)
                        if v and str(v).lower() != 'nan':
hl_keys.append(str(v))
                for s in settings.get('synonym', []):
                        if s and str(s).lower() != 'nan':
hl_keys.append(str(s))
```

```
                for key in set(hl_keys):
                    if not key.strip(): continue
                    idx = start_pos
                    while True:
                            idx = self.area_A.search(key, idx,
stopindex=end_pos, nocase=True)
                        if not idx: break
                        end_hl = f"{idx}+{len(key)}c"
                            self.area_A.tag_add("keyword", idx,
end_hl)
                        idx = end_hl
```

```
                self.area_A.insert(tk.END, "-" * 60 + "\n")

            self.area_A.see("1.0")
```

```
        def run_deep_search(self):
              """Целевая подпрограмма для ГЛУБОКОГО поиска с
проверкой фильтров"""

              # Перевірка: чи активований хоча б один фільтр
індексів
            if sum(self.active_filters) > 0:
                messagebox.showwarning(
                    "Обмеження пошуку",
                    "Глибокий пошук неможливий, якщо активовані
фільтри індексів. "
```

```
                    "Будь ласка, деактивуйте всі фільтри перед
запуском глибокого пошуку."
                )
                return # Вихід з функції, пошук не запускається
```

```
            # Якщо фільтри не активовані, продовжуємо виконання
                                        search_settings       =
self.get_search_filters(mode="deep")
                self.area_A.insert(tk.END,  "\n[УВАГА]:  ГЛИБОКИЙ
пошук (повне сканування)...")

            try:
                stages = ["Ініціалізація модулів", "Синтаксичний
аналіз", "Глибоке порівняння контенту", "Фінальне складання"]
                for i, stage in enumerate(stages):
                    step_val = (i + 1) * 25
                    self.area_A.insert(tk.END, f"\n[Этап {i+1}]:
{stage}...")
                    self.progress['value'] = step_val
                    self.win.update_idletasks()
                    time.sleep(0.4)

                self.area_A.insert(tk.END, "\n[УСПІХ]: Глибокий
аналіз бази завершено.")
            except Exception as e:
                self.area_A.insert(tk.END, f"\n[Помилка]: {e}")
                self.progress['value'] = 0
```

```
        def get_search_filters(self, mode="fast"):
            """Сбор параметров и подготовка синтаксического
ядра"""
            selected_synonym = self.cb_2.get().strip()
                all_related_synonyms  =  [selected_synonym]  if
selected_synonym else []
```

```
            if selected_synonym:
                path = wf2.get_scroll_path()
                if os.path.exists(path):
                    try:
```

```
                        df2 = pd.read_excel(path, sheet_name=1)
                        if 'Слово' in df2.columns and 'Синоним'
in df2.columns:
                                                              mask   =
df2['Слово'].astype(str).str.strip() == selected_synonym
                                                          found_syns   =
df2[mask]['Синоним'].dropna().astype(str).tolist()
                                for s in found_syns:
                                    clean_s = s.strip()
                                     if clean_s and clean_s not in
all_related_synonyms:
                                        all_related_synonyms.append
(clean_s)
                    except Exception as e:
                        print(f"Помилка читання синонімів: {e}")
```

```
                  active_ext   =   [i   +   1   for   i,   v   in
enumerate(self.active_filters) if v]

            params = {
                    "mode":  "БЫСТРЫЙ"  if  mode  ==  "fast"  else
"ГЛУБОКИЙ",
                "list": self.cb_1.get(),
                "synonym": all_related_synonyms,
                "word": self.entry_top.get(),
                "year_start": self.cb_year_start.get(),
                "year_end": self.cb_year_end.get(),
                "filters": active_ext
            }
```

```
            self.area_A.delete('1.0', tk.END)
            report = (
                f"--- РЕЗУЛЬТАТЫ ({params['mode']} ПОИСК) ---\n"
                f"По списку: {params['list']}\n"
                f"По синониму: {', '.join(params['synonym']) if
params['synonym'] else ''}\n"
                f"По слову: {params['word']}\n"
                   f"Период:  {params['year_start']  or  '...'}  -
{params['year_end'] or '...'}\n"
```

```
                    f"Активные расширения: {params['filters'] if
params['filters'] else 'нет'}\n"
                f"{'-'*40}"
            )
            self.area_A.insert(tk.END, report)
            return params
```

```
        def _update_year_constraints(self, event=None):
            start_val = self.cb_year_start.get()
            end_val = self.cb_year_end.get()
            if start_val:
                self.cb_year_end['values'] = [""] + [y for y in
self.full_years if int(y) >= int(start_val)]
            else:
                        self.cb_year_end['values']  =  [""]  +
self.full_years
            if end_val:
                 self.cb_year_start['values'] = [""] + [y for y
in self.full_years if int(y) <= int(end_val)]
            else:
                       self.cb_year_start['values']  =  [""]  +
self.full_years
```

```
        def _draw_elements(self):
            path = wf2.get_scroll_path()
            vals_1, vals_2 = [""], [""]
```

```
            if os.path.exists(path):
                try:
                    df1 = pd.read_excel(path, sheet_name=0)
                                vals_1  +=  sorted(df1.iloc[:,
0].dropna().astype(str).unique().tolist(), key=ukr_rus_sort_key)
                    df2 = pd.read_excel(path, sheet_name=1)
                    if 'Слово' in df2.columns:
                                                vals_2   +=
sorted(df2['Слово'].dropna().astype(str).unique().tolist(),
key=ukr_rus_sort_key)
                except Exception as e:
                    print(f"Помилка читання Scroll.xlsx: {e}")
```

```
            f1 = tk.Frame(self.top_frame); f1.pack(side="left",
fill="x", expand=True, padx=5)
            tk.Label(f1, text="Пошук за списком", font=("Arial",
8)).pack(anchor="w")
                self.cb_1  =  ttk.Combobox(f1,  values=vals_1);
self.cb_1.pack(fill="x")
```

```
            tk.Button(self.top_frame, text="->", width=3,
                                                command=lambda:
wf2.ScreenBWindow(self.win)).pack(side="left",            padx=2,
pady=(15,0))
```

```
            f2 = tk.Frame(self.top_frame); f2.pack(side="left",
fill="x", expand=True, padx=5)
            tk.Label(f2, text="Пошук за синонімом", font=("Arial",
8)).pack(anchor="w")
                self.cb_2  =  ttk.Combobox(f2,  values=vals_2);
self.cb_2.pack(fill="x")
```

```
            f3 = tk.Frame(self.top_frame); f3.pack(side="left",
fill="x", expand=True, padx=5)
              tk.Label(f3,  text="Пошук  словом",  font=("Arial",
8)).pack(anchor="w")
                            self.entry_top    =    tk.Entry(f3);
self.entry_top.pack(fill="x")
```

```
             tk.Label(self.top_frame, text="✎", font=("Arial",
14)).pack(side="left", padx=(0, 5), pady=(15,0))
```

```
                          tk.Label(self.right_panel,    text="3
року:").pack(pady=(10, 0))
            self.cb_year_start = ttk.Combobox(self.right_panel,
values=[""] + self.full_years, width=15)
            self.cb_year_start.pack(pady=2)
                self.cb_year_start.bind("<<ComboboxSelected>>",
self._update_year_constraints)
```

```
                        tk.Label(self.right_panel,    text="По
рік:").pack(pady=(5, 0))
            self.cb_year_end  = ttk.Combobox(self.right_panel,
values=[""] + self.full_years, width=15)
            self.cb_year_end.pack(pady=2)
                  self.cb_year_end.bind("<<ComboboxSelected>>",
self._update_year_constraints)

                  tk.Button(self.right_panel,   text="ІНДЕКСИ",
bg="#4a90e2", fg="white",
                                                  command=lambda:
ExtensionsWindow(self)).pack(fill="x", padx=10, pady=(20, 5))

                self.lbl_active  =  tk.Label(self.right_panel,
text="Активно: 0", font=("Arial", 9, "bold"))
            self.lbl_active.pack(pady=5)
```

```
            tk.Button(self.right_panel, text="Швидкий пошук",
bg="#5cb85c", fg="white",
                  command=self.run_fast_search).pack(fill="x",
padx=10, pady=2)

            tk.Button(self.right_panel, text="Повний пошук",
bg="#f0ad4e", fg="white",
                  command=self.run_deep_search).pack(fill="x",
padx=10, pady=2)
```

```
            self.progress = ttk.Progressbar(self.right_panel,
orient="horizontal", length=150, mode="determinate")
            self.progress.pack(fill="x", padx=15, pady=10)

                  tk.Button(self.right_panel,   text="ЗАКРИТИ",
bg="#d9534f", fg="white",
                                                  command=lambda:
return_to_main(self.win)).pack(side="bottom", fill="x", padx=10,
pady=15)
```

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

WorkFiles.py код

```
import os
import shutil
import stat
import csv
import zipfile
import hashlib
import re
import warnings
import pandas as pd
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor



# Попытка импорта сторонних библиотек
try:
    import rarfile
except ImportError:
    rarfile = None
```

```
try:
    from openpyxl import load_workbook
    warnings.filterwarnings("ignore", category=UserWarning,
module="openpyxl.worksheet.header_footer")
except ImportError:
    load_workbook = None
```

```
try:
    import xlrd
except ImportError:
    xlrd = None
```

```
try:
```

```
        from docx import Document
    except ImportError:
        Document = None
```

```
    try:
        import ezodf
    except ImportError:
        ezodf = None
```

```
    # Группы расширений
    DB_EXT = ['.accdb', '.db', '.sqlite', '.mdb', '.sql']
    TXT_EXT = ['.txt', '.log', '.ini', '.json', '.xml', '.bat',
'.conf', '.cfg']
    DOC_EXT = ['.docx', '.odt', '.rtf', '.doc']
    TBL_EXT = ['.xlsx', '.xls', '.csv', '.ods', '.xlsm']
    IMG_EXT = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.mp4',
'.avi', '.mkv', '.mov']
    ARCHIVE_EXTENSIONS = ['.zip', '.rar', '.7z', '.tar', '.gz',
'.bz2']
```

```
    ########################################
```

```
    def get_file_hash(file_path):
        """Генерация MD5 хэша для полной сверки содержимого
файлов."""
        hasher = hashlib.md5()
        try:
            with open(file_path, 'rb') as f:
                for chunk in iter(lambda: f.read(4096), b""):
                    hasher.update(chunk)
            return hasher.hexdigest()
        except:
            return None
```

```
    def process_3_3_o2_5(sheet_name, sheet_data):
        """
        Поиск заголовков в таблицах.
```

```
        Добавлен расширенный сценарий для сложных ведомостей и
инвентарных списков.
        """
        detected_columns = []
        # Основные ключевые слова
          keywords  =  ["код",  "назва",  "№",  "рік",  "year",
"accename", "площа", "паспорт"]

        # Дополнительные маркеры для ведомостей (как на фото)
        extra_keywords = [
                "інвентарний",  "номер",  "одиниця",  "виміру",
"кількість",
                    "вартість",   "експлуатацію",   "об'єкт",
"характеристика"
        ]

        all_keys = keywords + extra_keywords
```

```
        try:
            # 1. Случай: openpyxl (xlsx)
            if hasattr(sheet_data, 'iter_rows'):
                 # Ищем глубже (до 40 строки), так как шапки
ведомостей бывают длинными
                  for  row  in  sheet_data.iter_rows(min_row=1,
max_row=40, values_only=True):
                    row_content_list = [str(c).strip() for c in
row if c is not None]
                                          row_string    =    "
".join(row_content_list).lower()

                    # Проверяем наличие хотя бы двух совпадений
из списка ключей для точности
                      matches = [k for k in all_keys if k in
row_string]
                    if len(matches) >= 2 or (len(matches) == 1
and any(k in row_string for k in keywords)):
                        detected_columns = row_content_list[:6]
# Берем чуть больше колонок для контекста
                        break
```

```
            # 2. Случай: xlrd (old xls)
            elif hasattr(sheet_data, 'row_values'):
                for i in range(min(40, sheet_data.nrows)):
                    row = sheet_data.row_values(i)
                    row_content_list = [str(c).strip() for c in
row if str(c).strip()]
                                                row_string    =    "
".join(row_content_list).lower()

                        matches = [k for k in all_keys if k in
row_string]
                    if len(matches) >= 2 or (len(matches) == 1
and any(k in row_string for k in keywords)):
                        detected_columns = row_content_list[:6]
                        break
```

```
            # 3. Случай: ezodf (ods)
            elif hasattr(sheet_data, 'rows'):
                for i, row in enumerate(sheet_data.rows()):
                    if i > 40: break
                    row_content_list = [str(c.value).strip() for
c in row if c.value is not None]
                                                row_string    =    "
".join(row_content_list).lower()

                        matches = [k for k in all_keys if k in
row_string]
                    if len(matches) >= 2 or (len(matches) == 1
and any(k in row_string for k in keywords)):
                        detected_columns = row_content_list[:6]
                        break

        except Exception:
            pass
```

```
        if detected_columns:
            # Очищаем колонки от технических символов и лишних
пробелов
            clean_cols = [c.replace('\n', ' ').strip() for c in
detected_columns if len(c) > 1]
```

```
                        return f"Sheet: {sheet_name} ({', '.join(clean_cols[:5])})"

            return f"Sheet: {sheet_name}"
```

```
    def get_text_preview(file_path, ext):
        """Извлекает превью текста. Логика очистки RTF и фильтрация таблиц."""
        content = ""
        try:
            if ext == '.rtf':
                with open(file_path, 'r', errors='ignore') as f:
                    raw_data = f.read(40000)
                clean_rtf = raw_data
                last_content = ""
                while last_content != clean_rtf:
                    last_content = clean_rtf
                    clean_rtf = re.sub(r'\{[^{}]*\}', ' ', clean_rtf)
                clean_rtf = re.sub(r'\\([a-z]{1,32}[0-9-]*|' + r"\'[0-9a-f]{2})", ' ', clean_rtf, flags=re.I)
                content = " ".join(clean_rtf.split())
            elif ext == '.docx':
                from docx import Document
                doc = Document(file_path)
                content = " ".join([p.text for p in doc.paragraphs if p.text.strip()])
            elif ext == '.odt':
                import ezodf
                doc = ezodf.opendoc(file_path)
                content = " ".join([p.plaintext() for p in doc.body if hasattr(p, 'plaintext')])
            elif ext in ['.doc', '.xls']:
                return "Таблица"
            else:
                with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
                    content = f.read(10000)
        except:
            return "Таблица"
```

```
        if not content.strip() or not re.search(r'[а-яА-ЯёЁa-zA-Z]', content):
            return "Таблица"

            # Очистка от мусора (длинные последовательности спецсимволов)
        content = re.sub(r'[_]{3,}', ' ', content)

        sentences = re.findall(r'([^.!?\n]{15,}[.!?])', content)
         clean_sentences = [s.strip() for s in sentences if s.count(';') < 3 and s.count('\\') < 2]

        result = ""
        if len(clean_sentences) >= 2:
            result = " ".join(clean_sentences[:2])
        elif len(clean_sentences) == 1:
            result = clean_sentences[0]
        else:
            # Если это не текст, а список параметров (ведомость)
            lines = [l.strip() for l in content.splitlines() if len(l.strip()) > 10]
             # Проверка на наличие "табличных" слов в первых строках
             if lines and any(w in lines[0].lower() for w in ["інв", "код", "об'єкт", "дата"]):
                return "Таблица"
           result = " | ".join(lines[:2]) if lines else "Таблица"

        if result == "Таблица": return "Таблица"

        result = result.replace(';', ' ')
        if result.startswith(('=', '+', '-', '@', ' ', '\t')):
            result = "'" + result

        return result.strip()[:400]
    ############################
```

```
    ###################################
    def find_unrar_tool():
        standard_paths = [r"C:\Program Files\WinRAR\UnRAR.exe",
r"C:\Program Files (x86)\WinRAR\UnRAR.exe"]
        for path in standard_paths:
            if os.path.exists(path): return path
        return shutil.which("unrar") or shutil.which("rar")
```

```
    if rarfile:
        exe = find_unrar_tool()
        if exe: rarfile.UNRAR_TOOL = exe
```

```
    def handle_remove_readonly(func, path, excinfo):
        os.chmod(path, stat.S_IWRITE)
        func(path)
```

```
    def set_full_access(path):
        try:
            if os.path.isfile(path):
                os.chmod(path, stat.S_IREAD | stat.S_IWRITE |
stat.S_IEXEC)
            else:
                for root, dirs, files in os.walk(path):
                    for d in dirs: os.chmod(os.path.join(root,
d), stat.S_IREAD | stat.S_IWRITE | stat.S_IEXEC)
                    for f in files: os.chmod(os.path.join(root,
f), stat.S_IREAD | stat.S_IWRITE | stat.S_IEXEC)
        except Exception: pass
```

```
    def get_paths():
        cur = os.path.dirname(os.path.abspath(__file__))
        basa = os.path.join(cur, "Basa")
        add_path = os.path.join(basa, "Add")
        return {
            "root": cur,
                "infiles":  os.path.join(os.path.dirname(cur),
"infiles"),
            "basa": basa,
            "basa_inner": os.path.join(basa, "Basa"),
```

```
            "add": add_path,
            "in_path": os.path.join(add_path, "in"),
            "out_path": os.path.join(add_path, "out"),
            "arch": os.path.join(add_path, "Архивы"),
            "manifest": os.path.join(basa, "Manifest.csv")
        }
```

```
    def clear_empty_folders(path):
        if not os.path.exists(path): return
        for root, dirs, files in os.walk(path, topdown=False):
            for d in dirs:
                dir_path = os.path.join(root, d)
               if os.path.basename(dir_path) in ["Архивы", "in",
"out"]: continue
                try:
                    if not os.listdir(dir_path):
                                os.chmod(dir_path,  stat.S_IWRITE);
os.rmdir(dir_path)
                except Exception: pass
```

```
    def process_update_3_3_o1(log_callback):
        p = get_paths()
        for d in [p["add"], p["arch"], p["in_path"]]:
            if not os.path.exists(d): os.makedirs(d)
        try:
            files = os.listdir(p["infiles"])
        except Exception as e:
            log_callback(f"Помилка доступу: {e}", "red")
            return
        for item in files:
            src = os.path.join(p["infiles"], item)
            ext = os.path.splitext(item)[1].lower()
            if ext == '.tmp':
                try: os.remove(src)
                except: pass
                continue
            try:
                if ext in ARCHIVE_EXTENSIONS:
                    dst = os.path.join(p["arch"], item)
                    shutil.move(src, dst)
```

```
                        log_callback(f"Архив -> Архиви: {item}",
"blue")
                else:
                    dst = os.path.join(p["in_path"], item)
                    if os.path.isdir(src):
                                        shutil.copytree(src,  dst,
dirs_exist_ok=True)
                                                shutil.rmtree(src,
onerror=handle_remove_readonly)
                    else:
                        shutil.move(src, dst)
                    log_callback(f"Переміщено в Add/in: {item}",
"green")
            except Exception as e:
                log_callback(f"Ошибка {item}: {e}", "red")
        log_callback(" завершено ", "white")
```

```
    ####
```

```
    def process_integrate_3_3_o2(log_callback):
        p = get_paths()
                 if     not     os.path.exists(p["out_path"]):
os.makedirs(p["out_path"])
```

```
        def stage_1_process_in():
            any_moved = False
            if not os.path.exists(p["in_path"]): return False
              for  root,  dirs,  files  in  os.walk(p["in_path"],
topdown=False):
                for f in files:
                    f_path = os.path.join(root, f)
                    ext = os.path.splitext(f)[1].lower()
                    if ext == '.tmp' or f.startswith("~$"):
                        try:
                                    os.chmod(f_path,  stat.S_IWRITE);
os.remove(f_path)
                                    log_callback(f"Видалено смітевий
файл: {f}", "red"); any_moved = True
                        except: pass
```

```
                        continue
                    if ext in ARCHIVE_EXTENSIONS:
                        try:
                             dst_arch = os.path.join(p["arch"],
f)
                            shutil.move(f_path, dst_arch)
                           log_callback(f"Архив -> Архиви: {f}",
"blue"); any_moved = True
                        except: pass
                        continue
                    try:
                            rel_path = os.path.relpath(f_path,
p["in_path"])
                          dst_out = os.path.join(p["out_path"],
rel_path)
                           os.makedirs(os.path.dirname(dst_out),
exist_ok=True)
                        shutil.move(f_path, dst_out)
                      log_callback(f"Файл -> Out: {f}", "green");
any_moved = True
                    except: pass
            return any_moved
```

```
        def stage_2_unpack_archives():
            any_unpacked = False
            if not os.path.exists(p["arch"]): return False
            for f in os.listdir(p["arch"]):
                f_path = os.path.join(p["arch"], f)
                ext = os.path.splitext(f)[1].lower()
                success = False
                try:
                    if ext == '.zip':
                        with zipfile.ZipFile(f_path, 'r') as z:
z.extractall(p["in_path"])
                       success = True; log_callback(f"Распакован
ZIP: {f}", "blue")
                        elif ext == '.rar' and rarfile and
rarfile.UNRAR_TOOL:
                            with rarfile.RarFile(f_path) as r:
r.extractall(p["in_path"])
```

```
                        success = True; log_callback(f"Распакован
RAR: {f}", "blue")
                    if success:
                        os.remove(f_path); any_unpacked = True
                except Exception as e: log_callback(f"Помилка
распаковки {f}: {e}", "red")
            return any_unpacked
```

```
        def stage_4_final_integration():
            if not os.path.exists(p["out_path"]): return

            # Сценарий: Загрузка Манифеста для сверки дубликатов
            existing_manifest_data = {}
            if os.path.exists(p["manifest"]):
                try:
                    with open(p["manifest"], 'r', encoding='utf-
8-sig') as m:
                        reader = csv.reader(m, delimiter=';')
                        for row in reader:
                            if len(row) >= 2:
                                # Ключ: путь к файлу (вторая
колонка)
                                existing_manifest_data[row[1]]
= row
                except: pass
```

```
            with ThreadPoolExecutor(max_workers=4) as executor:
                for root, dirs, files in os.walk(p["out_path"]):
                    for f in files:
                        if f.startswith("~$"): continue

                        src_file = os.path.join(root, f)
                        rel_path = os.path.relpath(src_file,
p["out_path"])
                        dst_file = os.path.join(p["basa_inner"],
rel_path)
                        ext = os.path.splitext(f)[1].lower()

                        # --- СЦЕНАРИЙ ПРОВЕРКИ ДУБЛИКАТОВ ---
                        if dst_file in existing_manifest_data:
```

```
                            if os.path.exists(dst_file):
                               # Полная сверка файлов (сравнение
MD5 хэшей)
                                  if get_file_hash(src_file) ==
get_file_hash(dst_file):
                                   log_callback(f"Копія (пропуск):
{f}", "blue")
                                      try: os.remove(src_file) #
Удаляем дубликат из Out
                                     except: pass
                                     continue
                        # ----------------------------------
```

```
                                                          now    =
datetime.now().strftime('%d.%m.%Y %H:%M:%S')
                        f_size = os.path.getsize(src_file)

                        try:
                         os.makedirs(os.path.dirname(dst_file),
exist_ok=True)
                           extra_info = ""
                           processed_list = []
```

```
                            # Обработка таблиц
                              if ext in ['.xlsx', '.xlsm'] and
load_workbook:
                                 wb = load_workbook(src_file,
data_only=True, read_only=True)
                                for name in wb.sheetnames:
                                    processed_list.append(proce
ss_3_3_o2_5(name, wb[name]))
                                wb.close()
                            elif ext == '.xls' and xlrd:
                              rb = xlrd.open_workbook(src_file,
on_demand=True)
                                for name in rb.sheet_names():
                                    processed_list.append(proce
ss_3_3_o2_5(name, rb.sheet_by_name(name)))
                                rb.release_resources()
                            elif ext == '.ods' and ezodf:
```

```
                                    ds = ezodf.opendoc(src_file)
                                    for s in ds.sheets:
                                        processed_list.append(proce
ss_3_3_o2_5(s.name, s))
                                elif ext == '.csv':
                                    df_head = None
                                    for enc in ['utf-8', 'cp1251']:
                                        try:
                                                                    df_head  =
pd.read_csv(src_file,    nrows=5,    sep=None,    engine='python',
encoding=enc)
                                                break
                                        except: continue

                                    if df_head is not None:
                                                                cols  =  ",
".join(list(df_head.columns)[:5])
                                        processed_list.append(f"CSV
({cols})")

                                if processed_list:
                                                        extra_info  =  ",
".join(processed_list)
                                else:
                                                            extra_info  =
get_text_preview(src_file, ext)
```

```
                               extra_info = extra_info.replace(';',
' ')
                                shutil.move(src_file, dst_file)
                                set_full_access(dst_file)

                                manifest_row = [now, dst_file, ext,
f"{f_size} bytes", extra_info]
                                    with  open(p["manifest"],  'a',
newline='', encoding='utf-8-sig') as m:
                                                                csv.writer(m,
delimiter=';').writerow(manifest_row)
                                    log_callback(f"Интегрирован:
{rel_path}", "green")
```

```
                        except Exception as e:
                            log_callback(f"Ошибка интеграции {f}:
{e}", "red")
```

```
        log_callback("Старт ...", "white")
        for i in range(10):
            moved = stage_1_process_in()
            unpacked = stage_2_unpack_archives()
            if not moved and not unpacked:
                clear_empty_folders(p["in_path"])
                break
        stage_4_final_integration()
        log_callback(" завершено", "white")
    ####
```

```
    ####
    def clear_folder_safe(path, is_basa=False):
        if not os.path.exists(path): return
        for item in os.listdir(path):
            ip = os.path.join(path, item)
            if is_basa and item == "Manifest.csv": continue
            try:
                    if  os.path.isdir(ip):  shutil.rmtree(ip,
onerror=handle_remove_readonly)
                else:
                    os.chmod(ip, stat.S_IWRITE); os.remove(ip)
            except: pass
```

WorkFiles2.py код

```
    import pandas as pd
    import os
    import tkinter as tk
    from tkinter import ttk

    def get_scroll_path():
        return r"C:\Kod\Program\Basa\Scroll.xlsx"
```

```
    def get_basa_kod_path():
        return r"C:\Kod\Program\Basa_Kod.xlsx"
```

```
    def ukr_rus_sort_key(s):
        """Кастомный ключ сортировки: Украинский -> Русский -> Другие"""
        s = str(s).lower()
        ukr_specific = "ґєії"
        if any(c in ukr_specific for c in s):
            return (0, s)
        if any('а' <= c <= 'я' for c in s):
            return (1, s)
        return (2, s)
```

```
    def check_or_create_sheet2(path):
        if not os.path.exists(path):
            with pd.ExcelWriter(path, engine='openpyxl') as writer:
                pd.DataFrame(columns=["Источник"]).to_excel(writer, sheet_name='Sheet1', index=False)
                pd.DataFrame(columns=["Слово", "Синоним", "Исключение"]).to_excel(writer, sheet_name='Sheet2', index=False)
            return
        xls = pd.ExcelFile(path)
        if 'Sheet2' not in xls.sheet_names:
            with pd.ExcelWriter(path, engine='openpyxl', mode='a') as writer:
                pd.DataFrame(columns=["Слово", "Синоним", "Исключение"]).to_excel(writer, sheet_name='Sheet2', index=False)
```

```
    class ListPanel(tk.Frame):
        def __init__(self, parent, title):
            super().__init__(parent)
            tk.Label(self, text=title, font=("Arial", 8, "bold")).pack(side="top")
            self.tree = ttk.Treeview(self, columns=("value",), show="headings", height=10)
            self.tree.heading("value", text="")
            self.tree.column("value", width=110)
```

```
            self.scroll = tk.Scrollbar(self, orient="vertical",
command=self.tree.yview)
            self.tree.configure(yscrollcommand=self.scroll.set)
            self.tree.pack(side="left", fill="both", expand=True)
            self.scroll.pack(side="right", fill="y")
```

```
    class ScreenBWindow:
        def __init__(self, parent_win):
            self.top = tk.Toplevel(parent_win)
               self.top.title("Экран  B  —  Словари  (Независимые
списки)")
                    sw,   sh   =   self.top.winfo_screenwidth(),
self.top.winfo_screenheight()
            self.top.geometry(f"{int(sw*0.8)}x{sh//2}+{sw//10}+
{sh//4}")

            check_or_create_sheet2(get_scroll_path())
```

```
            self.controls = tk.Frame(self.top, padx=5, pady=5)
            self.controls.pack(side="top", fill="x")

                     tk.Button(self.controls,   text="Перенос",
bg="#f0ad4e",
                       command=self.transfer_with_exceptions).pa
ck(side="left", padx=2)

            self.ent_word = tk.Entry(self.controls, width=20)
            self.ent_word.pack(side="left", padx=10)

                    tk.Button(self.controls,   text="Добавить",
command=self.add_to_main).pack(side="left", padx=2)
                     tk.Button(self.controls,   text="Удалить",
bg="#ffdddd",     command=self.delete_selected).pack(side="left",
padx=2)
```

```
            self.list_container = tk.Frame(self.top)
             self.list_container.pack(fill="both", expand=True,
padx=10, pady=10)
```

```
                    self.l_exc   =   ListPanel(self.list_container,
"Исключения")
                      self.l_exc.pack(side="left",   fill="both",
expand=True)

             self.frame_m1 = tk.Frame(self.list_container)
             self.frame_m1.pack(side="left", padx=5)
                  tk.Button(self.frame_m1,   text="->",   width=3,
command=self.move_to_words).pack(pady=2)
                  tk.Button(self.frame_m1,   text="<-",   width=3,
command=self.move_to_exceptions).pack(pady=2)
```

```
                   self.l_words   =   ListPanel(self.list_container,
"Слова")
                     self.l_words.pack(side="left",   fill="both",
expand=True)
                      self.l_words.tree.bind("<<TreeviewSelect>>",
self.on_word_selected)

             self.frame_m2 = tk.Frame(self.list_container)
             self.frame_m2.pack(side="left", padx=5)
                   tk.Button(self.frame_m2,   text="+",   width=3,
command=self.add_synonym).pack(pady=2)
```

```
             self.l_copy = ListPanel(self.list_container, "Слова
(копия)")
                      self.l_copy.pack(side="left",   fill="both",
expand=True)

             self.frame_m3 = tk.Frame(self.list_container)
             self.frame_m3.pack(side="left", padx=5)
                  tk.Button(self.frame_m3,   text="<-",   width=3,
command=self.remove_synonym).pack(pady=2)
```

```
                    self.l_syn   =   ListPanel(self.list_container,
"Синонимы")
                      self.l_syn.pack(side="left",   fill="both",
expand=True)
```

```
            self.load_initial_data()
```

```
        def on_word_selected(self, event):
            selected = self.l_words.tree.selection()
            if not selected: return
                                                word          =
self.l_words.tree.item(selected[0])['values'][0]
                for  item  in  self.l_syn.tree.get_children():
self.l_syn.tree.delete(item)
            try:
                        df  =  pd.read_excel(get_scroll_path(),
sheet_name='Sheet2')
                row = df[df['Слово'].astype(str) == str(word)]
                if not row.empty:
                    syn_val = row.iloc[0].get('Синоним', "")
                                if  not  pd.isna(syn_val)  and
str(syn_val).strip():
                                    syns  =  [s.strip()  for  s  in
str(syn_val).split(',') if s.strip()]
                        for s in syns: self.l_syn.tree.insert("",
tk.END, values=(s,))
            except: pass
```

```
             def   _sync_and_save(self,   df_words,   df_exc,
focus_word=None, target_panel=None):
            """Сохраняет слова и исключения как два независимых
блока данных в одной таблице"""
            path = get_scroll_path()

           # 1. Чистим и сортируем слова (вместе с их синонимами)
            df_words = df_words[df_words['Слово'].str.strip() !=
''].copy()
                                    df_words['temp_sort']       =
df_words['Слово'].apply(ukr_rus_sort_key)
                                               df_words         =
df_words.sort_values('temp_sort').drop('temp_sort', axis=1)

            # 2. Чистим и сортируем исключения
            df_exc = df_exc[df_exc['Исключение'].str.strip() !=
''].copy()
```

```
                                        df_exc['temp_sort']       =
df_exc['Исключение'].apply(ukr_rus_sort_key)
                                                df_exc          =
df_exc.sort_values('temp_sort').drop('temp_sort', axis=1)

            # 3. Объединяем их вертикально (независимые строки)
                  final_df   =   pd.concat([df_words,   df_exc],
ignore_index=True)
```

```
                with  pd.ExcelWriter(path,  engine='openpyxl',
mode='a', if_sheet_exists='replace') as writer:
                 final_df.to_excel(writer, sheet_name='Sheet2',
index=False)

            self.load_initial_data()
            if focus_word and target_panel:
                for item in target_panel.tree.get_children():
                                                            if
str(target_panel.tree.item(item)['values'][0]) == str(focus_word):
                        target_panel.tree.selection_set(item)
                        target_panel.tree.see(item)
                        break
```

```
        def get_split_df(self):
            """Разделяет общую таблицу на два независимых
DataFrame"""
                    df    =    pd.read_excel(get_scroll_path(),
sheet_name='Sheet2').fillna('')
            df_words = df[df['Слово'].str.strip() != ''].copy()
           df_exc = df[df['Исключение'].str.strip() != ''].copy()
            # Обнуляем перекрестные данные для чистоты
            df_words['Исключение'] = ''
            df_exc['Слово'] = ''
            df_exc['Синоним'] = ''
            return df_words, df_exc
```

```
        def move_to_words(self):
            selected = self.l_exc.tree.selection()
            if not selected: return
```

```
            word = self.l_exc.tree.item(selected[0])['values'][0]
            df_w, df_e = self.get_split_df()

            # Удаляем из исключений, добавляем в слова
                df_e  =  df_e[df_e['Исключение'].astype(str)  !=
str(word)]
            if word not in df_w['Слово'].astype(str).values:
                df_w = pd.concat([df_w, pd.DataFrame([{"Слово":
word, "Синоним": "", "Исключение": ""}])], ignore_index=True)

            self._sync_and_save(df_w, df_e, word, self.l_words)
```

```
        def move_to_exceptions(self):
            selected = self.l_words.tree.selection()
            if not selected: return
                                                word          =
self.l_words.tree.item(selected[0])['values'][0]
            df_w, df_e = self.get_split_df()

            # Удаляем из слов (вместе с синонимами!), добавляем
в исключения
            df_w = df_w[df_w['Слово'].astype(str) != str(word)]
           if word not in df_e['Исключение'].astype(str).values:
                df_e = pd.concat([df_e, pd.DataFrame([{"Слово":
"", "Синоним": "", "Исключение": word}])], ignore_index=True)

            self._sync_and_save(df_w, df_e, word, self.l_exc)
```

```
        def add_synonym(self):
            sel_word = self.l_words.tree.selection()
            sel_copy = self.l_copy.tree.selection()
            if not sel_word or not sel_copy: return
                                               w_main         =
str(self.l_words.tree.item(sel_word[0])['values'][0]).strip()
                                                w_syn         =
str(self.l_copy.tree.item(sel_copy[0])['values'][0]).strip()
            if w_main == w_syn: return
```

```
            df_w, df_e = self.get_split_df()
```

```
            idx_m = df_w.index[df_w['Слово'] == w_main].tolist()
            idx_s = df_w.index[df_w['Слово'] == w_syn].tolist()

            if idx_m and idx_s:
                # Склеиваем синонимы
               syns_m = [s.strip() for s in str(df_w.at[idx_m[0],
'Синоним']).split(',') if s.strip()]
               syns_s = [s.strip() for s in str(df_w.at[idx_s[0],
'Синоним']).split(',') if s.strip()]

                seen = set([w_main.lower()])
                final = []
                for s in (syns_m + [w_syn] + syns_s):
                    if s.strip().lower() not in seen:
                        seen.add(s.strip().lower())
                        final.append(s.strip())

                df_w.at[idx_m[0], 'Синоним'] = ", ".join(final)
                df_w = df_w.drop(idx_s[0])
                      self._sync_and_save(df_w,  df_e,  w_main,
self.l_words)
```

```
        def remove_synonym(self):
            sel_syn = self.l_syn.tree.selection()
            sel_word = self.l_words.tree.selection()
            if not (sel_syn and sel_word): return
                                                s_val          =
str(self.l_syn.tree.item(sel_syn[0])['values'][0]).strip()
                                                m_val          =
str(self.l_words.tree.item(sel_word[0])['values'][0]).strip()
```

```
            df_w, df_e = self.get_split_df()
            idx = df_w.index[df_w['Слово'] == m_val].tolist()
            if idx:
                 cur = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                if s_val in cur: cur.remove(s_val)
                df_w.at[idx[0], 'Синоним'] = ", ".join(cur)
                if s_val not in df_w['Слово'].values:
```

```
                                        df_w   =   pd.concat([df_w,
pd.DataFrame([{"Слово": s_val, "Синоним": "", "Исключение": ""}])],
ignore_index=True)
                        self._sync_and_save(df_w,  df_e,  s_val,
self.l_words)
```

```
        def transfer_with_exceptions(self):
            path = get_scroll_path()
            try:
                df1 = pd.read_excel(path, sheet_name=0)
                df_w, df_e = self.get_split_df()

               ban = set(df_e["Исключение"].astype(str).unique())
| set(df_w["Слово"].astype(str).unique())
                for s_str in df_w["Синоним"].astype(str):
                   for p in s_str.split(','): ban.add(p.strip())
```

```
                                    source    =    df1.iloc[:,
0].dropna().astype(str).unique()
                new = [w for w in source if w not in ban]
                if new:
                                    df_w   =   pd.concat([df_w,
pd.DataFrame([{"Слово": w, "Синоним": "", "Исключение": ""} for w
in new])], ignore_index=True)
                self._sync_and_save(df_w, df_e)
            except: pass
```

```
        def add_to_main(self):
            val = self.ent_word.get().strip()
            if not val: return
            df_w, df_e = self.get_split_df()
            if val not in df_w['Слово'].values:
                df_w = pd.concat([df_w, pd.DataFrame([{"Слово":
val, "Синоним": "", "Исключение": ""}])], ignore_index=True)
               self._sync_and_save(df_w, df_e, val, self.l_words)
                self.ent_word.delete(0, tk.END)
```

```
        def delete_selected(self):
```

```
            s_exc, s_word, s_syn = self.l_exc.tree.selection(),
self.l_words.tree.selection(), self.l_syn.tree.selection()
            df_w, df_e = self.get_split_df()

            if s_exc:
                                                    v       =
str(self.l_exc.tree.item(s_exc[0])['values'][0])
                df_e = df_e[df_e['Исключение'].astype(str) != v]
            elif s_syn and s_word:
                                                v_s,     v_m     =
str(self.l_syn.tree.item(s_syn[0])['values'][0]).strip(),
str(self.l_words.tree.item(s_word[0])['values'][0]).strip()
                idx = df_w.index[df_w['Слово'] == v_m].tolist()
                if idx:
                   lst = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                    if v_s in lst: lst.remove(v_s)
                    df_w.at[idx[0], 'Синоним'] = ", ".join(lst)
            elif s_word:
                                                    v       =
str(self.l_words.tree.item(s_word[0])['values'][0])
                idx = df_w.index[df_w['Слово'] == v].tolist()
                if idx:
                  syns = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                    df_w = df_w.drop(idx[0])
                    for s in syns:
                        if s and s not in df_w['Слово'].values:
                                        df_w  =  pd.concat([df_w,
pd.DataFrame([{"Слово": s, "Синоним": "", "Исключение": ""}])],
ignore_index=True)
            self._sync_and_save(df_w, df_e)
```

```
        def load_initial_data(self):
            try:
                df_w, df_e = self.get_split_df()
                for p in [self.l_exc, self.l_words, self.l_copy,
self.l_syn]:
                            for  item  in  p.tree.get_children():
p.tree.delete(item)
```

```
                    for e in sorted(df_e['Исключение'].tolist(),
key=ukr_rus_sort_key):
                    if str(e).strip(): self.l_exc.tree.insert("",
tk.END, values=(e,))

                        for w in df_w['Слово'].tolist(): # Уже
отсортировано в _sync_and_save
                            self.l_words.tree.insert("", tk.END,
values=(w,))
                             self.l_copy.tree.insert("", tk.END,
values=(w,))
                except: pass
```

```
    if __name__ == "__main__":
        root = tk.Tk(); root.withdraw()
        app = ScreenBWindow(root)
        root.mainloop()
```

WorkFiles3.py код

```
    import pandas as pd
    import os
    import tkinter as tk
    from tkinter import ttk

    def get_scroll_path():
        return r"C:\Kod\Program\Basa\Scroll.xlsx"
```

```
    def ukr_rus_sort_key(s):
        """Кастомный ключ сортировки: Украинский -> Русский ->
Другие"""
        s = str(s).lower()
        ukr_specific = "ґєії"
        if any(c in ukr_specific for c in s):
            return (0, s)
        if any('а' <= c <= 'я' for c in s):
            return (1, s)
        return (2, s)
```

```
    def check_or_create_sheet2(path):
        """Создает файл или недостающие листы Sheet3 и Sheet4"""
        if not os.path.exists(path):
            with pd.ExcelWriter(path, engine='openpyxl') as
writer:
                pd.DataFrame(columns=["Источник"]).to_excel(wri
ter, sheet_name='Sheet3', index=False)
                pd.DataFrame(columns=["Слово", "Синоним",
"Исключение"]).to_excel(writer, sheet_name='Sheet4', index=False)
            return

        xls = pd.ExcelFile(path)
        sheets = xls.sheet_names

        # Режим добавления, если листы отсутствуют
        if 'Sheet3' not in sheets or 'Sheet4' not in sheets:
            with pd.ExcelWriter(path, engine='openpyxl',
mode='a', if_sheet_exists='overlay') as writer:
                if 'Sheet3' not in sheets:
                    pd.DataFrame(columns=["Источник"]).to_excel
(writer, sheet_name='Sheet3', index=False)
                if 'Sheet4' not in sheets:
                    pd.DataFrame(columns=["Слово", "Синоним",
"Исключение"]).to_excel(writer, sheet_name='Sheet4', index=False)
```

```
    class ListPanel(tk.Frame):
        def __init__(self, parent, title):
            super().__init__(parent)
            tk.Label(self, text=title, font=("Arial", 8,
"bold")).pack(side="top")
            self.tree = ttk.Treeview(self, columns=("value",),
show="headings", height=10)
            self.tree.heading("value", text="")
            self.tree.column("value", width=110)
            self.scroll = tk.Scrollbar(self, orient="vertical",
command=self.tree.yview)
            self.tree.configure(yscrollcommand=self.scroll.set)
            self.tree.pack(side="left", fill="both", expand=True)
            self.scroll.pack(side="right", fill="y")
```

```
    class ScreenBWindow:
        def __init__(self, parent_win):
            self.top = tk.Toplevel(parent_win)
               self.top.title("Экран  B  —  Словари  (Независимые
списки)")
                    sw,   sh   =   self.top.winfo_screenwidth(),
self.top.winfo_screenheight()
             self.top.geometry(f"{int(sw*0.8)}x{sh//2}+{sw//10}+
{sh//4}")

             check_or_create_sheet2(get_scroll_path())
```

```
             self.controls = tk.Frame(self.top, padx=5, pady=5)
             self.controls.pack(side="top", fill="x")

                      tk.Button(self.controls,   text="Перенос",
bg="#f0ad4e",
                        command=self.transfer_with_exceptions).pa
ck(side="left", padx=2)

             self.ent_word = tk.Entry(self.controls, width=20)
             self.ent_word.pack(side="left", padx=10)

                     tk.Button(self.controls,   text="Добавить",
command=self.add_to_main).pack(side="left", padx=2)
                      tk.Button(self.controls,   text="Удалить",
bg="#ffdddd",     command=self.delete_selected).pack(side="left",
padx=2)
```

```
             self.list_container = tk.Frame(self.top)
              self.list_container.pack(fill="both", expand=True,
padx=10, pady=10)
```

```
                  self.l_exc   =   ListPanel(self.list_container,
"Исключения")
                     self.l_exc.pack(side="left",   fill="both",
expand=True)

             self.frame_m1 = tk.Frame(self.list_container)
```

```
            self.frame_m1.pack(side="left", padx=5)
                tk.Button(self.frame_m1,  text="->",  width=3,
command=self.move_to_words).pack(pady=2)
                tk.Button(self.frame_m1,  text="<-",  width=3,
command=self.move_to_exceptions).pack(pady=2)
```

```
                self.l_words  =  ListPanel(self.list_container,
"Слова")
                  self.l_words.pack(side="left",  fill="both",
expand=True)
                   self.l_words.tree.bind("<<TreeviewSelect>>",
self.on_word_selected)

            self.frame_m2 = tk.Frame(self.list_container)
            self.frame_m2.pack(side="left", padx=5)
                 tk.Button(self.frame_m2,  text="+",  width=3,
command=self.add_synonym).pack(pady=2)
```

```
            self.l_copy = ListPanel(self.list_container, "Слова
(копия)")
                   self.l_copy.pack(side="left",  fill="both",
expand=True)

            self.frame_m3 = tk.Frame(self.list_container)
            self.frame_m3.pack(side="left", padx=5)
                tk.Button(self.frame_m3,  text="<-",  width=3,
command=self.remove_synonym).pack(pady=2)
```

```
                 self.l_syn  =  ListPanel(self.list_container,
"Синонимы")
                   self.l_syn.pack(side="left",  fill="both",
expand=True)
```

```
            self.load_initial_data()
```

```
        def on_word_selected(self, event):
            selected = self.l_words.tree.selection()
            if not selected: return
```

```
                                                word          =
self.l_words.tree.item(selected[0])['values'][0]
                    for  item  in  self.l_syn.tree.get_children():
self.l_syn.tree.delete(item)
                try:
                    # Читаем из Sheet4
                            df   =   pd.read_excel(get_scroll_path(),
sheet_name='Sheet4')
                    row = df[df['Слово'].astype(str) == str(word)]
                    if not row.empty:
                        syn_val = row.iloc[0].get('Синоним', "")
                                    if   not   pd.isna(syn_val)   and
str(syn_val).strip():
                                        syns  =  [s.strip()  for  s  in
str(syn_val).split(',') if s.strip()]
                                for s in syns: self.l_syn.tree.insert("",
tk.END, values=(s,))
                except: pass
```

```
                def    _sync_and_save(self,    df_words,    df_exc,
focus_word=None, target_panel=None):
                """Сохраняет слова и исключения как два независимых
блока данных в Sheet4"""
                path = get_scroll_path()

                df_words = df_words[df_words['Слово'].str.strip() !=
''].copy()
                                        df_words['temp_sort']       =
df_words['Слово'].apply(ukr_rus_sort_key)
                                                    df_words        =
df_words.sort_values('temp_sort').drop('temp_sort', axis=1)

                df_exc = df_exc[df_exc['Исключение'].str.strip() !=
''].copy()
                                        df_exc['temp_sort']         =
df_exc['Исключение'].apply(ukr_rus_sort_key)
                                                    df_exc          =
df_exc.sort_values('temp_sort').drop('temp_sort', axis=1)
```

```
                    final_df   =   pd.concat([df_words,   df_exc],
ignore_index=True)
```

```
                # Перезаписываем Sheet4
                    with  pd.ExcelWriter(path,  engine='openpyxl',
mode='a', if_sheet_exists='replace') as writer:
                    final_df.to_excel(writer, sheet_name='Sheet4',
index=False)

                self.load_initial_data()
                if focus_word and target_panel:
                    for item in target_panel.tree.get_children():
                                                                    if
str(target_panel.tree.item(item)['values'][0]) == str(focus_word):
                            target_panel.tree.selection_set(item)
                            target_panel.tree.see(item)
                            break
```

```
            def get_split_df(self):
                  """Разделяет  таблицу  Sheet4  на  два  независимых
DataFrame"""
                        df    =    pd.read_excel(get_scroll_path(),
sheet_name='Sheet4').fillna('')
                df_words = df[df['Слово'].str.strip() != ''].copy()
               df_exc = df[df['Исключение'].str.strip() != ''].copy()
                df_words['Исключение'] = ''
                df_exc['Слово'] = ''
                df_exc['Синоним'] = ''
                return df_words, df_exc
```

```
            def move_to_words(self):
                selected = self.l_exc.tree.selection()
                if not selected: return
               word = self.l_exc.tree.item(selected[0])['values'][0]
                df_w, df_e = self.get_split_df()

                    df_e  =  df_e[df_e['Исключение'].astype(str)  !=
str(word)]
                if word not in df_w['Слово'].astype(str).values:
```

```
                df_w = pd.concat([df_w, pd.DataFrame([{"Слово":
word, "Синоним": "", "Исключение": ""}])], ignore_index=True)

            self._sync_and_save(df_w, df_e, word, self.l_words)
```

```
        def move_to_exceptions(self):
            selected = self.l_words.tree.selection()
            if not selected: return
                                                word          =
self.l_words.tree.item(selected[0])['values'][0]
            df_w, df_e = self.get_split_df()

            df_w = df_w[df_w['Слово'].astype(str) != str(word)]
           if word not in df_e['Исключение'].astype(str).values:
                df_e = pd.concat([df_e, pd.DataFrame([{"Слово":
"", "Синоним": "", "Исключение": word}])], ignore_index=True)

            self._sync_and_save(df_w, df_e, word, self.l_exc)
```

```
        def add_synonym(self):
            sel_word = self.l_words.tree.selection()
            sel_copy = self.l_copy.tree.selection()
            if not sel_word or not sel_copy: return
                                                w_main        =
str(self.l_words.tree.item(sel_word[0])['values'][0]).strip()
                                                w_syn         =
str(self.l_copy.tree.item(sel_copy[0])['values'][0]).strip()
            if w_main == w_syn: return
```

```
            df_w, df_e = self.get_split_df()
           idx_m = df_w.index[df_w['Слово'] == w_main].tolist()
            idx_s = df_w.index[df_w['Слово'] == w_syn].tolist()

            if idx_m and idx_s:
               syns_m = [s.strip() for s in str(df_w.at[idx_m[0],
'Синоним']).split(',') if s.strip()]
                syns_s = [s.strip() for s in str(df_w.at[idx_s[0],
'Синоним']).split(',') if s.strip()]
```

```
                seen = set([w_main.lower()])
                final = []
                for s in (syns_m + [w_syn] + syns_s):
                    if s.strip().lower() not in seen:
                        seen.add(s.strip().lower())
                        final.append(s.strip())

                df_w.at[idx_m[0], 'Синоним'] = ", ".join(final)
                df_w = df_w.drop(idx_s[0])
                      self._sync_and_save(df_w,  df_e,  w_main,
self.l_words)
```

```
        def remove_synonym(self):
            sel_syn = self.l_syn.tree.selection()
            sel_word = self.l_words.tree.selection()
            if not (sel_syn and sel_word): return
                                                s_val          =
str(self.l_syn.tree.item(sel_syn[0])['values'][0]).strip()
                                                m_val          =
str(self.l_words.tree.item(sel_word[0])['values'][0]).strip()
```

```
            df_w, df_e = self.get_split_df()
            idx = df_w.index[df_w['Слово'] == m_val].tolist()
            if idx:
                 cur = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                if s_val in cur: cur.remove(s_val)
                df_w.at[idx[0], 'Синоним'] = ", ".join(cur)
                if s_val not in df_w['Слово'].values:
                                    df_w   =   pd.concat([df_w,
pd.DataFrame([{"Слово": s_val, "Синоним": "", "Исключение": ""}])],
ignore_index=True)
                        self._sync_and_save(df_w,  df_e,  s_val,
self.l_words)
```

```
        def transfer_with_exceptions(self):
            path = get_scroll_path()
            try:
                # Читаем из Sheet3
```

```
                df1 = pd.read_excel(path, sheet_name='Sheet3')
                df_w, df_e = self.get_split_df()

               ban = set(df_e["Исключение"].astype(str).unique())
| set(df_w["Слово"].astype(str).unique())
                for s_str in df_w["Синоним"].astype(str):
                   for p in s_str.split(','): ban.add(p.strip())
```

```
                                      source    =    df1.iloc[:,
0].dropna().astype(str).unique()
                new = [w for w in source if w not in ban]
                if new:
                                     df_w   =   pd.concat([df_w,
pd.DataFrame([{"Слово": w, "Синоним": "", "Исключение": ""} for w
in new])], ignore_index=True)
                self._sync_and_save(df_w, df_e)
            except: pass
```

```
        def add_to_main(self):
            val = self.ent_word.get().strip()
            if not val: return
            df_w, df_e = self.get_split_df()
            if val not in df_w['Слово'].values:
                df_w = pd.concat([df_w, pd.DataFrame([{"Слово":
val, "Синоним": "", "Исключение": ""}])], ignore_index=True)
               self._sync_and_save(df_w, df_e, val, self.l_words)
                self.ent_word.delete(0, tk.END)
```

```
        def delete_selected(self):
            s_exc, s_word, s_syn = self.l_exc.tree.selection(),
self.l_words.tree.selection(), self.l_syn.tree.selection()
            df_w, df_e = self.get_split_df()

            if s_exc:
                                                    v       =
str(self.l_exc.tree.item(s_exc[0])['values'][0])
                df_e = df_e[df_e['Исключение'].astype(str) != v]
            elif s_syn and s_word:
```

```
                                                v_s,      v_m      =
str(self.l_syn.tree.item(s_syn[0])['values'][0]).strip(),
str(self.l_words.tree.item(s_word[0])['values'][0]).strip()
                idx = df_w.index[df_w['Слово'] == v_m].tolist()
                if idx:
                   lst = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                    if v_s in lst: lst.remove(v_s)
                    df_w.at[idx[0], 'Синоним'] = ", ".join(lst)
            elif s_word:
                                                        v        =
str(self.l_words.tree.item(s_word[0])['values'][0])
                idx = df_w.index[df_w['Слово'] == v].tolist()
                if idx:
                  syns = [s.strip() for s in str(df_w.at[idx[0],
'Синоним']).split(',') if s.strip()]
                    df_w = df_w.drop(idx[0])
                    for s in syns:
                        if s and s not in df_w['Слово'].values:
                                        df_w  =  pd.concat([df_w,
pd.DataFrame([{"Слово":  s,  "Синоним":  "",  "Исключение":  ""}])],
ignore_index=True)
            self._sync_and_save(df_w, df_e)
```

```
        def load_initial_data(self):
            try:
                df_w, df_e = self.get_split_df()
               for p in [self.l_exc, self.l_words, self.l_copy,
self.l_syn]:
                            for  item  in  p.tree.get_children():
p.tree.delete(item)

                  for  e  in  sorted(df_e['Исключение'].tolist(),
key=ukr_rus_sort_key):
                    if str(e).strip(): self.l_exc.tree.insert("",
tk.END, values=(e,))

                for w in df_w['Слово'].tolist():
                            self.l_words.tree.insert("",  tk.END,
values=(w,))
```

```
                                self.l_copy.tree.insert("",  tk.END,
values=(w,))
            except: pass
```

```
    if __name__ == "__main__":
        root = tk.Tk(); root.withdraw()
        app = ScreenBWindow(root)
        root.mainloop()
```

WorkFiles4.py код

```
    import tkinter as tk
    from tkinter import ttk, messagebox
    import pandas as pd
    import os
    import threading
    import concurrent.futures
    import re

    # Попытка импорта библиотек
    try:
        from docx import Document
    except ImportError:
        Document = None
    try:
        import fitz  # PyMuPDF
    except ImportError:
        fitz = None
```

```
    # Пути
    BASE_DIR = r"C:\Kod\Program\Basa"
    SCROLL_PATH = os.path.join(BASE_DIR, "Scroll.xlsx")
    MANIFEST_PATH = os.path.join(BASE_DIR, "Manifest.csv")
    OUTPUT_PATH = os.path.join(BASE_DIR, "Basa_Kod.xlsx")
```

```
    class FileAnalystApp:
        def __init__(self, root):
            self.root = root
            self.root.title("Фільтрація та Аналіз вмісту")
```

```
            sw = self.root.winfo_screenwidth()
            sh = self.root.winfo_screenheight()
            width, height = int(sw * 0.8), int(sh * 0.8)
                    self.root.geometry(f"{width}x{height}+{(sw-
width)//2}+{(sh-height)//2}")
```

```
            self.main_container = tk.Frame(self.root)
            self.main_container.pack(fill="both", expand=True)
```

```
            self.check_vars = {}
            self._build_ui()
            self.refresh_display_lists()
```

```
        def _build_ui(self):
            # Верхняя панель фильтров
                self.top_frame  =  tk.Frame(self.main_container,
pady=10)
            self.top_frame.pack(side="top", fill="x")
```

```
                self.left_panel  =  tk.LabelFrame(self.top_frame,
text="Фільтр 1 (Sheet2)", padx=10, pady=10)
                  self.left_panel.pack(side="left",  fill="both",
expand=True, padx=5)

                self.right_panel  =  tk.LabelFrame(self.top_frame,
text="Фільтр 2 (Sheet4)", padx=10, pady=10)
                 self.right_panel.pack(side="right",  fill="both",
expand=True, padx=5)
```

```
            self.setup_filters()
```

```
            # Средняя часть
             self.content_frame  =  tk.Frame(self.main_container,
bg="white", bd=2, relief="groove")
              self.content_frame.pack(fill="both",  expand=True,
pady=10, padx=10)
```

```
                    tk.Label(self.content_frame,   text="ПОТОЧНІ
НАЛАШТУВАННЯ    (Sheet5)",    font=("Arial",    10,    "bold"),
bg="white").pack(pady=5)
```

```
                  self.canvas   =   tk.Canvas(self.content_frame,
bg="white", highlightthickness=0)
               self.scrollbar = ttk.Scrollbar(self.content_frame,
orient="vertical", command=self.canvas.yview)
                 self.scrollable_frame   =   tk.Frame(self.canvas,
bg="white")

              self.scrollable_frame.bind("<Configure>", lambda e:
self.canvas.configure(scrollregion=self.canvas.bbox("all")))
             self.canvas_frame = self.canvas.create_window((0, 0),
window=self.scrollable_frame, anchor="nw")
                   self.canvas.bind('<Configure>',   lambda   e:
self.canvas.itemconfig(self.canvas_frame, width=e.width))
```

```
              self.canvas.configure(yscrollcommand=self.scrollbar
.set)
                    self.canvas.pack(side="left",   fill="both",
expand=True)
              self.scrollbar.pack(side="right", fill="y")
```

```
               self.inner_left  =  tk.Frame(self.scrollable_frame,
bg="white")
                  self.inner_left.pack(side="left",  fill="both",
expand=True)
               self.inner_right = tk.Frame(self.scrollable_frame,
bg="white")
                 self.inner_right.pack(side="right",  fill="both",
expand=True)
```

```
              # Нижняя панель
                 self.bottom_frame  =  tk.Frame(self.root,  bd=1,
relief="sunken", padx=10, pady=10)
              self.bottom_frame.pack(side="bottom", fill="x")
```

```
                self.status_label  =  tk.Label(self.bottom_frame,
text=r"Готовність \ -", font=("Arial", 10, "bold"))
            self.status_label.pack(anchor="w")

                self.log_detail  =  tk.Label(self.bottom_frame,
text="Очікування...", font=("Consolas", 9), fg="#555")
            self.log_detail.pack(anchor="w")

            self.progress_container = tk.Frame(self.bottom_frame)
            self.progress_container.pack(fill="x", expand=True,
pady=5)

                                        self.progress         =
ttk.Progressbar(self.progress_container,     orient="horizontal",
mode="determinate")
                    self.progress.pack(side="left",   fill="x",
expand=True, padx=(5, 10))

                                    self.progress_label       =
tk.Label(self.progress_container, text="0 / 0", font=("Arial", 9,
"bold"), width=12)
            self.progress_label.pack(side="right")

                self.btn_run   =   tk.Button(self.bottom_frame,
text="ЗАПУСТИТИ АНАЛІЗ", bg="#4CAF50", fg="white", font=("Arial",
10, "bold"), padx=20, command=self.start_process)
            self.btn_run.pack(side="right", padx=10)

        # --- УПРАВЛЕНИЕ СПИСКАМИ ---
        def load_excel_safe(self, sheet_name):
            try:
                if os.path.exists(SCROLL_PATH):
                                return  pd.read_excel(SCROLL_PATH,
sheet_name=sheet_name)
                return pd.DataFrame()
            except: return pd.DataFrame()

        def setup_filters(self):
```

```
            df2 = self.load_excel_safe("Sheet2")
             data_l = df2.iloc[:, 0].dropna().tolist() if not
df2.empty else []
                self.combo_l = ttk.Combobox(self.left_panel,
values=data_l, width=30)
            self.combo_l.grid(row=0, column=0, padx=5)
               tk.Button(self.left_panel, text="+", width=3,
command=lambda:     self.add_to_sheet5(self.combo_l,     "Лево",
"Sheet2")).grid(row=0, column=1)
```

```
            df4 = self.load_excel_safe("Sheet4")
             data_r = df4.iloc[:, 0].dropna().tolist() if not
df4.empty else []
                self.combo_r = ttk.Combobox(self.right_panel,
values=data_r, width=30)
            self.combo_r.grid(row=0, column=0, padx=5)
               tk.Button(self.right_panel, text="+", width=3,
command=lambda:     self.add_to_sheet5(self.combo_r,     "Право",
"Sheet4")).grid(row=0, column=1)
```

```
             def   add_to_sheet5(self,   combo_widget,   side,
source_sheet):
            val = combo_widget.get().strip()
            if not val: return
            synonyms = ""
            df_source = self.load_excel_safe(source_sheet)
            if not df_source.empty:
               match = df_source[df_source.iloc[:, 0].astype(str)
== val]
                if not match.empty and df_source.shape[1] > 1:
                    synonyms = str(match.iloc[0, 1])
            try:
                with pd.ExcelFile(SCROLL_PATH) as xls:
                    all_sheets = {sn: pd.read_excel(xls, sn) for
sn in xls.sheet_names}
                    new_row = pd.DataFrame({"Сторона": [side],
"Значение": [val], "Синонимы": [synonyms]})
                                        all_sheets["Sheet5"]    =
pd.concat([all_sheets.get("Sheet5",  pd.DataFrame()),  new_row],
ignore_index=True)
```

```
                                with   pd.ExcelWriter(SCROLL_PATH,
engine='openpyxl') as writer:
                                for  sn,  df  in  all_sheets.items():
df.to_excel(writer, sheet_name=sn, index=False)
                    self.refresh_display_lists()
                except Exception as e: print(e)
```

```
            def delete_item(self, index):
                try:
                    with pd.ExcelFile(SCROLL_PATH) as xls:
                        all_sheets = {sn: pd.read_excel(xls, sn) for
sn in xls.sheet_names}
                    if "Sheet5" in all_sheets:
                        df = all_sheets["Sheet5"]
                        df = df.drop(index).reset_index(drop=True)
                        all_sheets["Sheet5"] = df
                                 with  pd.ExcelWriter(SCROLL_PATH,
engine='openpyxl') as writer:
                                for  sn,  sdf  in  all_sheets.items():
sdf.to_excel(writer, sheet_name=sn, index=False)
                        self.refresh_display_lists()
                except Exception as e:
                      messagebox.showerror("Помилка", f"Не вдалося
видалити: {e}")
```

```
            def refresh_display_lists(self):
                  for  widget  in  self.inner_left.winfo_children():
widget.destroy()
                for  widget  in  self.inner_right.winfo_children():
widget.destroy()
                self.check_vars.clear()
                df5 = self.load_excel_safe("Sheet5")
                if df5.empty: return
                for index, row in df5.iterrows():
                    side = str(row.get('Сторона', ''))
                    val = str(row.get('Значение', ''))
                    target_frame = self.inner_left if side == "Лево"
else self.inner_right
                     row_frame = tk.Frame(target_frame, bg="white",
pady=2)
```

```
                row_frame.pack(fill="x", padx=10, anchor="w")
                lbl_color = "blue" if side == "Лево" else "green"
                tk.Label(row_frame, text=val, font=("Arial", 10),
bg="white", fg=lbl_color, width=20, anchor="w").pack(side="left")
                var = tk.BooleanVar(value=True)
                self.check_vars[index] = var
                        tk.Checkbutton(row_frame,  variable=var,
bg="white").pack(side="left", padx=5)
                         tk.Button(row_frame,  text="Видалити",
font=("Arial", 8), bg="#FFEBEE", width=10, command=lambda i=index:
self.delete_item(i)).pack(side="left", padx=5)
```

```
        # --- ИНТЕГРИРОВАННАЯ ЛОГИКА АНАЛИЗА ---
        def start_process(self):
                        threading.Thread(target=self.main_logic,
daemon=True).start()
```

```
        def main_logic(self):
            self.btn_run.config(state="disabled")

            # ЭТАП 1: ФИЛЬТРЫ (С очисткой от nan)
                 self.status_label.config(text=r"Етап  1:  Збір
фільтрів", fg="black")
            df5 = self.load_excel_safe("Sheet5")
            active_filters = []
            if not df5.empty:
                for idx, var in self.check_vars.items():
                    if var.get() and idx < len(df5):
                        row = df5.iloc[idx]
                            main_term = str(row.get('Значение',
'')).strip().lower()
                        if not main_term or main_term == 'nan':
continue

                        syn_str = str(row.get('Синонимы', ''))
                        syns = []
                        if syn_str.lower() != 'nan':
                             syns = [s.strip().lower() for s in
syn_str.split(',')
```

```
                                                    if s.strip() and
s.strip().lower() != 'nan']

                        active_filters.append({'name': main_term,
'patterns': list(set([main_term] + syns))})
```

```
            # ЭТАП 2: МАНИФЕСТ (С поддержкой пробелов и папок)
               self.status_label.config(text=r"Етап 2: Читання
Manifest.csv", fg="black")
            final_file_paths = []
              valid_exts = ('.xlsx', '.xls', '.xlsm', '.docx',
'.doc', '.pdf', '.txt', '.csv', '.log')

            if not os.path.exists(MANIFEST_PATH):
               messagebox.showerror("Помилка", "Manifest.csv не
знайдено.")
                self.btn_run.config(state="normal"); return
```

```
            try:
                with open(MANIFEST_PATH, 'r', encoding='utf-8',
errors='ignore') as f:
                    for line in f:
                        line = line.strip()
                        if not line: continue
                         # Захват пути до расширения (с учетом
пробелов)
                                    match = re.search(r'([a-zA-
Z]:\\.*?\.(?:xlsx|xls|xlsm|docx|doc|pdf|txt|csv|log))',    line,
re.IGNORECASE)
                        if match:
                            entry_path = match.group(1).strip()
                            if os.path.exists(entry_path):
                             final_file_paths.append(entry_path)
                        else:
                             # Если это папка (без расширения в
строке)
                                                folder_path =
re.split(r'\s+\d+\s+byte|Sheet:', line)[0].strip()
                            if os.path.isdir(folder_path):
```

```
                                        for root, _, files in
os.walk(folder_path):
                                    for file in files:
                                                            if
file.lower().endswith(valid_exts):
                                                final_file_paths.ap
pend(os.path.join(root, file))
                final_file_paths = list(set(final_file_paths))
            except Exception as e:
                    messagebox.showerror("Помилка", f"Помилка
маніфесту: {e}")
                self.btn_run.config(state="normal"); return
```

```
            total_files = len(final_file_paths)
            if total_files == 0:
                  messagebox.showwarning("Увага", "Не знайдено
файлів для аналізу.")
                self.btn_run.config(state="normal"); return
```

```
            # ЭТАП 3: ПОДГОТОВКА ФАЙЛА
             self.status_label.config(text=r"Етап 3: Підготовка
файлу", fg="black")
            try:
                                    pd.DataFrame(columns=["Путь",
"Знайдено"]).to_excel(OUTPUT_PATH, index=False)
            except Exception as e:
                  messagebox.showerror("Помилка", f"Не вдалося
створити файл: {e}")
                self.btn_run.config(state="normal"); return
```

```
            # ЭТАП 4: АНАЛИЗ (ThreadPool + Deep Scan)
              self.status_label.config(text=r"Етап 4: Аналіз",
fg="black")
            self.progress['maximum'] = total_files
            self.progress['value'] = 0

                                                        with
concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
```

```
                future_to_path = {executor.submit(self.scan_file,
p, active_filters): p for p in final_file_paths}

                                        for     i,     future     in
enumerate(concurrent.futures.as_completed(future_to_path)):
                    p = future_to_path[future]
                    current_count = i + 1
                            self.log_detail.config(text=f"Файл:
{os.path.basename(p)}")
                    self.progress_label.config(text=f"{current_
count} / {total_files}")

                    res = future.result()
                    if res:
                        try:
                                                df_existing   =
pd.read_excel(OUTPUT_PATH)
                            df_combined = pd.concat([df_existing,
pd.DataFrame([res],        columns=["Путь",        "Знайдено"])],
ignore_index=True)
                                df_combined.to_excel(OUTPUT_PATH,
index=False)
                        except: pass

                    self.progress['value'] = current_count
                    self.root.update_idletasks()
```

```
             self.status_label.config(text=r"Виконано \ Успіх",
fg="green")
                    messagebox.showinfo("Готово",    f"Аналіз
завершено!\nФайлів      оброблено:      {total_files}\nРезультат:
{OUTPUT_PATH}")
            self.btn_run.config(state="normal")
```

```
        def scan_file(self, path, filters):
            """Интегрированная логика глубокого поиска."""
            if not os.path.exists(path): return None
            ext = os.path.splitext(path)[1].lower()
            content = ""
            try:
```

```
                if ext in ['.xlsx', '.xls', '.xlsm']:
                    with pd.ExcelFile(path) as xls:
                        # Читаем ВСЕ листы
                         content = " ".join([pd.read_excel(xls,
sn).to_string() for sn in xls.sheet_names]).lower()
                elif ext in ['.txt', '.csv', '.log']:
                        with open(path, 'r', encoding='utf-8',
errors='ignore') as f:
                            content = f.read().lower()
                elif ext in ['.docx', '.doc'] and Document:
                         content = "\n".join([p.text for p in
Document(path).paragraphs]).lower()
                elif ext == '.pdf' and fitz:
                    with fitz.open(path) as doc:
                         content = "".join([page.get_text() for
page in doc]).lower()

                if not content.strip(): return None
```

```
                # Поиск целого слова (regex)
                found = []
                for f in filters:
                    for pattern in f['patterns']:
                      if re.search(rf'\b{re.escape(pattern)}\b',
content):
                            found.append(f['name'])
                            break

                 return [path, ", ".join(set(found))] if found
else None
            except:
                return [path, "ПОМИЛКА ЧИТАННЯ"]
```

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

Dop.py код

```
    import pandas as pd
    import os
    import sys
    import re
    from itertools import zip_longest

    # ANSI цвета для консоли
    COLOR_KEY = '\033[92m'  # Зеленый
    COLOR_RESET = '\033[0m'
```

```
    def clean_and_split(text):
        """
        Очищает текст и разбивает на уникальные слова.
        Исключает слова короче 3-х символов и слова с цифрами.
        """
        if not isinstance(text, str):
            return []

        # Заменяем всё, кроме букв и дефисов, на пробелы
        cleaned_text = re.sub(r'[^a-zA-Za-яА-ЯёЁ\s-]', ' ', text)
        words = cleaned_text.lower().split()

        # Оставляем только длинные слова без цифр
        return [w for w in words if len(w) > 2 and w.isalpha()]
```

```
    def analyze_and_check_manifest(manifest_path, output_path):
        if not os.path.exists(manifest_path):
            print(f"Помилка: Манифест не знайдено")
            return
```

```
        try:
            # 1. Загрузка данных
            print("Завантаження Манифесту...")
                  df_manifest   =   pd.read_csv(manifest_path,   sep=';',
encoding='utf-8-sig', on_bad_lines='skip')
            df_manifest = df_manifest.fillna("")
            total_rows = len(df_manifest)
```

```
            # 2. Сбор ключей из Манифеста для файла Scroll
            print("Збір ключей-слів...")
            unique_keys = set()
            for _, row in df_manifest.iterrows():
                row_text = " ".join(map(str, row.values))
                unique_keys.update(clean_and_split(row_text))

            # Сортируем и сохраняем в Scroll (Лист 1)
```

```
            sorted_keys = sorted(list(unique_keys))
            df_scroll = pd.DataFrame(sorted_keys, columns=['Ключи-слова'])

            with pd.ExcelWriter(output_path, engine='openpyxl', mode='w')
as writer:
                df_scroll.to_excel(writer, sheet_name='Список слов',
index=False)
            print(f"Файл {output_path} оновлено. Знайдено ключів:
{len(sorted_keys)}")
```

```
            # 3. Построчная проверка Манифеста по собранным ключам
            print("\n--- Початок рядкової перевірки ---")
            global_match_mask = [False] * total_rows

            for idx, row in df_manifest.iterrows():
                # Преобразование в строку для поиска и вывода
                line_str = " | ".join(map(str, row.values))
                line_lower = line_str.lower()

                display_line = line_str
                has_any_key = False
```

```
                # Поиск совпадений с ключами
                for key in sorted_keys:
                    if key in line_lower:
                        has_any_key = True
                        # Подсветка найденного ключа
                       pattern = re.compile(re.escape(key), re.IGNORECASE)
                                                          display_line    =
pattern.sub(f"{COLOR_KEY}\\g<0>{COLOR_RESET}", display_line)
```

```
                # Вывод в консоль с логикой пауз
                if has_any_key:
                    global_match_mask[idx] = True
                    print(f"\n[{idx+1}/{total_rows}] {display_line}")
                else:
                    print(f"\n[{idx+1}/{total_rows}] {line_str}")
                     input("--> Збігів із ключами не знайдено. Натисніть
Enter, щоб продовжити...")
```

```
            print(f"\nПеревірку завершено. Оброблено рядків: {total_rows}")
```

```
        except Exception as e:
            print(f"\nПомилка системи: {e}")
```

```
def main():
    # Пути к файлам
    base_dir = r"C:\Kod\Program\Basa"
    manifest_file = os.path.join(base_dir, "Manifest.csv")
    scroll_file = os.path.join(base_dir, "Scroll.xlsx")

    analyze_and_check_manifest(manifest_file, scroll_file)
```

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

Dop2.py код

```
import pandas as pd
import os
import re
import time
import warnings
from openpyxl import load_workbook
from tqdm import tqdm

# Игнорируем технические предупреждения
warnings.filterwarnings('ignore')
```

```
COLOR_GREEN = '\033[92m'
COLOR_YELLOW = '\033[93m'
COLOR_RED = '\033[91m'
COLOR_CYAN = '\033[96m'
COLOR_RESET = '\033[0m'
```

```
TBL_EXT = ('.xlsx', '.xls', '.csv', '.ods', '.xlsm')
```

```
def get_sort_priority(text):
    """
    Присваивает приоритет слову для сортировки:
    1 - Украинский (содержит і, ї, є, ґ)
    2 - Русский (кириллица без специфических укр. букв)
    3 - Остальные (латиница, цифры)
    """
    text_lower = text.lower()

    # Регулярные выражения для определения языка
    ukr_chars = re.compile(r'[іїєґ]')
    rus_only_chars = re.compile(r'[ыэъё]') # Буквы, которых нет в украинском
    cyrillic = re.compile(r'[а-яёіїєґ]')
```

```
            if ukr_chars.search(text_lower):
                return (1, text_lower)
            elif cyrillic.search(text_lower):
                return (2, text_lower)
            else:
                return (3, text_lower)
```

```
        def process_cell_text(text):
            """Логика: до 3 слов в ячейке — единый термин."""
            if not isinstance(text, str) or not text.strip():
                return []

            clean_text = re.sub(r'[^a-zA-Za-яA-ЯёЁіїєґІЇЄҐ0-9\s-]', ' ', text).strip()
            words = clean_text.split()

            if 1 <= len(words) <= 3:
                term = " ".join(words)
                if any(c.isalpha() for c in term) and len(term) > 2:
                    return [term]
                return []
            else:
                return [w for w in words if len(w) > 2 and any(c.isalpha() for c in w)]
```

```
        def find_table_headers(file_path):
            headers_found = set()
            file_path = os.path.normpath(file_path.strip().strip("").strip("""))
            ext = os.path.splitext(file_path)[1].lower()

            try:
                if ext == '.csv':
                    df_dict = {'Sheet1': pd.read_csv(file_path, sep=None, engine='python', header=None,
on_bad_lines='skip')}
                else:
                    df_dict = pd.read_excel(file_path, header=None, sheet_name=None)
```

```
                for _, df in df_dict.items():
                    if df is None or df.empty: continue
                    df = df.dropna(how='all', axis=0).dropna(how='all', axis=1)
                    sample = df.head(15)
                    row_counts = sample.count(axis=1)
                    if row_counts.empty or row_counts.max() == 0: continue

                    header_idx = row_counts.idxmax()
                    header_row = df.loc[header_idx].astype(str).tolist()

                    for cell in header_row:
                        if cell and cell.lower() != 'nan':
```

```
                    headers_found.update(process_cell_text(cell))
            return headers_found
        except:
            return None

def main():
    B_DIR = r"C:\Kod\Program\Basa"
    MANIFEST = os.path.join(B_DIR, "Manifest.csv")
    SCROLL = os.path.join(B_DIR, "Scroll.xlsx")

        print(f"{COLOR_CYAN}>>> ЗАПУСК ОБРОБКИ З СОРТУВАЛЬНЯМ {COLOR_RESET}")

    if not os.path.exists(MANIFEST):
        print(f"{COLOR_RED}[ПОМИЛКА]{COLOR_RESET} Манифест не знайдено.")
        return

    # Чтение манифеста (пути во 2-м столбце)
    df_manifest = pd.read_csv(MANIFEST, sep=';', encoding='utf-8-sig', header=None)
    raw_paths = df_manifest[1].dropna().astype(str).tolist()
        valid_paths = [p.strip().strip('"') for p in raw_paths if p.strip().strip('"').lower().endswith(TBL_EXT)]

    all_terms = set()

    for path in tqdm(valid_paths, desc="Обробка таблиць", unit="файл", colour="green"):
        if os.path.exists(path):
            result = find_table_headers(path)
            if result:
                all_terms.update(result)

    # --- СОРТИРОВКА ---
    print(f"{COLOR_YELLOW}Сортування (Укр -> Рус -> Другие)...{COLOR_RESET}")
    # Используем функцию get_sort_priority как ключ
    sorted_terms = sorted(list(all_terms), key=get_sort_priority)
    # ------------------

    df_result = pd.DataFrame(sorted_terms, columns=['Термины из заголовков'])

    try:
        if os.path.exists(SCROLL):
            try:
                os.rename(SCROLL, SCROLL)
            except OSError:
                print(f"{COLOR_RED}[БЛОКИРОВКА]{COLOR_RESET} Закройте Scroll.xlsx!")
```

```
                return

            with pd.ExcelWriter(SCROLL, engine='openpyxl', mode='a', if_sheet_exists='replace') as
writer:
                df_result.to_excel(writer, sheet_name='Заголовки таблиц', index=False)

            print(f"\n{COLOR_GREEN}=== РАБОТУ ЗАВЕРШЕНО ==={COLOR_RESET}")
            print(f"Оброблено файлів: {COLOR_CYAN}{len(valid_paths)}{COLOR_RESET}")
                print(f"Розмір  словника:  {COLOR_CYAN}{len(sorted_terms)}{COLOR_RESET}
строк.")

        except Exception as e:
            print(f"{COLOR_RED}[ПОМИЛКА]{COLOR_RESET}: {e}")

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

Eb.py код

```
    import pandas as pd
    import os
    import re

    # --- Инициализация библиотек для работы с документами ---
    try:
        from docx import Document
    except ImportError:
        Document = None
    try:
        import fitz  # PyMuPDF
    except ImportError:
        fitz = None

    # --- КОНФИГУРАЦИЯ ПУТЕЙ ---
    BASE_DIR = r"C:\Kod\Program\Basa"
    SCROLL_PATH = os.path.join(BASE_DIR, "Scroll.xlsx")
    MANIFEST_PATH = os.path.join(BASE_DIR, "Manifest.csv")
    OUTPUT_PATH = os.path.join(BASE_DIR, "Basa_Kod.xlsx")

    def log(message):
        """Вывод системных сообщений."""
        print(f"[LOG] {message}")

    def get_active_filters():
        """БЛОК 1: Загрузка фильтров с очисткой от пустых значений (nan)."""
```

```
            log("--- Шаг 1: Загрузка фильтров ---")
            try:
                df = pd.read_excel(SCROLL_PATH, sheet_name="Sheet5")
                active_filters = []
                for _, row in df.iterrows():
                    val = str(row.get('Значение', '')).strip()
                    if not val or val.lower() == 'nan':
                        continue

                    name = val.lower()
                    raw_syns = str(row.get('Синонимы', ''))
                    syns = []
                    if raw_syns.lower() != 'nan' and raw_syns.strip():
                        syns = [s.strip().lower() for s in raw_syns.split(',')
                                if s.strip() and s.strip().lower() != 'nan']

                    patterns = list(set([name] + syns))
                    active_filters.append({'name': name, 'patterns': patterns})
                log(f"Загружено активных фильтров: {len(active_filters)}")
                return active_filters
            except Exception as e:
                log(f"Ошибка в Scroll.xlsx: {e}")
                return []
```

```
        def extract_paths_from_manifest():
            """БЛОК 2: Извлечение путей. Исправлено под пробелы и структуру Manifest.csv."""
            log("--- Шаг 2: Анализ путей в Manifest.csv ---")
            final_files = []
            valid_exts = ('.xlsx', '.xls', '.xlsm', '.docx', '.doc', '.pdf', '.txt', '.csv', '.log')

            if not os.path.exists(MANIFEST_PATH):
                log(f"Манифест не знайдено за шляхом: {MANIFEST_PATH}")
                return []
```

```
            try:
                with open(MANIFEST_PATH, 'r', encoding='utf-8', errors='ignore') as f:
                    for line in f:
                        line = line.strip()
                        if not line: continue

                            # НОВАЯ ЛОГИКА: Отрезаем всё, что идет после расширения файла или технических пометок
                            # Ищем подстроку, начинающуюся с диска (C:\) и заканчивающуюся на расширении
                        match = re.search(r'([a-zA-Z]:\\.*?\.(?:xlsx|xls|xlsm|docx|doc|pdf|txt|csv|log))', line, re.IGNORECASE)

                        if match:
```

```
                    entry_path = match.group(1).strip()
                else:
                  # Если расширения нет (это папка), берем часть строки до первого упоминания 'byte' или 'Sheet'
                    entry_path = re.split(r'\s+\d+\s+byte|Sheet:', line)[0].strip()

                if os.path.exists(entry_path):
                    if os.path.isdir(entry_path):
                        # Рекурсивный поиск файлов внутри папки
                        for root, dirs, files in os.walk(entry_path):
                            for file in files:
                                if file.lower().endswith(valid_exts):
                                    final_files.append(os.path.join(root, file))
                    else:
                        final_files.append(entry_path)
                else:
                    # Логируем только реальные ошибки, чтобы не спамить
                    if entry_path.startswith(('C:', 'c:')):
                        log(f"Шлях не знайдено: {entry_path}")

            unique_files = list(set(final_files))
            log(f"Всього файлів для анализу: {len(unique_files)}")
            return unique_files
        except Exception as e:
            log(f"Помилка читання Manifest.csv: {e}")
            return []
```

```
    def scan_file(path, filters):
        """БЛОК 3: Глубокое сканирование контента."""
        ext = os.path.splitext(path)[1].lower()
        content = ""
        try:
            if ext in ['.xlsx', '.xls', '.xlsm']:
                with pd.ExcelFile(path) as xls:
                    # Читаем все листы (актуально для биометрии и журналов)
                    content = " ".join([pd.read_excel(xls, sn).to_string() for sn in xls.sheet_names]).lower()
            elif ext in ['.txt', '.csv', '.log']:
                with open(path, 'r', encoding='utf-8', errors='ignore') as f:
                    content = f.read().lower()
            elif ext in ['.docx', '.doc'] and Document:
                content = "\n".join([p.text for p in Document(path).paragraphs]).lower()
            elif ext == '.pdf' and fitz:
                with fitz.open(path) as doc:
                    content = "".join([pg.get_text() for pg in doc]).lower()

            if not content.strip(): return None
```

```
            found = []
            for f in filters:
                for p in f['patterns']:
                    # Поиск целого слова
```

```
                        if re.search(rf'\b{re.escape(p)}\b', content):
                            log(f"   [+] Знайдено '{p}' в {os.path.basename(path)}")
                            found.append(f['name'])
                            break
                return [path, ", ".join(sorted(set(found)))] if found else None
            except:
                return None
```

```
        def main():
            """ГЛАВНЫЙ ЦИКЛ."""
            log("=== ЗАПУСК ОТЛАДКИ V5.1 (ИСПРАВЛЕНЫ ПРОБЕЛЫ) ===")

            filters = get_active_filters()
            file_paths = extract_paths_from_manifest()

            if not filters or not file_paths:
                log("Пауза: немає даных для работы.")
                return
```

```
            # Очистка старого отчета
            pd.DataFrame(columns=["Шлях", "Знайдено"]).to_excel(OUTPUT_PATH, index=False)
```

```
            log(f"--- Шаг 3: Сканування {len(file_paths)} файлів ---")
            for path in file_paths:
                result = scan_file(path, filters)
                if result:
                    try:
                        df = pd.read_excel(OUTPUT_PATH)
                                                pd.concat([df,    pd.DataFrame([result],    columns=["Шлях",    "Знайдено"])],
ignore_index=True).to_excel(OUTPUT_PATH, index=False)
                    except:
                        pass
```

```
            log(f"=== ГОТОВО. Результаты збережено в: {OUTPUT_PATH} ===")
            input("Нажмите Enter...")
```

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