НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«17» грудня 2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека та захист інформації»

на тему: Встановлення потоків поширення інформації через месенджери в задачах кібербезпеки інформаційного простору

Виконав (-ла): здобувач вищої освіти II курсу, групи ФБ-31мп
(шифр групи)

Чорний Анатолій Юрійович
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафедри ІБ к.т.н. Стьопочкіна І.В.
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент кафедри ММЗІ к.т.н. Кучинська Н.В.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський)
Спеціальність – 125 «Кібербезпека та захист інформації»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«17» грудня 2024 р.

ЗАВДАННЯ
на магістерську дисертацію здобувачу вищої освіти

Чорному Анатолію Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Встановлення потоків поширення інформації через месенджери в задачах кібербезпеки інформаційного простору

,

керівник роботи: Стьопочкіна Ірина Валеріївна, кандидат наук, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «08» листопада 2024 р. № 5013-с

2. Термін подання здобувачем вищої освіти роботи «11» грудня 2024 р.

3. Вихідні дані до роботи: документація по Telegram API, документація по neo4j API, літературні та інформаційні джерела.

4. Зміст роботи:

1. Вступ
2. Поширення інформації через месенджери: ризики та виклики кібербезпеки
3. Аналіз підходів до виявлення джерел та потоків інформації у месенджерах
4. Розробка та реалізація системи для виявлення потоків поширення інформації
5. Розробка стартап-проєкту

6. Висновки

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Презентація до захисту дисертації.

6. Дата видачі завдання: 01.09.2024

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.09.2024	Виконано
2	Вивчення та аналіз літератури	01.09.2024 - 15.09.2024	Виконано
3	Аналіз підходів до виявлення джерел та потоків інформації у месенджерах	15.09.2024 - 10.10.2024	Виконано
4	Написання та тестування програмного коду	10.10.2024 - 31.10.2024	Виконано
5	Аналіз результатів	31.10.2024 - 15.11.2024	Виконано
6	Проходження переддипломної практики	02.09.2024— 27.10.2024	Виконано
7	Написання магістерської дисертації	15.11.2024 - 01.12.2024	Виконано
8	Отримання допуску до захисту	01.12.2024	Виконано
9	Оформлення магістерської дисертації	01.12.2024 - 15.12.2024	Виконано
10	Захист магістерської дисертації	17.12.2024	Виконано

Здобувач вищої освіти _____ Анатолій ЧОРНИЙ
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Ірина СТЬОПОЧКІНА
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Робота обсягом 111 сторінок, містить 9 ілюстрацій, 17 таблиць, 1 додаток та 17 літературних посилань.

Метою даної роботи є розвиток методів аналізу потоків інформації у месенджерах із використанням графових БД на прикладі Telegram для задач безпеки кіберпростору.

Об’єктом дослідження є інформація у месенджерах.

Предметом дослідження є алгоритми встановлення походження та спорідненості інформації із різних інформаційних середовищ.

Методами дослідження є аналіз літературних джерел, проведення експерименту з використанням програмних засобів.

Результати роботи представлені у вигляді графів, таблиць, рисунків, що ілюструють мережі поширення інформації, а також у вигляді програмного забезпечення, яке реалізує автоматизовану систему збору даних та аналізу мереж у месенджерах.

Результати роботи можуть бути використані для моніторингу й аналізу інформаційних потоків у месенджерах у контексті забезпечення кібербезпеки та захисту інформаційного простору.

Результати роботи доповідалися на 1 Міжнародній науково-практичній конференції «New Horizons in Scientific Research: Challenges and Solutions»

Ключові слова: кібербезпека, інформаційні потоки, месенджери, графові бази даних, автоматизація збору даних.

ABSTRACT

The paper is 111 pages long, contains 9 illustrations, 17 tables, 1 appendix, and 17 references.

The purpose of this work is to develop methods for analyzing information flows in messengers using graph databases on the example of Telegram.

The object of study is information in messengers.

The subject of the study is algorithms for establishing the origin and affinity of information from different information environments.

The research methods are the analysis of literary sources, conducting an experiment using software tools.

The results of the work are presented in the form of graphs, tables, figures illustrating information dissemination networks, as well as in the form of software that implements an automated system for collecting data and analyzing networks in messengers.

The results of the work can be used to monitor and analyze information flows in messengers in the context of cybersecurity and information space protection.

The results of the work were presented at the 1st International Scientific and Practical Conference “New Horizons in Scientific Research: Challenges and Solutions”

Keywords: cybersecurity, information flows, messengers, graph databases, data collection automation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - База даних.

API – Application Programming Interface, програмний інтерфейс, що забезпечує доступ до функцій і даних платформи.

Neo4j – графова база даних, призначена для зберігання та аналізу даних у вигляді графів.

Репост - Використання чужого контенту в себе у блозі чи в соціальній мережі, з посиланням на автора.

Хештег – маркер у тексті, що починається зі знака # і використовується для класифікації контенту.

Медіафайл – цифровий файл, що містить мультимедійний контент (фото, відео, аудіо).

Домен – унікальне ім’я веб-ресурсу, зазначене в посиланні.

ВСТУП

Сучасний інформаційний простір дедалі більше залежить від соціальних мереж та месенджерів, які стали ключовими платформами для обміну інформацією. Одним із найбільш популярних месенджерів є Telegram, що забезпечує широкі можливості для швидкого поширення даних, зокрема й дезінформації, шкідливих повідомлень або контенту, що порушує кібербезпеку. У зв'язку з цим виникає необхідність аналізу потоків інформації для своєчасного виявлення потенційних загроз та запобігання їхньому впливу на користувачів і інформаційний простір загалом.

Проблеми кібербезпеки, пов'язані з поширенням інформації через месенджери, мають кілька аспектів:

- Поширення дезінформації та чуток, які можуть викликати паніку або агресію серед аудиторії.
- Розповсюдження шкідливих програм, що загрожують безпеці пристроїв користувачів.
- Атаки на інформаційний простір, спрямовані на маніпуляцію громадською думкою.

Існуючі методи аналізу інформаційних потоків здебільшого зосереджені на загальному моніторингу або фільтрації контенту, але часто не враховують специфіку месенджерів, що мають графоподібну структуру зв'язків між користувачами та каналами. У такому контексті стає актуальним використання технологій графових БД, які дозволяють моделювати та аналізувати мережі поширення інформації, встановлювати її джерела та ідентифікувати зв'язки між учасниками інформаційного обміну.

Актуальність даної роботи обумовлюється відсутністю вітчизняних рішень, орієнтованих на специфіку аналізу інформаційних потоків у месенджерах,

зокрема в Telegram, що є одним з найбільш популярних каналів комунікації в Україні. Існуючі міжнародні підходи здебільшого зосереджені на загальних методах аналізу інформаційних мереж або орієнтовані на інші платформи, що не враховують особливостей українського інформаційного простору, таких як мовні та культурні чинники. Водночас вивчення мереж поширення інформації в месенджерах є критично важливим для забезпечення кібербезпеки, оскільки вони часто є каналами для поширення шкідливої інформації, пропаганди та дезінформації. У зв'язку з цим, розробка ефективних методів аналізу інформаційних потоків у месенджерах, зокрема в Telegram, з використанням сучасних технологій графових БД, є нагальною потребою для вирішення проблем захисту національної кібербезпеки та інформаційного простору України.

Метою даної роботи є розвиток методів аналізу потоків інформації у месенджерах із використанням графових БД на прикладі Telegram для задач безпеки кіберпростору.

Для досягнення мети необхідно виконати такі завдання:

- провести огляд існуючих методів аналізу інформаційних потоків у месенджерах;
- дослідити можливості використання Telegram API для збору даних;
- розробити архітектуру системи для аналізу мереж поширення інформації;
- реалізувати автоматизовану систему збору даних;
- протестувати розроблену систему на реальних даних та оцінити її ефективність.

Об’єктом дослідження є інформація, що поширюється у месенджерах.

Предметом дослідження є методи та алгоритми встановлення походження і спорідненості інформації з різних каналів у месенджерах.

Наукова новизна: запропоновано вдосконалений алгоритм встановлення походження та спорідненості інформації у месенджерах, який відрізняється від

існуючих інтеграцією автоматизованого збору даних із використанням Telegram API та застосуванням графових БД для моделювання мереж поширення. Алгоритм забезпечує підвищену точність і продуктивність аналізу за рахунок ефективної обробки великих обсягів даних і виявлення як явних, так і прихованих зв'язків між каналами.

Практичне значення: результати роботи можуть бути застосовані для моніторингу та аналізу інформаційних потоків у месенджерах, що сприятиме забезпеченню кібербезпеки та захисту інформаційного простору.

Апробація: результати роботи були представлені на 1-й Міжнародній науково-практичній конференції «New Horizons in Scientific Research: Challenges and Solutions».

1 ПОШИРЕННЯ ІНФОРМАЦІЇ ЧЕРЕЗ МЕСЕНДЖЕРИ: РИЗИКИ ТА ВИКЛИКИ КІБЕРБЕЗПЕКИ

1.1 Поняття та роль месенджерів у сучасному інформаційному просторі

Месенджери є невід'ємною частиною сучасного цифрового суспільства. Вони слугують платформами для швидкого й ефективного обміну інформацією. Завдяки високій доступності та легкості використання месенджери стали популярними як серед пересічних громадян, так і серед бізнесу, державних структур та організацій. Сьогодні такі сервіси, як Telegram, WhatsApp, Signal, Viber, Facebook Messenger, охоплюють мільярди користувачів по всьому світу, формуючи масштабні мережі взаємодії [1].

Основною особливістю месенджерів є їх здатність передавати інформацію практично миттєво. У глобалізованому світі це значно прискорює обмін даними, підвищує ефективність комунікації та сприяє побудові інформаційного суспільства. Разом з тим, активне використання месенджерів породжує нові виклики для кібербезпеки, оскільки ці платформи також слугують засобом для поширення дезінформації, пропаганди, шкідливого контенту та координації злочинних дій.

Месенджери мають ключову роль у формуванні сучасного інформаційного простору. Вони не тільки замінюють традиційні засоби комунікації, такі як телефон чи електронна пошта, але й створюють нові можливості для інтерактивності, персоналізації та масштабування. Наприклад, Telegram дозволяє створювати канали та ботів для автоматизації роботи, що стало важливим інструментом у маркетингу, політиці, освіті та навіть кібербезпеці. Завдяки широким функціональним можливостям месенджери стають ядром багатьох соціальних та професійних взаємодій, змінюючи способи, якими люди отримують і поширюють інформацію.

Однак такі особливості також створюють ризики для безпеки. Месенджери часто використовують шифрування для забезпечення конфіденційності, що ускладнює моніторинг потенційно небезпечного контенту. Наприклад, наскрізне шифрування у WhatsApp чи Signal значно ускладнює доступ до інформації навіть для служб правопорядку [2]. У цьому контексті месенджери стають інструментом, який одночасно сприяє свободі слова й приватності, але й слугує прихистком для злочинців, які поширюють дезінформацію або здійснюють кібератаки [3].

Ще одним аспектом впливу месенджерів є їх здатність формувати суспільні настрої та впливати на громадську думку. Завдяки вірусному поширенню контенту через репости, поширення через групи чи канали, інформація може швидко охоплювати великі аудиторії. Така швидкість поширення часто використовується для маніпулювання громадською думкою, організації масових заходів чи навіть дестабілізації політичної ситуації.

Отже, месенджери є не лише засобом комунікації, але й потужним фактором трансформації сучасного інформаційного простору. Їхня роль у суспільстві вимагає глибокого аналізу та підвищеної уваги до ризиків, пов’язаних із їх використанням, що робить цю тему актуальною для досліджень у сфері кібербезпеки

1.2 Представлення месенджеру як соціальний граф

Месенджери можна розглядати не лише як платформи для обміну інформацією, а і як складні соціальні графи, що відображають взаємозв'язки між користувачами, каналами та групами [4]. У контексті аналізу інформаційних потоків соціальний граф є потужним інструментом, який дозволяє моделювати та вивчати динаміку комунікацій, визначати ключові вузли (користувачів або канали) та шляхи поширення контенту.

Соціальний граф — це структура, де вершини представляють суб'єкти

взаємодії, а ребра — зв'язки між ними. У випадку месенджерів вершинами можуть бути індивідуальні користувачі, групи, або канали, тоді як ребра символізують різні типи взаємодії, такі як надсилання повідомлень, підписки, репости чи відповіді. Цей підхід дозволяє створювати візуалізації соціальних мереж та аналізувати взаємозв’язки у межах платформи [5].

Моделювання месенджера як соціального графа відкриває низку можливостей для вивчення інформаційних потоків. Наприклад, через аналіз ступенів вузлів (кількості їх зв’язків) можна визначити найвпливовіших користувачів або канали, які відіграють ключову роль у поширенні інформації.

Особливістю соціального графа у месенджерах є його динамічність. У процесі взаємодії користувачів граф постійно змінюється: додаються нові вершини, створюються або видаляються зв'язки, змінюється інтенсивність комунікації. Ця динамічність ускладнює аналіз, але також дозволяє виявляти тимчасові патерни, такі як сплески активності чи швидке поширення певного контенту.

Ще одним важливим аспектом є багаторівневість взаємодії. У месенджерах існують різні типи зв’язків: від прямих комунікацій між двома користувачами до багатоадресного поширення контенту через канали чи групи. Кожен з цих рівнів може бути представленим у соціальному графі з відповідним типом ребер, що дає змогу моделювати як локальні, так і глобальні процеси поширення інформації.

Розглядаючи месенджер як соціальний граф, можна також оцінювати його вразливість до атак. Наприклад, виявлення «ключових» вузлів дозволяє прогнозувати, як видалення певного каналу або користувача вплине на всю мережу. Це має критичне значення для задач кібербезпеки, оскільки дозволяє розробляти стратегії протидії поширенню шкідливої інформації та підвищувати стійкість інформаційного простору.

Таким чином, представлення месенджера у вигляді соціального графа є потужним інструментом для моделювання та аналізу процесів поширення інформації. Цей підхід дозволяє не лише краще зрозуміти структуру та динаміку

взаємодій, але й підвищити ефективність заходів із захисту інформаційного простору.

1.3 Визначення інформаційних потоків у месенджерах та їх загрози

Інформаційні потоки у месенджерах — це динамічні канали передачі даних, які включають різноманітні види контенту, що поширюються серед користувачів, таких як текстові повідомлення, медіафайли, посилання та файли. Поширення цієї інформації є важливим елементом сучасного інформаційного простору, особливо у умовах геополітичних конфліктів, таких як російсько-українська війна. У цей період месенджери виступають не тільки як засоби комунікації, але й як інструменти маніпулювання громадською думкою, дезінформації та психологічного впливу.

Загроза, яку несуть інформаційні потоки у месенджерах, полягає в тому, що вони можуть використовуватися для поширення шкідливого контенту, маніпуляцій та пропаганди. У разі війни, як у випадку з російською агресією проти України, месенджери стають каналами для дезінформації, фейкових новин, а також координації дій терористичних груп і підривної діяльності. Агресор активно використовує месенджери для розповсюдження неправдивої інформації, поширення паніки, створення фальшивих новин про ситуацію на фронті, а також для мобілізації своїх прихильників і радикалів через анонімні канали [7].

Одним з прикладів такої загрози є використання месенджерів для поширення фальшивих новин, що стосуються військових подій. Російські пропагандисти активно створюють канали та групи, в яких публікують неправдиві або перекручені інформаційні повідомлення, спрямовані на підрив морального духу українського населення і військових. Така інформація може містити фейки про поразки українських сил, сфабриковані дані про мирні переговори або звинувачення на адресу українських керівників. З допомогою соціальних графів і

механізмів репостів ці повідомлення швидко поширюються серед широкої аудиторії, включаючи користувачів, які не завжди можуть перевірити достовірність інформації.

Крім того, в умовах війни особливу увагу слід приділяти інформаційним потокам, які використовуються для проведення кібератак. Військові операції росії супроводжуються не лише фізичними атаками, але й кібератаками, які можуть бути організовані через месенджери. Наприклад, через такі платформи, як Telegram, можуть бути організовані фішингові кампанії, поширення шкідливих програм чи вірусів, спрямованих на компрометацію систем критичної інфраструктури або на крадіжку персональних даних. Ці атаки можуть мати серйозні наслідки для національної безпеки, оскільки інформаційні технології стали важливим елементом в управлінні державою, фінансовими потоками та військовими операціями.

Ще однією важливою загрозою є створення і просування «фальшивих» інформаторів або експертів, які надають спотворену інформацію про події. Вони можуть використовувати месенджери для розповсюдження вигаданих історій, що призводять до формування хибного розуміння ситуації серед аудиторії. Це включає як окремих користувачів, так і великі групи, що беруть участь у політичних або громадських обговореннях, таких як вибори або міжнародні конференції. В умовах війни така діяльність може бути спрямована на формування негативного іміджу України на міжнародній арені, що посилює загрозу інформаційної ізоляції та погіршення міжнародних відносин.

Відповідно, необхідно розглядати інформаційні потоки в месенджерах як один з основних каналів, через який можуть бути спричинені серйозні кіберзагрози. Це вимагає від державних і приватних органів постійної уваги до моніторингу та аналізу таких потоків [8], а також впровадження відповідних механізмів для їх блокування, фільтрації та виявлення дезінформації. Такі заходи повинні включати як технічні методи, такі як розпізнавання шкідливого контенту та автоматичне блокування, так і правові й організаційні стратегії, спрямовані на

запобігання маніпулюванню громадською думкою через месенджери.

Отже, інформаційні потоки в месенджерах мають великий потенціал як для конструктивного використання, так і для маніпуляцій. У контексті війни, де кожен канал комунікації стає частиною інформаційної боротьби, важливо враховувати ці загрози при розробці стратегій кібербезпеки та боротьби з дезінформацією.

1.4 Шкідлива інформація в месенджерах: види та вплив на кібербезпеку

Шкідлива інформація, що поширюється через месенджери, є серйозною загрозою для кібербезпеки, особливо в умовах війни. Вона може мати різні форми і впливати на користувачів на кількох рівнях, включаючи психологічний, соціальний та технічний. У випадку з російсько-українським конфліктом, месенджери стали важливим інструментом для дезінформаційних кампаній, пропаганди та організації підривної діяльності. Однак шкідлива інформація може мати й інші форми, такі як шкідливі програми, фішинг або зловмисні файли, які використовуються для здійснення кібератак.

1.4.1 Дезінформація та фейки

Один з найбільш поширених типів шкідливої інформації в месенджерах — це дезінформація, яка включає фальшиві новини, маніпуляції фактами та перекручування інформації. Під час війни дезінформація стає потужним інструментом для впливу на громадську думку як в Україні, так і за її межами. Росія активно використовує месенджери для поширення неправдивої інформації про воєнні події, ставлення до агресії з боку світових лідерів, та для формування спотвореного образу України у світі.

Наприклад, фейки про поразки українських військ, фальшиві новини про

переговори чи капітуляцію — все це може бути розповсюджене через канали месенджерів, де інформація здебільшого не перевіряється і поширюється швидко через репости та групи. Ці повідомлення можуть призвести до паніки, зневіри серед населення, або навіть до деморалізації військових. Окрім того, вони можуть використатися для створення хаосу, дестабілізації ситуації, викликаної підвищеним рівнем невизначеності та недовіри.

Шкідлива інформація у вигляді фейків також може мати цілеспрямовану мету підривати міжнародну підтримку України, зокрема, маніпулюючи міжнародними партнерами та організаціями. Це може включати фальшиві звинувачення у порушенні прав людини, спотворення інформації щодо гуманітарної ситуації або звинувачення в порушенні міжнародних законів, що може вплинути на політичні процеси та вплинути на надання допомоги.

1.4.2 Шкідливі програми та кібератаки

Іншим видом шкідливої інформації є шкідливі програми, такі як віруси, трояни, руткіти та шпигунські програми, які можуть бути розповсюджені через месенджери [8]. Вони можуть потрапляти до користувачів як прикріплені файли або посилання на вебсайти, що містять шкідливий код. Такі програми здатні викрадати персональні дані, зламувати облікові записи, а також здійснювати атаки на критичну інфраструктуру. В умовах війни такі атаки можуть бути використані для компрометації військових, урядових чи приватних систем, що сприяє посиленню ворожої кіберстратегії.

Особливо небезпечними є кібератаки, спрямовані на знищення або маніпуляцію даними в системах, які забезпечують життєдіяльність країни — енергетичних мережах, банківських системах, урядових структурах. Месенджери можуть стати каналом для доставки шкідливого програмного забезпечення внаслідок довіри користувачів до інформації, отриманої через ці платформи.

Шкідливі файли можуть маскуватися під легітимні документи або інструменти, що створює додаткову складність для виявлення та блокування таких загроз.

1.4.3 Фішинг та соціальна інженерія

Ще однією серйозною загрозою є фішинг — метод обману користувачів з метою отримання конфіденційних даних, таких як паролі, номери кредитних карток, доступ до облікових записів. Фішингові повідомлення можуть бути надіслані через месенджери, де зловмисники представляються авторитетними особами, організаціями або навіть військовими структурами, щоб змусити користувачів надати особисті дані або перейти на підроблені вебсайти, що виглядають як офіційні ресурси. Окрім того, фішингові атаки можуть здійснюватися через публічні джерела інформації, такі як публікації у соціальних мережах, форумах або навіть на фіктивних новинних сайтах, де шкідливі посилання поширюються під виглядом перевіреної інформації або офіційних повідомлень [9]. Це дозволяє зловмисникам охоплювати ширшу аудиторію, спонукаючи користувачів до небезпечних дій під впливом довіри до джерела.

Соціальна інженерія, яка є частиною фішингових атак, також використовує маніпуляцію емоціями та психологічним тиском на жертву. Наприклад, зловмисники можуть створювати фальшиві екстрені ситуації, що спонукають користувачів до негайного реагування, наприклад, надати доступ до особистих акаунтів або перейти за шкідливими посиланнями. В умовах війни такі атаки можуть мати надзвичайно серйозні наслідки, спричиняючи втрату важливих державних або військових даних.

1.4.4 Вплив шкідливої інформації на кібербезпеку

Шкідлива інформація в месенджерах здатна значно погіршити рівень кібербезпеки, як на рівні окремих користувачів, так і на рівні державних та корпоративних структур. Вона може призводити до порушення конфіденційності, цілісності та доступності даних, створюючи загрози для національної безпеки та стабільності. Тому в умовах гібридної війни важливо розробляти та впроваджувати стратегії для моніторингу, виявлення і нейтралізації шкідливої інформації в месенджерах, а також проводити інформаційну гігієну серед населення, щоб мінімізувати вплив таких загроз.

Висновки до розділу 1

Розділ, присвячений аналізу ризиків і викликів кібербезпеки, пов’язаних із поширенням інформації через месенджери, розкриває їхню багатогранну роль у сучасному інформаційному просторі. Месенджери, які стали невід’ємною частиною комунікації, виступають платформами для миттєвого обміну інформацією, значно спрощуючи комунікацію, але водночас створюючи потенційні загрози. Їхнє використання варіюється від побутового до стратегічного, що підвищує їх значимість, але й посилює виклики у сфері кібербезпеки

Розгляд месенджерів як соціальних графів демонструє складність їхньої структури та динамічність взаємодій між користувачами, групами та каналами. Такий підхід дозволяє виявляти ключові вузли та вивчати поширення контенту, включаючи шкідливий. В умовах гібридної війни інформаційні потоки у месенджерах набувають вирішального значення, стаючи інструментами маніпуляції, дезінформації та координації кібератак. Застосування шкідливих програм, фішингових атак і розповсюдження дезінформації демонструє, наскільки вразливим є цей канал комунікації, особливо у періоди конфліктів

Особливу увагу привертає вплив шкідливої інформації на громадську думку, психологічний стан користувачів та безпеку держави. Військові конфлікти надають месенджерам додаткову роль у поширенні пропаганди та створенні хибної реальності. Тому ефективна протидія таким загрозам вимагає комплексного підходу, що включає технічні, організаційні та правові заходи. Це не лише підвищить рівень захищеності інформаційного простору, а й сприятиме розвитку довіри серед користувачів і суспільства в цілому.

2 АНАЛІЗ ПІДХОДІВ ДО ВИЯВЛЕННЯ ДЖЕРЕЛ ТА ПОТОКІВ ІНФОРМАЦІЇ У МЕСЕНДЖЕРАХ

2.1 Методи аналізу інформаційних потоків у месенджерах

Аналіз інформаційних потоків у месенджерах є важливою складовою сучасних досліджень у сфері кібербезпеки, оскільки він дозволяє виявляти потенційні загрози, відстежувати поширення дезінформації та забезпечувати безпеку інформаційного простору. Месенджери стали основними засобами комунікації, що робить їх актуальними об'єктами для вивчення з точки зору аналізу інформаційних потоків. Для цього застосовуються різноманітні методи та підходи, які дозволяють моделювати та візуалізувати процеси поширення інформації.

Першим і основним підходом є **аналіз змісту повідомлень**. За допомогою текстового аналізу можна визначити основні теми та ключові слова, які розповсюджуються в межах каналів і груп. Цей метод дозволяє не тільки ідентифікувати популярні теми, але й оцінити емоційне забарвлення повідомлень, що може бути корисним для виявлення маніпуляцій та пропаганди.

Другим важливим методом є **аналіз мережі зв'язків** між учасниками комунікації. Месенджери часто використовують групові чати або канали, де учасники можуть обмінюватися повідомленнями. За допомогою побудови графів зв'язків між користувачами або каналами можна відстежити, як інформація передається з одного користувача до іншого або як повідомлення розповсюджуються в межах великої мережі. Для цього використовуються методи теорії графів, які дозволяють визначити важливість конкретних вузлів (користувачів чи каналів), а також маршрути, через які проходять повідомлення.

Метод **аналізу метаданих** також є важливим аспектом у вивченні інформаційних потоків. Месенджери зберігають метадані, такі як час

відправлення повідомлення, географічне місцезнаходження користувача, його IP-адреса, що можуть бути використані для виявлення закономірностей поширення інформації. Наприклад, якщо певні повідомлення регулярно з'являються в різних каналах у певний час, це може вказувати на централізовану схему розповсюдження інформації.

Окрім цього, важливим є **виявлення і аналіз шаблонів репостів**. У месенджерах часто спостерігається явище репостів, коли інформація передається від одного каналу або користувача до іншого. Виявлення та аналіз цих репостів може допомогти відстежити корінь інформаційного потоку та виявити потенційні джерела дезінформації чи маніпуляцій.

Не менш важливим є **аналіз поведінки користувачів** в месенджерах, який дозволяє виявити автоматизовані або скоординовані дії (наприклад, боти, які масово поширюють певну інформацію). Виявлення таких аномалій потребує застосування технік аналізу великих даних і побудови профілів користувачів.

Таким чином, методи аналізу інформаційних потоків у месенджерах включають різноманітні підходи: від текстового аналізу до глибинного машинного навчання та вивчення мережі зв'язків між учасниками комунікації. Комбінація цих методів дає змогу ефективно відслідковувати та моделювати поширення інформації в месенджерах, що має важливе значення для забезпечення кібербезпеки в інформаційному просторі.

2.2 Використання API для аналізу відкритих даних у месенджерах

Одним із важливих аспектів дослідження інформаційних потоків у месенджерах є ефективний доступ до відкритих даних, що дозволяє здійснювати глибокий аналіз контенту, поведінки користувачів та динаміки поширення інформації. Для цього використовуються програмні інтерфейси (API), які забезпечують можливість інтеграції з месенджерами, надаючи доступ до

різноманітних даних, таких як повідомлення, метадані, користувацькі профілі та інші параметри, що можуть бути корисними для аналізу.

Одним із найпопулярніших інструментів для аналізу відкритих даних у месенджерах є **API Telegram** [10]. Telegram є одним з найбільш поширених месенджерів, і його API надає широкі можливості для збору даних, що дозволяє здійснювати як базовий, так і глибокий аналіз потоків інформації. Використовуючи Telegram API, можна отримати доступ до повідомлень у каналах, групах та чатах, а також отримати метадані, такі як час публікації, ідентифікатори користувачів, інформацію про перегляди та взаємодії з повідомленнями. Це дозволяє вивчати не лише вміст повідомлень, але й динаміку їх поширення, виявляючи можливі шаблони або аномалії.

Іншим популярним інструментом є **API Discord** [11], який дозволяє отримувати інформацію про сервери, канали, повідомлення та учасників. Використовуючи API Discord, можна аналізувати не тільки текстову інформацію, але й інші аспекти комунікації, наприклад, голосові повідомлення, реакції на повідомлення та інші інтерфейсні елементи [16]. Це дає змогу створювати багатогранні моделі для вивчення взаємодії користувачів у певних спільнотах і з'ясувати, які канали та групи є основними центрами поширення інформації.

Особливу увагу слід звернути на **аналітику груп та каналів**, яку можна здійснювати за допомогою API. Багато месенджерів надають функції, що дозволяють отримувати статистику за групами та каналами, зокрема кількість підписників, рівень активності, частоту публікацій та інші показники, що дозволяє визначити, які канали є найбільш впливовими в інформаційному просторі. Вивчення таких статистичних даних може допомогти виявити центри, через які проходять основні потоки інформації.

Одним з важливих аспектів використання API є **автоматизація збору даних**. Багато месенджерів дозволяють створювати боти, які можуть автоматично збирати повідомлення з різних джерел, фільтруючи за необхідними критеріями, наприклад, за хештегами, ключовими словами або часом публікації. Це дозволяє

зібрати великі обсяги даних для подальшого аналізу та обробки. Завдяки можливості налаштування ботів на збори інформації в реальному часі, можна відстежувати, як інформація поширюється в межах певних груп чи каналів.

Але використання API для збору даних у месенджерах має певні обмеження. Одним із таких обмежень є **конфіденційність та анонімність користувачів**, оскільки доступ до приватних даних вимагає додаткових дозволів або спеціальних погоджень з користувачами. Більшість месенджерів мають обмеження на кількість запитів, що можна зробити за певний період часу, що може впливати на ефективність збору даних. Крім того, деякі месенджери обмежують доступ до даних третім особам через API, щоб захистити конфіденційність своїх користувачів.

Тим не менш, правильне використання API дозволяє значно спростити процес збору відкритих даних і зробити його більш ефективним. Зокрема, можна створювати аналітичні панелі, що дозволяють в реальному часі відстежувати поширення конкретної інформації, визначати ключові точки поширення та вивчати поведінку користувачів. Крім того, зібрані дані можна обробляти за допомогою різних методів аналізу, таких як текстовий аналіз, класифікація повідомлень або виведення статистики щодо популярних тем та каналів.

Таким чином, використання API є ефективним інструментом для збору та аналізу відкритих даних у месенджерах, що дає змогу отримувати важливу інформацію для подальшого вивчення інформаційних потоків і поширення інформації в контексті кібербезпеки.

2.3 Використання графових БД для аналізу потоків інформації

Графові бази даних набули великої популярності в останні роки завдяки своїй здатності ефективно обробляти складні взаємозв'язки між об'єктами та відображати ці зв'язки у вигляді графів [12]. У контексті аналізу потоків

інформації в месенджерах, графові бази даних є потужним інструментом для моделювання, збереження та аналізу інформаційних мереж, що утворюються під час поширення повідомлень між користувачами, каналами та групами. Графи дозволяють наочно відображати як інформація поширюється, а також допомагають виявляти ключові вузли та важливі зв'язки в інформаційних потоках.

Однією з основних переваг **графових БД** є їх здатність ефективно зберігати та маніпулювати даними про зв'язки між об'єктами. У випадку з месенджерами, об'єктами можуть бути користувачі, канали, групи, повідомлення або навіть хештеги, а зв'язки між ними можуть відображати різноманітні відносини, такі як репости, коментарі, взаємодії або передавання повідомлень. Графові структури дозволяють безпосередньо відображати мережу зв'язків між об'єктами, що є природним способом зберігання інформації про її поширення.

Один з найбільш поширених інструментів для роботи з графовими базами даних — **Neo4j**, який дозволяє створювати графи з вузлів і зв'язків, що містять різноманітні атрибути[13]. У контексті аналізу потоків інформації в месенджерах, вузли можуть бути представлені як канали, користувачі, повідомлення або інші елементи комунікації, а зв'язки між ними можуть позначати такі відносини, як "направлення повідомлення", "репост", "підписка на канал", тощо. З використанням таких графових БД можна побудувати моделі, що відображають реальні потоки інформації, допомагаючи ідентифікувати основні шляхи поширення і виявляти ключові точки або вузли, які відіграють важливу роль у дистрибуції контенту.

Для прикладу, коли користувач публікує повідомлення в каналі або групі, це повідомлення може бути репощене іншими користувачами або каналами, що створює зв'язки між різними вузлами в графі. Аналіз таких зв'язків дозволяє визначити, які канали або групи є найбільш впливовими у певній мережі, а також відстежити шляхи, якими інформація поширюється від одного вузла до іншого. Це дозволяє не тільки виявити потоки інформації, але й аналізувати, як інформація змінюється або адаптується на різних етапах свого поширення.

Графові бази даних особливо ефективні для **виявлення спільнот або кіл користувачів**, які мають високий рівень взаємодії між собою. Це дозволяє виявити інформаційні "кластери", де певна група користувачів активно обмінюється інформацією, і відслідковувати, як ці кластери можуть впливати на більші мережі. Такі спільноти можуть бути важливими при аналізі дезінформації або маніпуляцій, оскільки виявлення аномальних або занадто зв'язних мереж може сигналізувати про координацію певних груп для поширення фальшивих новин або пропаганди.

Аналіз центральності є ще однією корисною технікою при використанні графових БД для аналізу потоків інформації. За допомогою метрик, таких як **степенева центральність**, **центральність міжвузлових зв'язків** або **центральність близькості**, можна визначити, які вузли (користувачі або канали) є найважливішими для поширення інформації в межах мережі. Наприклад, канал або користувач, що має високий рівень центральності, може мати більший вплив на те, як і куди інформація буде поширюватися в месенджерах. Це особливо важливо для виявлення джерел дезінформації або маніпуляцій, коли потрібно з'ясувати, хто є основним ініціатором поширення певної інформації.

Завдяки **застосуванню алгоритмів машинного навчання** до графових БД можна ще більше полегшити виявлення аномалій і вивчення патернів поширення інформації. Наприклад, алгоритми класифікації можуть бути використані для виявлення каналів, які активно поширюють фальшиві новини.

Отже, використання графових БД для аналізу потоків інформації в месенджерах є потужним інструментом для вивчення складних мережевих зв'язків і динаміки поширення інформації. Це дозволяє не тільки виявляти важливі вузли і шляхи поширення інформації, але й створювати детальні моделі, що допомагають виявити аномалії, дезінформацію або зловмисні мережі, що мають значний вплив на інформаційний простір

2.4 Використання реляційних БД для аналізу відкритих даних у месенджерах

Реляційні бази даних є традиційним інструментом для зберігання та обробки даних, що організовані в таблиці з чіткими зв'язками між ними. Вони добре підходять для задач, де необхідно забезпечити збереження структурованої інформації, такої як дані про дописи в каналах месенджера. Використання реляційних БД для аналізу відкритих даних дозволяє ефективно зберігати, управляти та аналізувати великі обсяги інформації про повідомлення, що публікуються в месенджерах.

У випадку з дописами в каналах месенджера можна використовувати одну таблицю для зберігання даних про кожен допис. Ця таблиця може містити кілька важливих атрибутів, таких як текст повідомлення, час публікації, ідентифікатор каналу, ідентифікатор користувача, що опублікував повідомлення, а також інші метадані, такі як кількість переглядів чи взаємодій. Завдяки реляційній структурі бази даних можна зберігати ці дані у вигляді записів, кожен з яких містить відомості про окремий допис.

Однією з основних переваг використання реляційних БД є **забезпечення цілісності та організованості даних**. У випадку, коли необхідно проводити аналітичні дослідження, можна легко отримати зведену інформацію, яка дозволяє відслідковувати активність у каналах, виявляти тренди та визначати найбільш впливові дописи чи канали. Наприклад, можна здійснювати аналіз того, як часто публікуються дописи в певних каналах, який контент є найбільш популярним серед підписників, а також проводити порівняння активності між різними каналами.

Також, **реляційні бази даних дозволяють здійснювати потужний фільтр даних** за різними критеріями. Наприклад, можна створити запити для отримання дописів, що містять певні ключові слова, або для виявлення дописів, що набрали

найбільше взаємодій за певний проміжок часу. Такий підхід дозволяє більш гнучко підходити до аналізу, відбираючи лише ті дані, які є найбільш релевантними для конкретних досліджень.

Незважаючи на те, що реляційні бази даних є досить ефективними для зберігання і обробки структурованої інформації, вони мають певні обмеження в контексті роботи з великими обсягами неструктурованих або змінних даних. Наприклад, коли йдеться про **взаємодії користувачів із повідомленнями**, такі як коментарі, лайки або репости, реляційна база може бути не такою гнучкою, як інші типи БД, наприклад, графові. У таких випадках можуть виникнути труднощі з підтримкою складних зв'язків між різними типами даних.

Однак для задач, що вимагають **збереження історії публікацій** і основних характеристик дописів (наприклад, текст, час, ідентифікатор каналу, ідентифікатор автора), реляційні бази даних можуть бути дуже зручними. Вони дають змогу організувати дані так, щоб можна було здійснювати різні форми аналізу, наприклад, вимірювання **активності каналу** або **темпів публікації дописів**.

При використанні реляційних БД також можна забезпечити **ефективну автоматизацію збору та зберігання даних**. Створення автоматичних процесів, які регулярно збирають дані про нові дописи в каналах месенджера та записують їх у таблицю, дозволяє вести моніторинг інформаційних потоків у реальному часі. Ці процеси можуть бути налаштовані на регулярний збір даних або на запити, що робить їх надзвичайно гнучкими для різних сценаріїв.

Ще однією перевагою реляційних БД є **інтеграція з іншими системами для аналізу даних**. Наприклад, за допомогою реляційної бази даних можна зберігати дані для подальшого використання в аналітичних інструментах, таких як Python, для статистичного аналізу або побудови моделей машинного навчання. Це дозволяє проводити детальний аналіз потоків інформації, оцінюючи динаміку публікацій, вивчаючи популярність певних тем та прогнозуючи тенденції розвитку інформаційних потоків.

Отже, реляційні бази даних є потужним інструментом для зберігання та аналізу даних про дописи в каналах месенджерів. Їх використання дозволяє ефективно організовувати, зберігати і обробляти структуровану інформацію, що є важливою частиною дослідження поширення інформації в інформаційному просторі. Хоча реляційні бази можуть мати певні обмеження щодо роботи зі складними взаємозв'язками, їх здатність до організації та управління даними робить їх важливим інструментом для аналізу відкритих даних у месенджерах.

2.5 Автоматизація збору відкритих даних у задачах кібербезпеки

Автоматизація збору відкритих даних у месенджерах є важливою складовою сучасних досліджень у галузі кібербезпеки, оскільки забезпечує ефективний доступ до великої кількості інформації для подальшого аналізу [15]. У випадку дослідження потоків інформації через месенджери [14], зокрема через канали, автоматизований підхід дозволяє отримувати дані про дописи швидко, масштабовано та з мінімальними людськими витратами. Це особливо актуально в умовах, коли обсяги інформації постійно зростають, а динаміка її поширення потребує своєчасного моніторингу.

Автоматизація збору даних базується на використанні API месенджера, що надає доступ до публічних даних, таких як текст дописів, дата та час публікації, ідентифікатор або назва каналу, кількість переглядів, коментарів та інша метаінформація. Інтеграція з API дозволяє створювати автоматизовані системи збору даних, які можуть працювати за кількома сценаріями:

1. **Регулярний збір даних.** Система може запускатися з певною періодичністю (наприклад, щогодини або щодня), щоб оновлювати інформацію про нові дописи у визначених каналах.
2. **Моніторинг у реальному часі.** У цьому випадку система працює безперервно, реагуючи на появу нових даних у режимі реального часу.

3. **Запитуваний збір даних.** Автоматизований процес активується за запитом дослідника або аналітика для збору даних з конкретних каналів або за певними критеріями.

Окрім збору, автоматизація також забезпечує **попереднє опрацювання даних**, що включає фільтрацію, очищення та стандартизацію інформації. Наприклад, текст дописів може бути очищений від зайвих символів, емодзі або тегів, що не є важливими для аналізу. Метадані можуть бути перетворені у зручний для подальшого зберігання формат, наприклад, датою публікації у стандартному часовому форматі. Такі підходи дозволяють значно знизити обсяги ручної роботи, мінімізуючи ризик помилок і підвищуючи точність результатів.

Важливою складовою автоматизації є **масштабованість**, що дозволяє адаптувати системи збору до роботи з великими обсягами даних. Наприклад, коли необхідно зібрати інформацію з сотень або навіть тисяч каналів одночасно, система може бути розгорнута у кластерному середовищі або з використанням хмарних технологій. Це дає змогу забезпечити високу продуктивність та уникнути перевантажень під час інтенсивного збору даних.

Автоматизація також відіграє ключову роль у вирішенні завдань **моніторингу кіберзагроз**. Завдяки постійному доступу до відкритих даних, системи можуть використовуватися для виявлення підозрілих активностей у месенджерах, таких як масове поширення фальшивих новин, скоординовані кампанії з дезінформації або активність груп, що займаються незаконними діями. Такі дані можуть бути оперативно передані аналітичним системам для подальшого аналізу та виявлення загроз у кіберпросторі.

Іншим аспектом автоматизації є **забезпечення юридичної відповідності**. Оскільки збір даних з месенджерів повинен відповідати законодавству про захист даних і конфіденційність, автоматизовані системи можуть бути налаштовані таким чином, щоб збирати виключно публічну інформацію, доступну через API. Це дозволяє уникнути порушень прав користувачів та забезпечити етичний підхід до

роботи з даними.

Крім того, автоматизовані системи збору даних можуть бути інтегровані з інструментами для подальшого аналізу, такими як реляційні або графові бази даних, інструменти машинного навчання або аналітичні панелі. Це дозволяє забезпечити безперервний потік від збору до аналізу, що значно пришвидшує отримання інсайтів та підвищує ефективність досліджень.

Отже, автоматизація збору відкритих даних у месенджерах є критично важливим етапом для аналізу потоків інформації в задачах кібербезпеки. Вона забезпечує ефективний, швидкий та масштабований доступ до великих обсягів інформації, дозволяючи оперативно реагувати на загрози в інформаційному просторі. Завдяки інтеграції з сучасними аналітичними інструментами автоматизація відкриває широкі можливості для дослідження та захисту інформаційного середовища від потенційних загроз.

Висновки до розділу 2

У межах розділу було здійснено аналіз сучасних підходів до виявлення джерел та потоків інформації у месенджерах, що є важливою складовою досліджень у галузі кібербезпеки інформаційного простору.

По-перше, розглянуто **методи аналізу інформаційних потоків**, які базуються на принципах виявлення джерел поширення, шляхів передачі даних та вивчення динаміки їх розповсюдження. Було показано, що ефективність цих методів залежить від здатності враховувати особливості роботи месенджерів, зокрема їхню децентралізовану природу та різноманітність типів контенту.

По-друге, досліджено можливості **використання API месенджерів** для збору відкритих даних. API забезпечують доступ до публічної інформації, такої як дописи в каналах, метадані про взаємодії користувачів та інші дані, що є ключовими для аналізу інформаційних потоків. Було виявлено, що використання

API дозволяє автоматизувати процес збору даних і забезпечити їх зручне інтегрування у бази даних.

По-третє, розглянуто **графові бази даних**, які є потужним інструментом для візуалізації та аналізу взаємозв'язків між об'єктами. Виявлено, що вони особливо ефективні для аналізу складних мережевих структур, таких як канали та їх взаємодії у месенджерах. За допомогою графових БД можна будувати моделі інформаційних потоків, виявляти ключові вузли та оцінювати вплив окремих каналів на загальну мережу.

Також проаналізовано **реляційні бази даних**, які залишаються актуальними для задач, пов'язаних із збереженням та обробкою структурованих даних про дописи. Реляційні бази забезпечують простоту доступу до даних і можливість проведення різноманітних фільтраційних та статистичних операцій, що робить їх зручними для стандартних аналітичних завдань.

Окрему увагу приділено **автоматизації збору відкритих даних**, яка є ключовим елементом у дослідженнях кібербезпеки. Автоматизовані процеси дозволяють зменшити людські витрати, забезпечити високу швидкість збору даних і працювати з великими обсягами інформації. Було наголошено на важливості дотримання етичних та юридичних норм під час автоматизації, зокрема збору лише публічної інформації.

На основі проведеного аналізу можна зробити висновок, що ефективне виявлення джерел і потоків інформації у месенджерах потребує комплексного підходу, що поєднує автоматизовані методи збору даних, використання передових БД (графових і реляційних) та адаптивні аналітичні методи. Такий підхід дозволяє не лише забезпечити глибоке розуміння динаміки поширення інформації, а й сприяти підвищенню рівня кібербезпеки інформаційного простору

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ ВИЯВЛЕННЯ ПОТОКІВ ПОШИРЕННЯ ІНФОРМАЦІЇ

3.1 Вибір середовища для збору даних

Процес вибору середовища для збору даних є важливим етапом при розробці системи для виявлення потоків поширення інформації через месенджери. Вибір середовища, в якому буде здійснюватися збір даних, потребує аналізу різних критеріїв, таких як технічні можливості платформи, доступність даних, а також соціальні аспекти, які впливають на ефективність збору та аналізу інформації.

Перш за все, одним з основних технічних критеріїв є доступність API для інтеграції та збору даних. Платформи, що мають добре документовані та потужні API, дозволяють здійснювати більш ефективний збір даних, що включає як базову інформацію про канали та групи, так і більш складні взаємодії, такі як отримання текстів повідомлень, метаданих або навіть взаємодію з користувачами. Важливо, щоб API дозволяло отримувати дані у реальному часі або з певною затримкою, оскільки це впливає на можливість оперативного відстеження поширення інформації. Більшість популярних месенджерів, таких як Telegram, WhatsApp, Facebook Messenger, пропонують різні рівні доступу до даних, що необхідно враховувати при виборі платформи для збору.

Наступним критерієм є соціальні аспекти. Вибір платформи для збору даних залежить також від її популярності серед користувачів, а також від того, яка частина цієї аудиторії активно обмінюється інформацією, що має значення для кібербезпеки. Для досліджень, пов'язаних з аналізом потоків інформації, важливо, щоб на платформі були присутні великі групи або канали, де активно відбувається поширення даних, що може включати як новини, так і потенційно шкідливу або маніпулятивну інформацію. Платформи, на яких користувачі активно взаємодіють, обмінюються новинами або активно обговорюють різні питання, є більш

підходящими для таких досліджень.

Не менш важливими є юридичні та етичні критерії. Важливо, щоб обрана платформа забезпечувала належну рівновагу між доступністю даних та захистом приватності користувачів. Це має значення для збору даних, зокрема щодо того, чи дозволяє платформа збір публічної інформації без порушення прав користувачів. З точки зору кібербезпеки, важливо також врахувати, які заходи безпеки застосовуються для захисту даних користувачів і як це може вплинути на збір та аналіз інформації.

Іншим аспектом є технічна сумісність обраної платформи з іншими інструментами та технологіями, які використовуються для збору та аналізу даних. Наприклад, якщо для аналізу потоків інформації планується використовувати графову базу даних Neo4j, то важливо, щоб API месенджера забезпечувало можливість легко інтегрувати дані з цією технологією, а також підтримувало необхідний формат даних. Крім того, з технічної точки зору, важливо враховувати обмеження по кількості запитів, швидкості отримання даних та наявності спеціальних інструментів для аналізу даних на рівні API.

Загалом, для успішної реалізації системи збору даних важливо не тільки технічні можливості платформи, а й соціальні аспекти, такі як популярність серед користувачів та типи даних, що можуть бути доступні для аналізу. При цьому обов'язково слід враховувати етичні та юридичні аспекти, щоб уникнути порушень прав користувачів.

3.1.1 Порівняння доступності даних у популярних соціальних мережах

У сучасному інформаційному середовищі існує безліч соціальних мереж, які використовуються як для особистих комунікацій, так і для обміну інформацією в професійному контексті. Кожна з платформ має свої особливості у доступності даних, технічні можливості API та різні етичні та соціальні аспекти, що необхідно

враховувати при виборі середовища для збору даних. Нижче розглянуто найбільш популярні соціальні мережі, зокрема в Україні та світі.

Таблиця 3.1 - Доступність даних у популярних соціальних мережах

Платформа	Доступність даних	Соціальні критерії	Технічні критерії	Етичні критерії
Facebook	API Facebook Graph API дозволяє отримувати публічні дані зі сторінок, груп, та обговорень; приватні дані доступні через OAuth.	Найбільш популярна платформа для обміну інформацією, але складно збирати дані через різноманітність акаунтів і груп.	Graph API добре документоване, але має обмеження на частоту запитів, що ускладнює збір у реальному часі.	Суворі правила конфіденційності, дані доступні лише з публічних сторінок і груп.
Twitter	API дозволяє отримувати публічні твіти, дані про користувачів, хештеги, але обмежує доступ до приватних повідомлень.	Важлива платформа для новин та громадських дискусій, широко використовується медіа, політиками та організаціями.	API підтримує пошук за хештегами, фільтрацію, але обмежує кількість запитів для великих обсягів даних.	Великий обсяг публічної інформації доступний для досліджень; важливо враховувати контекст повідомлень для збереження етики.
Telegram	API дозволяє отримувати дані з публічних каналів, груп, повідомлень через Bot API; приватні дані потребують авторизації.	Популярний серед молоді, активістів і політиків; багато публічних каналів для аналізу інформаційних потоків.	Добре документоване API, інтегрується з графовими базами даних; ефективний збір у реальному часі.	Велика кількість відкритих даних; важливо уникати порушення прав користувачів у контексті приватної інформації.
Instagram	API дозволяє доступ до даних публічних акаунтів і постів, але обмежено через політики захисту даних Facebook.	Використовується для маркетингу, брендингу; важлива для аналізу інформаційних потоків у медіа.	API має обмеження на кількість запитів; можуть бути потрібні додаткові стратегії для збору великих обсягів даних.	Обмежений доступ до даних, дотримуються суворих політик конфіденційності.

Кінець таблиці 3.1

Viber	Немає публічного API; є API для ботів, але доступ до приватних чатів і груп обмежений.	Популярний месенджер в Україні, використовується для особистих комунікацій; обмежені публічні групи для аналізу.	API для ботів обмежене доступом до даних; відсутність відкритих даних ускладнює автоматизацію збору.	Платформа для особистих комунікацій; доступ до даних без згоди користувачів є неприпустимим.
WhatsApp	Немає відкритого API для збору даних з приватних чатів[17]; лише обмежений API для бізнес-інтеграцій.	Популярний серед широкого кола користувачів, але не підходить для збору публічної інформації.	API обмежений бізнес-інтеграціями; немає доступу до історії повідомлень.	Дані захищені наскрізним шифруванням; доступ без згоди користувачів є неприпустимим.
Discord	API дозволяє збір даних з публічних серверів; для приватних потрібен доступ до акаунтів чи адміністраторські права.	Популярний серед молоді, геймерів та тематичних спільнот; придатний для аналізу потоків у публічних чатах.	API добре документоване, дозволяє збір даних у реальному часі; зручний для аналізу інформаційних потоків.	Публічна інформація доступна; важливо дотримуватись політики конфіденційності для приватних чатів чи серверів.

У підсумку, кожен з цих месенджерів має свої особливості щодо доступності даних, технічних можливостей API та етичних аспектів. **Viber** та **WhatsApp** мають обмежений доступ до даних через їхню орієнтацію на захист приватності користувачів, в той час як **Telegram** надає більш широкі можливості для збору публічної інформації, особливо в тематичних групах і каналах, що робить його більш привабливим для досліджень потоків інформації.

3.1.2 Порівняння можливостей API кожної платформи

Порівнюючи можливості API для збору даних з різних платформ, важливо звернути увагу на технічні можливості інтеграції, доступ до різноманітних типів даних, обмеження та зручність використання. Нижче наведено порівняння основних можливостей API для Viber, WhatsApp, Discord, Telegram та Facebook.

Таблиця 3.2 - Можливість API кожної платформи

Платформа	**Типи даних**	**Можливості інтеграції**	**Обмеження**	**Документація та зручність використання**
Viber	Базові дані про взаємодію з ботами, повідомлення в чатах з ботами.	Створення ботів, автоматизація комунікації. Особисті чати або групи недоступні без згоди.	Відсутність доступу до публічних даних. Обмежений функціонал.	Документація чітка, але API має обмежений функціонал для масового збору даних.
WhatsApp	Дані бізнес-комунікацій через WhatsApp Business API, повідомлення клієнтів.	Інтеграція для бізнесу, взаємодія через API в контексті бізнес-зв’язку.	Немає доступу до приватних чатів, історії або груп без згоди.	Документація орієнтована на бізнес, обмежена для дослідницьких потреб.
Discord	Текстові повідомлення, голосові дані, метадані про канали, інформація про ролі і користувачів.	Інтеграція ботів для збору повідомлень, перегляду публічних каналів, даних про учасників.	Приватні сервери/чати доступні лише з адміністраторськими правами.	Добре документоване API з багатьма прикладами та бібліотеками.
Telegram	Текстові повідомлення, медіафайли, метадані каналів і груп, дані про взаємодію користувачів.	Створення ботів для збору даних з публічних каналів/груп. Доступ до репостів, інформації про користувачів і час публікацій.	Приватні групи/чати доступні тільки за згодою.	Дуже добре документоване, з великою кількістю бібліотек для різних мов програмування.

Кінець таблиці 3.2

Facebook	Дані публічних сторінок, постів, коментарів, обмежені дані про користувачів (залежно від налаштувань конфіденційності).	Збір даних з публічних сторінок, груп через Graph API. Приватні повідомлення через авторизацію користувача (OAuth).	Доступ до особистих даних обмежений.	Потужне API, але складне для збору великих обсягів даних через політики доступу.
Instagram	Публічні дані про профілі, пости, коментарі, взаємодії, метадані з бізнес-акаунтів через Graph API.	Отримання публічної інформації профілів, аналіз взаємодії з постами через Instagram Graph API.	Обмеження доступу до приватних акаунтів та прямої комунікації.	Добре документована, але обмежена політиками конфіденційності для збору великих обсягів даних.
Twitter	Дані про твіти, ретвіти, лайки, взаємодії, хештеги, інформація про користувачів.	Доступ до публічних твітів, аналітика хештегів, взаємодій. Можливість отримання історії твітів через Academic API.	Доступ до приватних акаунтів можливий лише з авторизацією. Academic API має обмеження на обсяг і частоту запитів.	API є добре задокументованим, підтримує аналітику та є зручним для наукових досліджень, але має обмеження за частотою.

- **Telegram** і **Discord** надають найбільш відкритий доступ до даних через API для збору інформації з публічних каналів і серверів, що робить їх ідеальними для аналізу потоків інформації.
- **Viber** і **WhatsApp** мають обмежений доступ до даних, орієнтуючись на особисті комунікації, що ускладнює збору даних для аналізу.
- **Facebook** забезпечує обмежений доступ до публічних сторінок і груп через **Graph API**, але обмеження щодо приватних даних ставлять його в менш зручне становище для збору даних в задачах кібербезпеки.

Таким чином, найзручнішими для збору даних для аналізу потоків інформації є **Telegram** та **Discord**, завдяки відкритому доступу до публічних даних та зручним API.

3.1.3 Обґрунтування вибору Telegram як основного середовища.

Вибір Telegram для збору даних для аналізу потоків поширення інформації обґрунтований кількома важливими факторами, пов'язаними з технічними, соціальними та етичними аспектами. По-перше, Telegram надає відкритий доступ до публічних даних через API, що дозволяє збирати інформацію з каналів і груп без потреби отримання додаткових дозволів від користувачів. API підтримує велику кількість запитів, що дає можливість ефективно працювати з повідомленнями та метаданими, такими як час публікації, типи медіа та географічне положення користувачів.

Telegram є однією з найбільш популярних платформ для комунікації в Україні та світі. Публічні канали стали важливим інструментом для організацій, медіа та політичних рухів, що активно використовують платформу для донесення інформації до широкої аудиторії. Цей аспект робить Telegram ідеальним середовищем для вивчення поширення інформації, оскільки на платформі регулярно публікуються новини та виникають явища, що можна проаналізувати з точки зору інформаційної безпеки.

З етичної точки зору, Telegram дозволяє зберігати анонімність користувачів, оскільки не вимагає особистої інформації для реєстрації. Для збору даних з публічних каналів це не порушує конфіденційності, оскільки інформація є відкритою для всіх користувачів.

Telegram має переваги порівняно з іншими платформами, такими як Discord чи Viber, завдяки великій кількості публічних каналів і розвиненій інфраструктурі для інтеграції з різними інструментами. Це дозволяє розробляти ефективні системи збору та аналізу даних, що робить платформу зручною для досліджень інформаційних потоків у кібербезпеці.

Таким чином, Telegram є одним з найбільш підходящих інструментів для збору даних щодо поширення інформації завдяки відкритому доступу до

публічних даних, технічним можливостям для автоматизації збору та етичній відповідності вимогам конфіденційності.

3.2 Збір даних з використанням Telegram API

Збір даних за допомогою Telegram API є ключовим етапом дослідження, оскільки від його ефективності залежить якість аналізу інформаційних потоків. Telegram API забезпечує автоматизований доступ до даних публічних каналів, груп, приватних чатів, повідомлень, файлів, медіа та метаданих.

Telegram API включає два основних інструменти: Bot API та MTProto API. Bot API використовується для створення ботів, що взаємодіють із користувачами, збирають повідомлення, метадані та виконують автоматизовані функції. MTProto API є більш потужним і надає прямий доступ до серверів Telegram, дозволяючи працювати з групами, каналами та іншими ресурсами, такими як вбудовані медіа або боти.

Для доступу до API потрібне налаштування токенів і прав доступу до даних, включаючи тексти повідомлень, типи медіа, дати публікації, авторів і структуру груп. Це дає змогу автоматизувати збір даних у режимі реального часу. Однак існують обмеження, такі як ліміти на кількість запитів, що вимагають впровадження механізмів обробки помилок і повторних запитів.

Отримані дані потребують фільтрації та обробки для підготовки до аналізу. Наприклад, необхідно відсіяти зайві медіа-файли, нормалізувати текст та врахувати специфіку форматів даних Telegram. Важливо також забезпечити конфіденційність і дотримуватися етичних норм, використовуючи лише публічно доступну інформацію.

Загалом Telegram API дозволяє створити гнучку систему збору даних, що забезпечує аналіз і моделювання інформаційних потоків.

3.2.1 Огляд Telegram API та можливостей для збору даних

Для інтеграції з Telegram API існує кілька популярних бібліотек, що спрощують процес збору даних і дозволяють ефективно працювати з платформою. Оскільки Telegram API надає доступ до багатьох функцій месенджера, таких як отримання повідомлень, взаємодія з каналами, групами та ботами, вибір правильної бібліотеки є важливим кроком при розробці системи збору даних. Одними з найпоширеніших рішень є **Telethon**, **Pyrogram** та **Aiogram**, кожна з яких має свої особливості та переваги.

Короткий опис бібліотек:

1. **Telethon**: Потужна Python-бібліотека для роботи з Telegram API через MTProto. Відзначається високою швидкістю і гнучкістю, що робить її ідеальною для великих проектів із збору даних. Підтримує взаємодію з повідомленнями, чатами, каналами, ботами та медіафайлами.
2. **Pyrogram**: Простота використання і мінімальна кількість коду є ключовими особливостями цієї бібліотеки. Pyrogram забезпечує асинхронне програмування і підходить для проєктів, де критичні зручність та швидкість розробки.
3. **Aiogram**: Спеціалізована на створенні асинхронних ботів для Telegram. Відзначається високою швидкістю в асинхронному середовищі, проте може бути менш ефективною для великих обсягів даних через специфіку асинхронного підходу

Таблиця 3.3 - Найпопулярніші бібліотеки для роботи з Telegram API

Характеристика	Telethon	Pyrogram	Aiogram
Основне призначення	Робота з Telegram API	Простота взаємодії з Telegram	Створення асинхронних ботів
Протокол	MTProto	MTProto	MTProto
Швидкість	Висока	Помірна	Висока
Обсяг запитів	Великий	Середній	Малий-середній
Асинхронність	Підтримується	Підтримується	Підтримується
Зручність для новачків	Складніше, вимагає більше коду	Просте використання	Середнє
Оптимальне застосування	Великі проекти, збір даних	Швидка розробка, нескладні задачі	Розробка ботів
Медіафайли та ресурси	Підтримується	Підтримується	Підтримується
Недоліки	Складність для новачків	Менш підходить для великих проєктів	Ускладнена робота з великими даними

Інші бібліотеки, такі як **python-telegram-bot** та **Telepot**, також можуть бути використані для взаємодії з Telegram API, проте вони менш популярні для задач збору великих обсягів даних. Вони зазвичай застосовуються для розробки простих ботів або взаємодії з API на базовому рівні.

Зважаючи на всі переваги та особливості, для проекту збору даних з Telegram найкращим вибором є **Telethon**. Це пояснюється кількома факторами: перш за все, Telethon надає найкращу продуктивність при роботі з великими обсягами даних і дозволяє легко взаємодіяти з Telegram через MTProto. Бібліотека також підтримує більшість функцій Telegram API, забезпечуючи гнучкість і масштабованість, необхідні для проведення досліджень, що вимагають збору та обробки великих обсягів інформації. Крім того, Telethon має гарну документацію та велику спільноту, що полегшує вирішення проблем, які можуть виникнути під час розробки.

3.2.1.1 Основні способи взаємодії з Telegram API.

Для взаємодії з Telegram API за допомогою бібліотеки **Telethon** існують два основні способи доступу: **MTProto** та **Bot API**. Кожен з них має свої особливості і застосовується в залежності від типу задач і вимог до інтеграції.

MTProto — це основний протокол, який Telegram використовує для комунікації між клієнтами і серверами. Взаємодія через MTProto надає більше можливостей та гнучкості порівняно з Bot API. За допомогою Telethon, який підтримує цей протокол, можна отримати доступ до всіх функцій Telegram, включаючи взаємодію з публічними каналами, приватними чатами, групами, а також отримання і надсилання повідомлень, медіа-файлів, метаданих, управління контактами та інше.

MTProto дозволяє працювати безпосередньо з основною інфраструктурою Telegram, що забезпечує повний контроль над отримуваними даними і дозволяє обходити обмеження, характерні для Bot API. Наприклад, через MTProto можна отримувати повідомлення з будь-яких публічних каналів або груп, навіть якщо бот не є їх адміністратором, а також взаємодіяти з великими обсягами даних без значних обмежень на запити. Це робить MTProto ідеальним для завдань, що вимагають збору і аналізу великих обсягів інформації.

Використовуючи **Telethon** з MTProto, можна отримати доступ до значно ширшого спектра функцій, що робить цей підхід більш підходящим для задач, пов'язаних з аналізом інформаційних потоків. Telethon через MTProto дозволяє створювати підключення безпосередньо до серверів Telegram, що надає можливість отримувати повідомлення з каналів, груп і навіть приватних чатів, якщо це необхідно для аналізу. Крім того, це дозволяє інтегрувати додаткові можливості, як-от отримання історії повідомлень, фільтрація контенту за певними критеріями (наприклад, за текстом або медіафайлами) і організація збору даних у режимі реального часу. У вашому випадку, коли мова йде про збори даних з

публічних каналів для подальшого аналізу мереж поширення інформації, використання **MTProto з Telethon** є найбільш оптимальним варіантом. Це дасть змогу не тільки отримати доступ до великих обсягів даних з каналів і груп, а й забезпечить високу швидкість обробки запитів, що особливо важливо при роботі з інформацією, що постійно оновлюється. Окрім того, можливість безпосередньої взаємодії з сервером Telegram через MTProto забезпечує більшу гнучкість у налаштуванні системи збору даних, що в свою чергу дозволяє краще контролювати процес і отримувати точніші результати аналізу.

3.2.1.2 Основні методи та функції MTProto API.

MTProto API, у поєднанні з бібліотекою **Telethon**, надає доступ до потужних функцій, що дозволяють взаємодіяти з платформою Telegram на глибшому рівні. Telethon організує взаємодію з MTProto API через кілька підмодулів, таких як **TelegramClient**, **errors**, **types**, і **functions**.

1. **TelegramClient** — це основний клас для підключення до Telegram через MTProto. Він є точкою входу для виконання запитів, таких як отримання повідомлень, надсилання повідомлень, взаємодія з каналами або групами. Телеграм-клієнт автоматично керує сесіями, обробляє запити та відповіді, а також забезпечує високий рівень абстракції для роботи з API.
2. **errors** — цей підмодуль дозволяє обробляти помилки, які можуть виникнути під час роботи з API, такі як перевищення ліміту запитів або доступ до ресурсів, до яких немає прав доступу. Завдяки цьому підмодулю, користувач може зручно реагувати на помилки, реалізуючи відповідні механізми повтору запитів або обробки винятків.
3. **types** — цей підмодуль містить всі необхідні типи даних для роботи з API, наприклад, **User**, **Message**, **Channel** тощо. Ці типи використовуються для

створення об'єктів, що представляють елементи Telegram, та для зручності подальшої обробки даних.

4. **functions** — підмодуль, що включає безліч функцій, за допомогою яких можна взаємодіяти з різними частинами Telegram. Він надає API для отримання інформації про канали, чати, повідомлення, користувачів та інші ресурси.

Серед найбільш корисних функцій, що можуть бути використані для збору даних і аналізу потоків інформації:

- **client.takeout** — ця функція дозволяє отримувати архіви даних користувача, включаючи повідомлення, контакти, медіафайли тощо. Вона є корисною для збору повних даних з особистих чатів або каналів, де необхідно зібрати всю історію взаємодії для подальшого аналізу.
- **functions.chatlists.CheckChatlistInviteRequest** — використовується для перевірки запрошень у списки чатів. Ця функція може бути корисною для збору даних, коли потрібно дізнатися, чи є користувач запрошеним до певного чату або каналу, що важливо для досліджень щодо поширення інформації.
- **functions.channels.GetFullChannelRequest** — дозволяє отримати детальну інформацію про канал, включаючи метадані, кількість учасників, списки адміністраторів та інші важливі дані. Це може бути корисно для аналізу структури каналів і визначення ступеня їхнього впливу на потоки інформації.
- **functions.messages.CheckChatInviteRequest** — ця функція дозволяє перевірити доступність запрошень для приєднання до чату чи каналу. Вона може бути використана для збору даних про приховані або закриті групи, що є важливим при виявленні потоку інформації в закритих колах.
- **get_entity** — ця функція дозволяє отримувати об'єкти Telegram (канали, групи, користувачів) за їхніми іменами або ID. Це корисно для створення

абстракцій і зручного доступу до різних елементів платформи без необхідності вручну шукати їх в системі.

- **get_messages** — функція для отримання повідомлень з каналів або чатів. Вона дозволяє отримувати як окремі повідомлення, так і певний їхній набір за заданими параметрами (наприклад, за датою, типом повідомлення). Ця функція є основною для збору текстової інформації з каналів і груп для подальшого аналізу.

Усе це дозволяє **Telethon** в поєднанні з **MTProto** API забезпечити повний доступ до даних Telegram і зібрати необхідну інформацію для аналізу. Для ваших задач, що пов'язані із збором великих обсягів даних з каналів і груп, використання **MTProto** разом із **Telethon** є найбільш ефективним варіантом, оскільки дозволяє обробляти запити на високій швидкості та з мінімальними обмеженнями, що особливо важливо при роботі з великими обсягами інформації.

3.2.1.3 Обмеження доступу до даних

При роботі з Telegram API через **Telethon**, слід враховувати ряд обмежень, які можуть вплинути на доступність і коректність виконання запитів. Telegram встановлює обмеження на доступ до даних з метою збереження продуктивності, безпеки та захисту від зловживань. Крім того, бібліотека **Telethon** надає підмодуль **errors**, який дозволяє обробляти помилки та правильно реагувати на різні типи обмежень з боку API.

3.2.1.4 Обмеження зі сторони Telegram

1. **Ліміти запитів** — Telegram накладає обмеження на кількість запитів, які можна виконати за певний період часу. Це стосується як отримання повідомлень, так і відправки запитів на сервери. Зокрема, на користувачів накладаються ліміти на:
    - Кількість запитів до API за певний проміжок часу.
    - Кількість запитів до конкретних методів.
    - Періоди обмежень, що можуть тривати від кількох секунд до кількох годин залежно від інтенсивності запитів.
2. **Часові обмеження** — Telegram накладає обмеження на кількість одночасних запитів і забезпечує їхня виконання з певними затримками для уникнення перевантаження серверів. Порушення цих лімітів може призвести до того, що доступ до певних функцій API буде тимчасово заблоковано.

3.2.1.5 Обробка помилок з підмодулем errors

Бібліотека **Telethon** обробляє помилки, що виникають при взаємодії з API, через спеціальний підмодуль **errors**. Він включає в себе різноманітні типи помилок, які можуть виникнути під час роботи з Telegram. Ось деякі з них, з якими можна зіткнутися:

1. **errors.rpcerrorlist.UsernameInvalidError** — ця помилка виникає, коли вказано некоректне або неіснуюче ім'я користувача або каналу. Це може статися, якщо неправильне ім'я використовується для пошуку користувача чи каналу або якщо ім'я було змінено або видалено.

2. **errors.rpcbaseerrors.BadRequestError** — з'являється, коли запит, відправлений на сервер, не відповідає вимогам Telegram API. Це може бути пов'язано з неправильно сформованим запитом або невірним параметром.
3. **errors.TakeoutInitDelayError** — помилка, яка з'являється при спробі ініціалізації експорту даних через **takeout**. Цей тип помилки вказує на те, що сервіс не може обробити запит через технічні або тимчасові проблеми, наприклад, при перевищенні ліміту запитів.
4. **errors.FloodWaitError** — виникає, коли система виявляє занадто велику кількість запитів з одного акаунта за короткий період часу. Telegram вводить тимчасове обмеження на подальші запити, що може тривати від кількох хвилин до кількох годин. Це є механізмом захисту від спаму та зловживань.
5. **errors.InviteHashExpiredError** — ця помилка з'являється, коли ви намагаєтесь приєднатися до каналу або групи за застарілим або непотрібним запрошенням. Telegram обмежує час дії запрошень, і після його закінчення спроба приєднатися буде заблокована.
6. **errors.ChannelPrivateError** — ця помилка виникає, коли користувач намагається доступити або отримати інформацію про приватний канал або групу, в якій він не є учасником. Це обмеження забезпечує конфіденційність та захист приватних груп.
7. **errors.rpcerrorlist.InviteHashExpiredError** — схожа на **InviteHashExpiredError**, але ця помилка виникає під час запиту на приєднання до приватного чату або каналу через спеціальне посилання, яке стало недійсним або застарілим.
8. **errors.rpcerrorlist.UsernameNotOccupiedError** — ця помилка з’являється, коли для вказаного імені користувача або каналу не знайдено відповідного об'єкта в системі. Вона може бути результатом неправильно введеного імені або спроби отримати доступ до неіснуючого об'єкта.

Усі ці обмеження і помилки демонструють необхідність ретельно планувати і оптимізувати взаємодію з Telegram API через **Telethon**, зокрема при зборі великих обсягів даних. Наприклад, при плануванні збору повідомлень із каналів необхідно враховувати ліміти на запити та періоди відпочинку після перевищення лімітів (наприклад, через **FloodWaitError**). Завдяки гнучкості підмодуля **errors**, можна ефективно обробляти ці ситуації та знижувати ризики для стабільності збору даних.

3.2.2 Підготовка до збору даних

Ефективний збір даних із Telegram залежить від ретельної підготовки, яка включає розробку критеріїв фільтрації каналів та вибір релевантних джерел для аналізу. Цей етап є ключовим для забезпечення якості зібраних даних і подальшого аналізу інформаційних потоків.

3.2.2.1 Розробка критеріїв фільтрації каналів

Одним із перших завдань у підготовці до збору даних є визначення критеріїв, за якими будуть відбиратися Telegram-канали. Ці критерії мають враховувати як технічні, так і тематичні аспекти, що відповідають меті дослідження.

1. **Тематика каналів**. Обираються канали, які публікують інформацію, релевантну до предмету аналізу. Наприклад, якщо дослідження спрямоване на аналіз дезінформації, то основна увага приділяється новинним каналам, джерелам конспірології або платформам, що розповсюджують пропаганду.
2. **Кількість підписників**. Для дослідження важливі як великі, так і невеликі канали. Популярні канали забезпечують доступ до значного обсягу

інформації, тоді як менші за кількістю підписників канали можуть виявити нішеві або приховані потоки інформації.

3. **Доступність повідомлень**. Вибираються канали з відкритим доступом до архіву повідомлень. Це дозволяє отримати історичні дані та проаналізувати, як змінювалися потоки інформації в часі.
4. **Географічна прив'язка та мова**. Враховується географічний регіон і мова публікацій, які можуть бути важливими для виявлення особливостей поширення інформації у різних країнах чи серед мовних груп.

3.2.2.2 Вибір каналів для аналізу

Після визначення критеріїв починається процес пошуку та відбору конкретних Telegram-каналів для аналізу. Цей етап включає як ручні, так і автоматизовані підходи.

1. **Ручний відбір**. Аналітики переглядають канали, які потенційно відповідають встановленим критеріям, на основі попереднього знання тематики, рекомендацій від спільноти або популярності каналів у медіапросторі. Ручний підхід забезпечує високу точність, але може бути обмеженим за масштабом.
2. **Використання пошукових інструментів**. Telegram надає можливість шукати канали за ключовими словами, хештегами або тематиками. Наприклад, можна шукати канали, пов'язані з певними політичними подіями, брендами або інформаційними кампаніями.
3. **Автоматизовані методи**. За допомогою бібліотек на кшталт Telethon можливо автоматично отримувати списки каналів, які відповідають певним параметрам. Наприклад, зібрати всі канали з певної категорії або з ключовими словами в описі чи назві.

Ретельно відібрані Telegram-канали становлять основу для подальшого збору даних. Важливо враховувати, що вибір джерел впливає на якість результатів аналізу, тому слід уникати включення джерел із сумнівною репутацією або тих, які містять надмірну кількість нерелевантної інформації.

3.2.3 Автоматизація процесу збору даних

Автоматизація процесу збору даних є ключовим етапом у дослідженні потоків поширення інформації, що дозволяє ефективно працювати з великими обсягами даних і мінімізувати участь людини. Цей процес передбачає використання спеціально розроблених скриптів для взаємодії з Telegram API, обробки отриманих даних і збереження їх у зручному форматі. Основна мета — створення автоматизованої системи збору, фільтрації та аналізу даних, яка здатна масштабуватися для роботи з великими мережами.

Нижче наведено спрощений алгоритм автоматизації збору даних у вигляді блок схеми на рисунку 3.1.

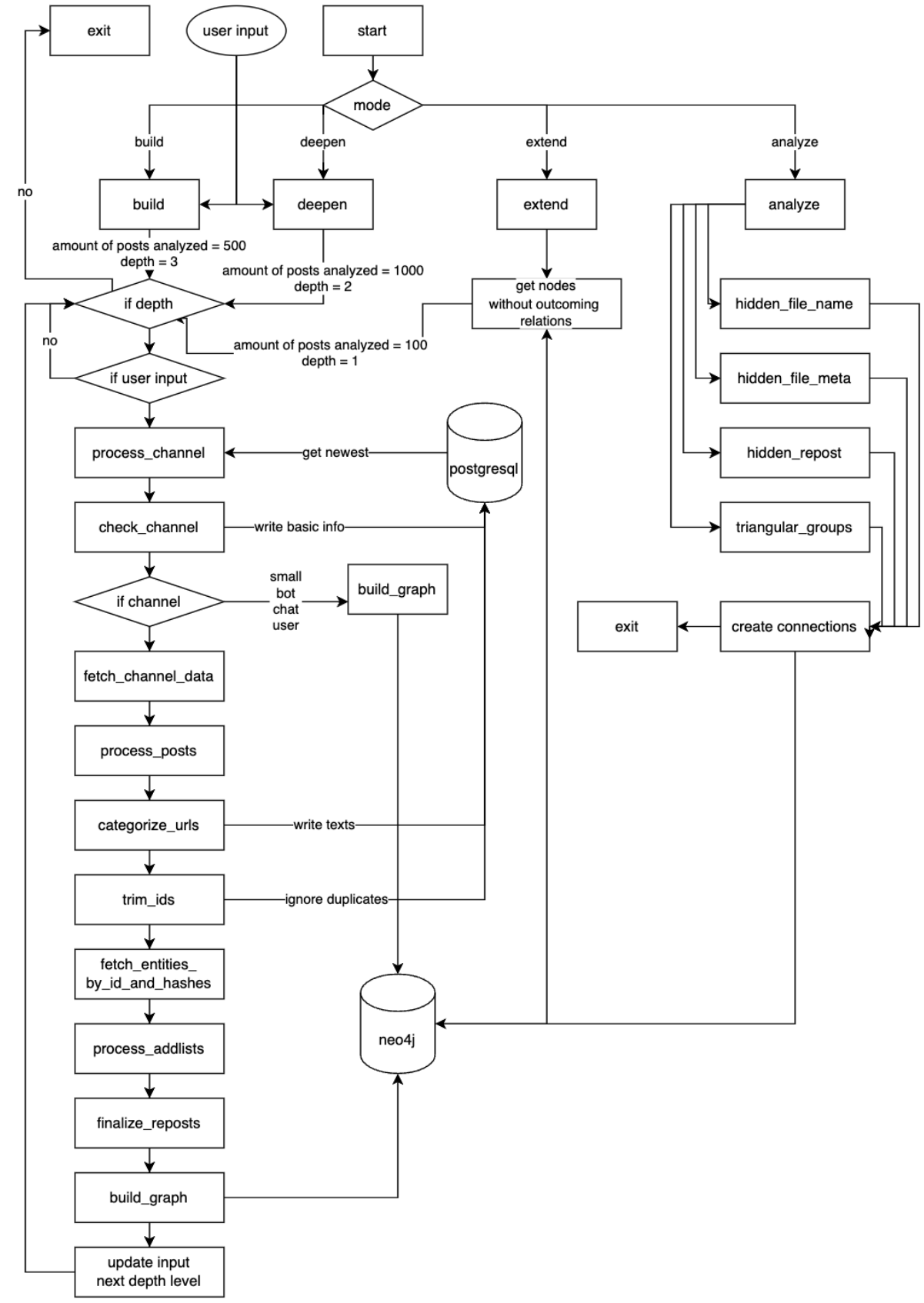

Рисунок 3.1 - Алгоритм автоматизації збору даних

Цей алгоритм дає змогу зібрати дані про канали, повідомлення, репости, медіа та інші атрибути, необхідні для аналізу потоків інформації в мережах Telegram. У наступних підрозділах буде детально розглянуто ключові етапи: написання скриптів для збору повідомлень, фільтрація даних, обробка повідомлень і зберігання отриманих даних у структурованій базі для подальшого аналізу.

3.2.3.1 Написання скриптів для отримання повідомлень

Для автоматизації збору повідомлень із вибраних каналів використовуються асинхронні скрипти, які взаємодіють з API Telegram через Telethon. Основним завданням є отримання постів із каналів та збереження їх у базу даних для подальшого аналізу. Скрипти повинні обробляти як текстові повідомлення, так і медіа-файли, що супроводжують їх.

Процес включає виклик функцій API для отримання інформації про канали та пости, а також обробку отриманих даних для збереження в зручному форматі. У випадку медіа-матеріалів скрипти повинні також здійснювати перевірку типу медіа (фото, документи, відео), щоб правильно зберігати відповідні атрибути, як розміри, формат, ID файлів та інші характеристики.

3.2.3.2 Фільтрація даних

Фільтрація даних є важливим кроком, оскільки зібрані повідомлення можуть включати значну кількість нерелевантної або зайвої інформації, яка не підлягає подальшому аналізу. Для ефективної фільтрації використовуються різноманітні підходи, зокрема:

1. **Фільтрація за URL-адресами**. Для визначення типу вмісту часто використовуються URL-адреси, що містяться у повідомленнях. Наприклад, якщо повідомлення містить посилання на Telegram-канали або специфічні ресурси, скрипт може класифікувати ці посилання для подальшого аналізу. Базуючись на таких URL, можна ідентифікувати канали або чати, з яких приходить інформація.
2. **Фільтрація за медіа**. Для дослідження важлива тільки текстова інформація, тому повідомлення з медіа можуть бути відфільтровані або оброблені окремо. У випадку з фото чи документами, важливо не зберегти самі файли, а зібрати додаткову інформацію, таку як розмір або формат.
3. **Фільтрація за змістом**. Текст повідомлень також підлягає фільтрації. Використовуючи методи обробки тексту, можна видаляти зайві символи, такі як посилання або непотрібні пробіли, що дозволяє зосередитись на змістовному вмісті постів. Також використовуються алгоритми для визначення схожості тексту, що дозволяє ідентифікувати репости або дублікати інформації.

3.2.3.3 Завдання для обробки повідомлень

Після отримання і попереднього очищення даних від непотрібних елементів, скрипти займаються аналізом змісту кожного повідомлення. Це включає в себе:

- Виявлення URL-адрес у тексті повідомлення.
- Ідентифікація хештегів і класифікація повідомлень за тематичними категоріями.
- Обробка медіафайлів (фото, документи) і збереження їх атрибутів для подальшої обробки.

Процес автоматизації дозволяє значно прискорити збирання даних з великих обсягів Telegram-каналів, а також зменшити людський фактор і ймовірність помилок при зборі інформації. У результаті цього кроку отримані дані зберігаються у відповідній структурі для подальшого аналізу, що робить систему збору даних високоефективною та гнучкою для подальшого використання в дослідженнях потоків інформації в мережі.

3.2.4 Зберігання отриманих даних

Створення схеми бази даних для зберігання інформації: Розробка схеми бази даних є важливим етапом для організації зберігання даних про повідомлення, канали, та їх атрибути. У цьому контексті необхідно визначити структуру таблиць, які будуть зберігати основні атрибути інформації, такі як ідентифікатори каналів, текст повідомлень, метадані (час публікації, автори, медіа-файли) та інші важливі характеристики. Таблиці повинні бути оптимізовані для швидкого доступу та ефективного зберігання даних.

Наприклад, створюється таблиця для зберігання повідомлень (telegram_posts), де кожен запис включає унікальний ідентифікатор каналу, ідентифікатор поста, текст, медіа та інші атрибути. Для уникнення дублювання даних використовуються обмеження у вигляді унікальних ключів для певних комбінацій полів, що дозволяє зберігати лише унікальні записи. Для зберігання асоціацій між хешами та даними використовуються окремі таблиці, наприклад, hash_to_data, яка зберігає зв'язки між хешами, назвами каналів та їх користувачами.

Для оптимізації взаємодії з Telegram API та зменшення кількості запитів до нього використовуються допоміжні таблиці, такі як id_to_username і hash_to_data.

1. Таблиця id_to_username: Ця таблиця дозволяє зберігати відповідності між ідентифікаторами каналів і їхніми іменами. Вона використовується для того,

щоб уникнути повторних запитів до API Telegram, коли необхідно отримати інформацію про канал за його ідентифікатором. Це значно скорочує кількість запитів і підвищує ефективність збору даних, особливо коли працюється з великою кількістю каналів.

2. Таблиця hash_to_data: Ця таблиця зберігає асоціації між хешами повідомлень та їх метаданими, такими як заголовки каналів і імена користувачів. Замість того, щоб кожного разу звертатися до API для перевірки, чи є хеш повідомлення в БД або для отримання додаткової інформації про канал, таблиця hash_to_data зберігає ці дані для повторного використання. Це знову зменшує навантаження на API і знижує час обробки.

Завдяки використанню таких таблиць значно зменшується кількість запитів до Telegram API, що є критичним при роботі з великими обсягами даних. Це дозволяє зберігати високу ефективність процесу збору даних і мінімізувати ризики перевантаження системи або порушення обмежень на запити до API Telegram.

Організація збереження текстів, метаданих та інших атрибутів повідомлень: Тексти повідомлень зберігаються в БД у таблиці telegram_posts разом з метаданими, такими як час публікації (post_date), автор поста (post_author), та додаткові атрибути (наприклад, медіа файли). Всі ці дані зберігаються у вигляді структурованих записів, де кожен пост отримує свій унікальний ідентифікатор для подальшого аналізу.

Для кожного повідомлення фіксується, чи є воно пересланим (через fwd_from), та якщо так — зберігаються відповідні ідентифікатори, щоб можна було відслідковувати потоки інформації в межах каналів. У разі наявності медіа-файлів (фото, відео) зберігаються метадані файлів, такі як розмір, тривалість та інші характеристики. Крім того, для текстів повідомлень застосовуються алгоритми для визначення подібності повідомлень, що зберігається в полях, як-от fuzz_ratio_1 або fuzz_ratio_2.

Збереження метаданих допомагає не лише в подальшому аналізі інформаційних потоків, але й у визначенні характеру контенту — чи є він оригінальним, чи перепостом з іншого каналу. Для цього використовуються додаткові атрибути, які дозволяють класифікувати повідомлення за типом контенту.

Цей підхід дозволяє організувати ефективне збереження та доступ до даних для подальшого аналізу інформаційних потоків в межах месенджерів, особливо в рамках дослідження кібербезпеки.

3.3 Розробка алгоритму аналізу та побудова графу потоків інформації

3.3.1 Попередня обробка даних

Попередня обробка даних є критичним етапом у процесі аналізу потоків інформації в месенджерах. Вона включає в себе кілька важливих кроків, зокрема токенізацію, нормалізацію текстів повідомлень, а також виділення ключових атрибутів, необхідних для подальшого побудови графу. Графова модель в даному випадку використовує як вузли канали, користувачів, ботів, домени з посилань та хештеги.

3.3.1.1 Виділення ключових атрибутів для побудови графу

Після токенізації та нормалізації тексту здійснюється виділення ключових атрибутів, які стають основою для побудови графу. Основні атрибути включають:

- **Канали**: Ідентифікація каналів, з яких надходять повідомлення, є важливим етапом. У кожному повідомленні визначається його канал, і відповідно, створюється вузол, що відображає цей канал у графі.

- **Користувачі**: Повідомлення також містять інформацію про користувача, який надіслав повідомлення. Якщо користувач часто згадується в різних каналах, це буде відображено у вигляді зв’язків між користувачами і каналами.
- **Боти**: Деякі повідомлення можуть містити в собі посилання на ботів, і ці посилання (чи теги), дуже легко вирізняються за обов’язковим закінченням “bot” у посиланні.
- **Домени з посилань**: Аналіз повідомлень також дозволяє виділяти домени, що містяться в URL-посиланнях, за допомогою регулярних виразів. Ці домени стають окремими вузлами графу, що відображають веб-ресурси, на які посилаються користувачі.
- **Хештеги**: Хештеги виступають важливими маркерами для класифікації контенту і виокремлення тематики повідомлень. Вони також використовуються для створення вузлів у графі, що пов’язані з відповідними каналами або користувачами.

Ці атрибути є основою для створення вузлів у графі і визначають, які зв’язки між ними повинні бути створені на наступному етапі.

Таким чином, попередня обробка даних дозволяє структурувати велику кількість текстової інформації у вигляді взаємопов’язаних елементів, готових для побудови графової моделі потоків інформації.

3.3.2 Побудова графової моделі

Побудова графової моделі потоків інформації включає кілька етапів, які спрямовані на створення та взаємодію вузлів і зв'язків у графовій БД, зокрема Neo4j. Цей процес дозволяє візуалізувати та аналізувати поширення інформації між різними елементами, такими як канали, користувачі, боти, домени та хештеги. Основними етапами є створення вузлів, встановлення зв'язків між ними та візуалізація результатів.

Граф $G = \{V, E\}$, де, V - множина вершин графа, $E \subseteq V \times V$ - множина ребер графа.

Множина вершин:

$$V = \{v_1, v_2 \dots v_n\}, v = (name, type)$$

де, $name$ - ім'я вершини (унікальний ідентифікатор)

$type \in \{CHANNEL, DOMAIN, USER, CHAT, BOT, SMALL, HASHTAG\}$

Множина ребер:

$$E = \{e_1, e_2 \dots e_m\}, e = \{v_i, v_j, count, type\}$$

де $v_1, v_2 \in V$, $count \in Z$,

$type \in \{REPOSTED, REFERENCES, TAGGED, TRIANGULAR\_GROUPS, HIDDEN\_REPOST, HIDDEN\_FILE\_META, HIDDEN\_FILE\_NAME\}$.

Операції над графом:

1. $create\_node(G, v, type)$: $G = (V, E) \rightarrow G' = (V', E)$, $V' = V \cup \{v\}$, якщо $v \notin V$
2. $update\_node\_label(G, v, new\_type)$:
   $G = (V, E) \rightarrow G' = (V', E)$, $V' = (V \setminus \{v\}) \cup \{v' = (name, new\_type)\}$
3. $create\_relationship(G, vi, vj, type, count)$: $G = (V, E) \rightarrow G' = (V, E')$

4. $create\_relationships\_in\_group(G, Vg, type)$: $G = (V, E) \rightarrow G' = (V, E')$,
   $Vg \subseteq V$ – підмножина вершин,
   $E' = E \cup \{(v_i, v_j, type, 1): v_i, v_j \in Vg, v_i = v_{ij}\}$

3.3.2.1 Створення вузлів та ребер у графовій БД

Першим кроком у побудові графу є створення вузлів, які будуть представляти основні елементи системи, такі як канали, користувачі, боти, домени та хештеги. Для кожного елементу в системі (каналу, користувачу, домену, хештегу) необхідно перевірити наявність такого вузла в графовій БД перед його створенням, щоб уникнути дублювання. Якщо вузол ще не існує, він створюється з відповідними властивостями.

Для створення вузлів використовується командний запит до бази даних, який перевіряє наявність вузла з певним атрибутом (наприклад, name). Якщо вузол з таким атрибутом уже існує, новий вузол не створюється. Якщо ж вузол відсутній, система автоматично генерує новий вузол з необхідними властивостями, такими як ім’я каналу, користувача або домену.

Після створення вузлів необхідно встановити зв’язки між ними. Зв’язки (або ребра) в графі визначають, як елементи системи взаємодіють один з одним. Наприклад, між каналами може бути створено зв’язок "REPOSTED", який вказує на те, що один канал зробив репост повідомлення з іншого каналу. Крім того, можуть бути створені зв’язки типу "REFERENCES" між каналами та доменами, якщо в повідомленнях присутні URL-посилання.

Для кожного зв’язку також визначається кількість повторів або взаємодій, що дозволяє оцінити інтенсивність поширення інформації. Всі ці зв’язки створюються за допомогою командних запитів, що гарантують правильну інтеграцію зв’язків з відповідними вузлами.

3.3.2.2 Візуалізація графу потоків інформації

Після того, як всі вузли та зв’язки будуть створені, наступним кроком є візуалізація графу. Візуалізація дозволяє побачити, як інформація розповсюджується між різними каналами, користувачами та іншими елементами системи. Вона є важливим інструментом для аналізу та вивчення тенденцій в поширенні інформації, а також для виявлення важливих каналів чи користувачів, що грають центральну роль у процесах поширення.

За допомогою інтерфейсу Neo4j, можна побудувати візуалізацію графу, яка наочно демонструватиме зв’язки між каналами, користувачами та іншими атрибутами. Така візуалізація може допомогти виявити впливові вузли (канали або користувачів), а також визначити, які канали мають найбільше зв’язків з іншими.

3.3.3 Оптимізація алгоритму

Оптимізація алгоритму аналізу інформаційних потоків є важливим етапом у забезпеченні ефективної роботи системи, зокрема при обробці великих обсягів даних та зменшенні кількості зайвих вузлів і зв’язків у графовій моделі. Ключовими аспектами оптимізації є зменшення навантаження на систему та покращення швидкості аналізу, що є необхідним для роботи з реальними даними.

3.3.3.1 Врахування великих обсягів даних

Одним із важливих факторів, що визначають ефективність аналізу, є великий обсяг даних, який необхідно обробити. У випадку з аналізом потоків інформації в месенджерах, обробка великої кількості повідомлень і каналів може значно збільшити навантаження на систему, тому важливо застосовувати

оптимізовані методи збереження та обробки даних.

Для зменшення навантаження використовуються кілька стратегій. По-перше, програма забезпечує попередню обробку даних, що дозволяє видалити непотрібні чи дубльовані повідомлення, а також ігнорувати менш важливі канали або зв'язки. Це дозволяє знизити кількість вузлів і зв’язків у графі та зменшити обсяг даних, що обробляються на наступних етапах.

По-друге, реалізовано збереження лише найбільш релевантних даних для подальшого аналізу, що також допомагає знизити навантаження на систему. Використання індексації та ефективних запитів до бази даних дозволяє оптимізувати доступ до даних, що особливо важливо при роботі з великими масивами інформації.

3.3.3.2 Зменшення кількості зайвих вузлів та зв’язків

Іншим важливим аспектом оптимізації є зменшення кількості зайвих вузлів та зв'язків, що можуть виникати під час аналізу даних. Для цього в алгоритм впроваджено механізм фільтрації, що дозволяє створювати лише ті зв’язки, які дійсно мають значення для аналізу інформаційного потоку. Наприклад, якщо канал не має явного зв’язку з іншими каналами або важливими атрибутами, такі зв’язки не створюються, що допомагає уникнути перевантаження графа зайвими даними.

Також у системі реалізовано зменшення дублювання інформації, зокрема для каналів, які повторно публікують однакові повідомлення. В такому випадку кількість зв’язків між вузлами зменшується, оскільки кожен зв’язок буде мати лише один запис, що підвищує ефективність і точність моделі.

3.3.3.3 Режими роботи програми

Програма пропонує кілька режимів роботи, що дозволяють адаптувати аналіз під різні потреби. Один із режимів передбачає побудову основної графової моделі на основі зібраних даних, що дозволяє швидко отримати загальну картину поширення інформації між каналами та іншими елементами системи. Однак для більш детального аналізу передбачено додаткові функції, що дозволяють поглибити дослідження.

У режимі поглибленого аналізу програма може збирати додаткові повідомлення, які не були проаналізовані на попередньому етапі. Це особливо корисно, якщо необхідно зібрати більше даних для конкретних каналів, щоб уточнити або доповнити існуючі зв’язки та інформацію в графі.

Інший режим дозволяє розширити граф, досліджуючи канали, що на даний момент не мають вихідних зв’язків. Це дає можливість виявити нові потенційні зв’язки та більш глибоко вивчити ті частини інформаційного простору, які не були охоплені раніше.

Таким чином, система є гнучкою і може працювати в різних режимах, забезпечуючи як швидкий базовий аналіз, так і можливість проведення більш детального та глибокого дослідження інформаційних потоків. Це дозволяє оптимізувати процеси збору та обробки даних відповідно до конкретних потреб користувача.

3.3.4 Виявлення зв’язків

Одним із важливих критеріїв для визначення зв’язків між вузлами є виявлення явних репостів. У рамках аналізу потоків інформації в месенджерах явними репостами вважаються випадки, коли канал публікує повідомлення або контент, який вже був опублікований іншим каналом. Це створює пряму

залежність між каналами та дозволяє побудувати зв’язки в графовій моделі.

Для виявлення явних репостів система аналізує публікації, які були повторно опубліковані на різних каналах. Визначення таких зв’язків відбувається через аналіз метаданих повідомлень, зокрема за допомогою таких атрибутів, як час публікації, контент та джерело. Якщо повідомлення, що публікується на каналі, збігається з повідомленням, яке вже було опубліковане іншим каналом, то між цими каналами створюється зв'язок типу "REPOSTED". Такий підхід дозволяє створювати чітке дерево зв'язків між каналами на основі явних репостів, що допомагає виявляти, як і через які канали інформація поширюється в мережі.

3.3.4.1 Ідентифікація прихованих зв’язків

Крім явних репостів, існують також приховані зв’язки між вузлами, які можуть бути не так очевидними, але мають велике значення для аналізу. Ідентифікація таких зв’язків є важливим етапом, оскільки дозволяє більш глибоко розуміти механізми поширення інформації.

Приховані зв’язки можуть виникати, коли кілька каналів взаємодіють через спільні елементи, такі як користувачі, боти, чати або домени з посиланнями. Ідентифікація таких зв’язків потребує аналізу непрямих взаємодій між вузлами. Одним із підходів є виявлення каналів, які мають спільних користувачів, ботів або чати, що взаємодіють із кількома каналами одночасно.

Для виявлення прихованих зв'язків система використовує запити до графової бази даних, орієнтуючись на підрахунок кількості зв’язків між вузлами. Наприклад, якщо користувач або бот активно взаємодіє з кількома каналами, можна зробити висновок, що ці канали можуть бути взаємопов’язані. Зокрема, якщо один і той же користувач має зв’язки з кількома каналами, це може вказувати на потенційну мережу каналів, пов'язаних між собою через цього користувача. Подібний підхід застосовується для виявлення зв’язків між каналами через

спільних ботів та чатів.

Це дає можливість не лише ідентифікувати більш очевидні зв’язки, але й побудувати більш складну картину взаємодії каналів, боту або чатів, що дозволяє глибше розуміти, як інформація може поширюватися через неявні, приховані канали.

3.3.4.2 Виявлення прихованих зв’язків на основі медіафайлів

Один із способів ідентифікації прихованих зв'язків полягає у вивченні медіафайлів, які були використані в публікаціях на різних каналах. За допомогою аналізу метаданих медіафайлів, таких як назви файлів, розміри, вага, висота або тривалість медіа, можна виявити канали, що публікують однакові або схожі медіафайли. Якщо кілька каналів публікують ідентичні медіафайли або файли з дуже схожими характеристиками, між цими каналами може бути прихований зв’язок, навіть якщо на перший погляд взаємодії між ними не видно.

Для виявлення таких зв’язків використовуються запити до бази даних, які об'єднують інформацію про медіафайли з усіх каналів. Наприклад, якщо один і той же медіафайл був опублікований на кількох каналах, це може вказувати на можливу взаємодію між цими каналами. Це дозволяє виявляти зв'язки на основі спільних медіафайлів і створювати мережу зв’язків між каналами, які інакше могли б залишитися непоміченими.

3.3.4.3 Виявлення прихованих зв’язків через метадані медіафайлів

Ще один метод виявлення прихованих зв'язків полягає в аналізі метаданих медіафайлів, таких як вага, розмір, висота та тривалість. Якщо кілька каналів публікують медіафайли, які мають схожі характеристики (наприклад, однаковий

розмір, вага або тривалість відео), це може свідчити про спільне джерело або схожі теми контенту. Такий підхід дозволяє збудувати більш детальну мережу взаємодій між каналами, виявляючи приховані зв'язки, які не завжди відображаються в очевидних репостах чи згадках.

3.3.4.4 Ідентифікація прихованих зв'язків через текстові повідомлення

Інший важливий метод для виявлення прихованих зв’язків полягає в аналізі тексту повідомлень на різних каналах. Зокрема, використання методу "fuzzy matching" для виявлення схожості текстів дозволяє знайти спільні фрагменти повідомлень, які можуть вказувати на приховану взаємодію між каналами. Якщо два або більше канали публікують повідомлення, що мають схожу або однакову текстову частину, це може бути ознакою того, що вони взаємодіють або отримують контент з одного джерела. Цей підхід допомагає виявляти неявні репости або переклади повідомлень між каналами, навіть якщо ці канали не мають явного зв'язку.

3.3.4.5 Виявлення прихованих зв’язків через тріангуляцію каналів

Тріангуляція каналів є ще одним методом для виявлення прихованих зв'язків. Цей метод полягає в аналізі послідовностей взаємодій між трьома каналами. Якщо канал A публікує повідомлення, яке потім перетворюється на контент для каналу B, і цей контент також публікується на каналі C, можна зробити висновок, що ці канали взаємодіють між собою, хоча ці взаємодії можуть бути непрямими. Тріангуляція допомагає побудувати більш складні зв'язки між каналами, що дає змогу розширити мережу взаємодій і краще зрозуміти потоки інформації між каналами.

Таким чином, ідентифікація прихованих зв’язків базується на комплексному аналізі медіафайлів, текстових повідомлень та послідовностей взаємодій між каналами. Це дозволяє не тільки виявляти явні зв'язки, а й розкривати складні мережі прихованих взаємодій, що є важливим для глибокого аналізу інформаційних потоків у месенджерах

3.3.4.6 Виявлення прихованих зв’язків за допомогою плагіну GDS

Для покращення ідентифікації прихованих зв'язків у графових базах даних можна застосувати спеціалізовані інструменти та алгоритми, зокрема плагін **Graph Data Science (GDS)** для Neo4j [18]. Цей плагін надає широкий набір алгоритмів, які дозволяють глибше аналізувати структури графів і виявляти приховані взаємодії, які не завжди очевидні.

Для ідентифікації прихованих зв'язків у графових базах даних можна ефективно використовувати алгоритми Louvain, SCC (Strongly Connected Components) та Node Similarity, які доступні через плагін Graph Data Science (GDS) для Neo4j.

1. **Виявлення спільнот за допомогою Louvain**:

   Алгоритм Louvain дозволяє ідентифікувати групи вузлів, які мають більше зв'язків між собою, ніж з іншими частинами графа. Це корисно для виявлення прихованих спільнот каналів, які активно взаємодіють, навіть якщо ці взаємодії неочевидні. Наприклад, групи каналів, що регулярно репостять один одного, можуть бути частиною єдиної спільноти.

2. **Аналіз сильно зв'язаних компонентів (SCC)**:

   Алгоритм SCC використовується для виявлення підграфів, де будь-який вузол доступний з будь-якого іншого вузла в межах цього підграфа. Це дає змогу виявити канали, які формують тісно пов’язані мережі, що можуть бути важливими для розуміння структури поширення інформації.

3. **Вимірювання схожості вузлів за допомогою Node Similarity**:

   Node Similarity дозволяє оцінити схожість між вузлами на основі спільних сусідів або атрибутів. Це корисно для виявлення каналів зі схожим контентом або поведінкою, навіть якщо вони не мають прямих зв’язків через репости. Наприклад, схожість може бути обчислена на основі таких характеристик, як спільні хештеги чи згадки.

Використання цих трьох алгоритмів дозволяє значно покращити розуміння структури графа, знаходити приховані зв'язки та глибше аналізувати інформаційні потоки у месенджерах.

Застосування GDS для виявлення прихованих зв'язків дозволяє не тільки підвищити точність аналізу, а й дозволяє ефективно обробляти великі обсяги даних, що необхідно для аналізу інформаційних потоків в умовах великих месенджерних мереж. Ці методи дозволяють знаходити нові взаємодії між каналами, які раніше не були помічені, що значно розширює можливості для дослідження інформаційних мереж.

3.4 Аналіз отриманих результатів та виявлення мереж поширення

3.4.1 Візуальний аналіз побудованого графу

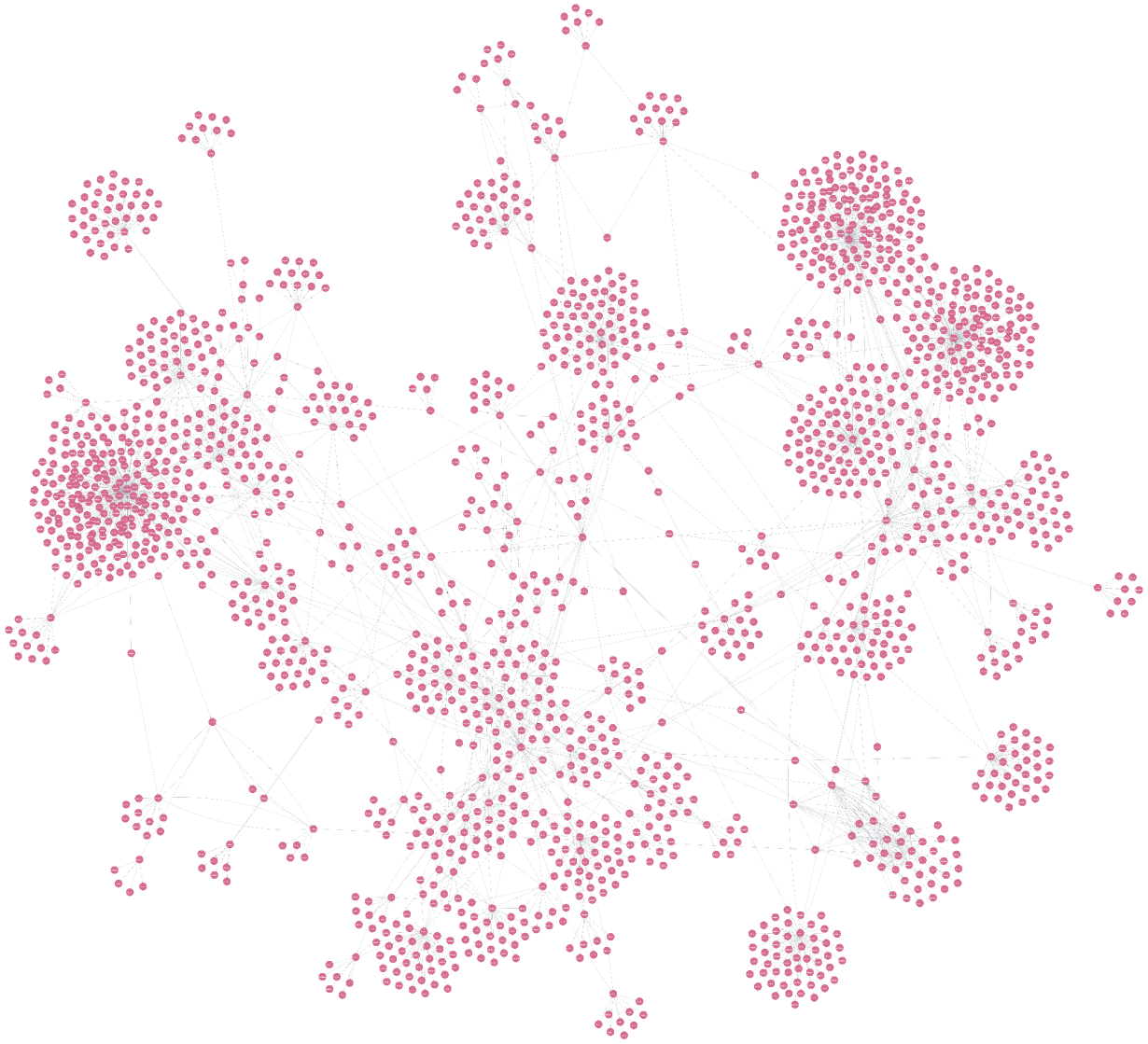

Рисунок 3.2 - Побудований граф

Граф на рисунку 3.2 має складну структуру, яка складається з кількох основних компонентів. Нижче подано ключові аспекти його аналізу:

1. Загальна структура графу

Граф складається з численних вузлів, згрупованих у кілька щільно зв’язаних кластерів. Між кластерами спостерігається невелика кількість зв’язків, що

свідчить про їхню модульність. Кластери формують окремі спільноти, які можуть відповідати групам каналів чи спільнотам у месенджерах. Невелика кількість зв’язків між ними може означати обмежений інформаційний обмін між групами.

2. Кластери (спільноти)

Виділяються окремі спільноти, що мають високу щільність зв’язків усередині. Розмір кластерів варіюється: є як великі групи, так і менші. Для підтвердження цієї структури варто провести кластеризацію графу за допомогою алгоритмів, таких як Louvain. Це допоможе зрозуміти, як інформація поширюється в межах спільнот і між ними.

3. Зв’язність графу

Граф є зв’язаним, тобто будь-який вузол можна досягти через шлях із кількох зв’язків. Водночас у ньому є вузли з малою кількістю зв’язків, які мають маргінальну роль. Для підтвердження зв’язності слід розрахувати щільність графу та визначити вузли з низьким ступенем зв’язності. Це дасть змогу оцінити, наскільки ефективно інформація може поширюватися по всій мережі.

4. Висновок візуального аналізу

Аналіз цього графу показує типову для соціальних мереж структуру, де є чітко виражені спільноти з високою щільністю внутрішніх зв’язків. Кластери, що виділяються, ймовірно, відповідають групам, які активно взаємодіють всередині, але мають обмежену взаємодію з іншими групами. Центральні вузли або хаби, що мають великий ступінь зв’язності, є основними точками поширення інформації, тоді як менш зв’язані вузли, ймовірно, виконують роль периферійних елементів з обмеженою участю в процесі поширення. Структура графу вказує на значну роль кластерів у розповсюдженні інформації всередині кожної групи, а також на можливу ізоляцію між різними групами, що потребує подальшого вивчення.

3.4.2 Аналіз виявлених зв’язків

3.4.2.1 Виявлення прихованих зв’язків на основі медіафайлів

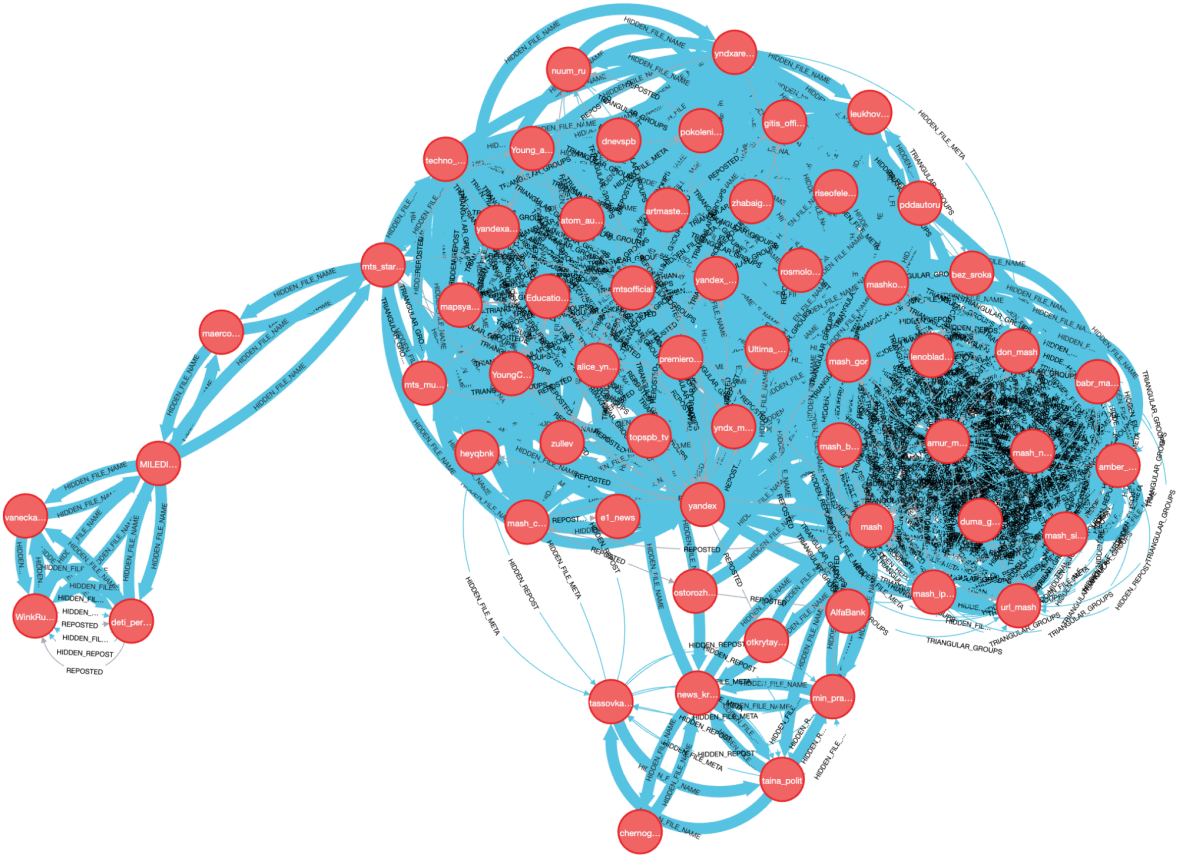

Рисунок 3.3 - Візуалізація прихованих зв’язків на основі медіафайлів

Графік на рисунку 3.3 ілюструє приховані зв’язки між каналами, які були виявлені через аналіз файлів із однаковими іменами. Метод базується на пошуку каналів, що розміщували медіафайли з ідентичними назвами, навіть без явного посилання один на одного. Спільне використання таких файлів свідчить про можливу координацію, наявність спільного джерела або приховані взаємодії між каналами.

3.4.2.2 Виявлення прихованих зв’язків через метадані медіафайлів

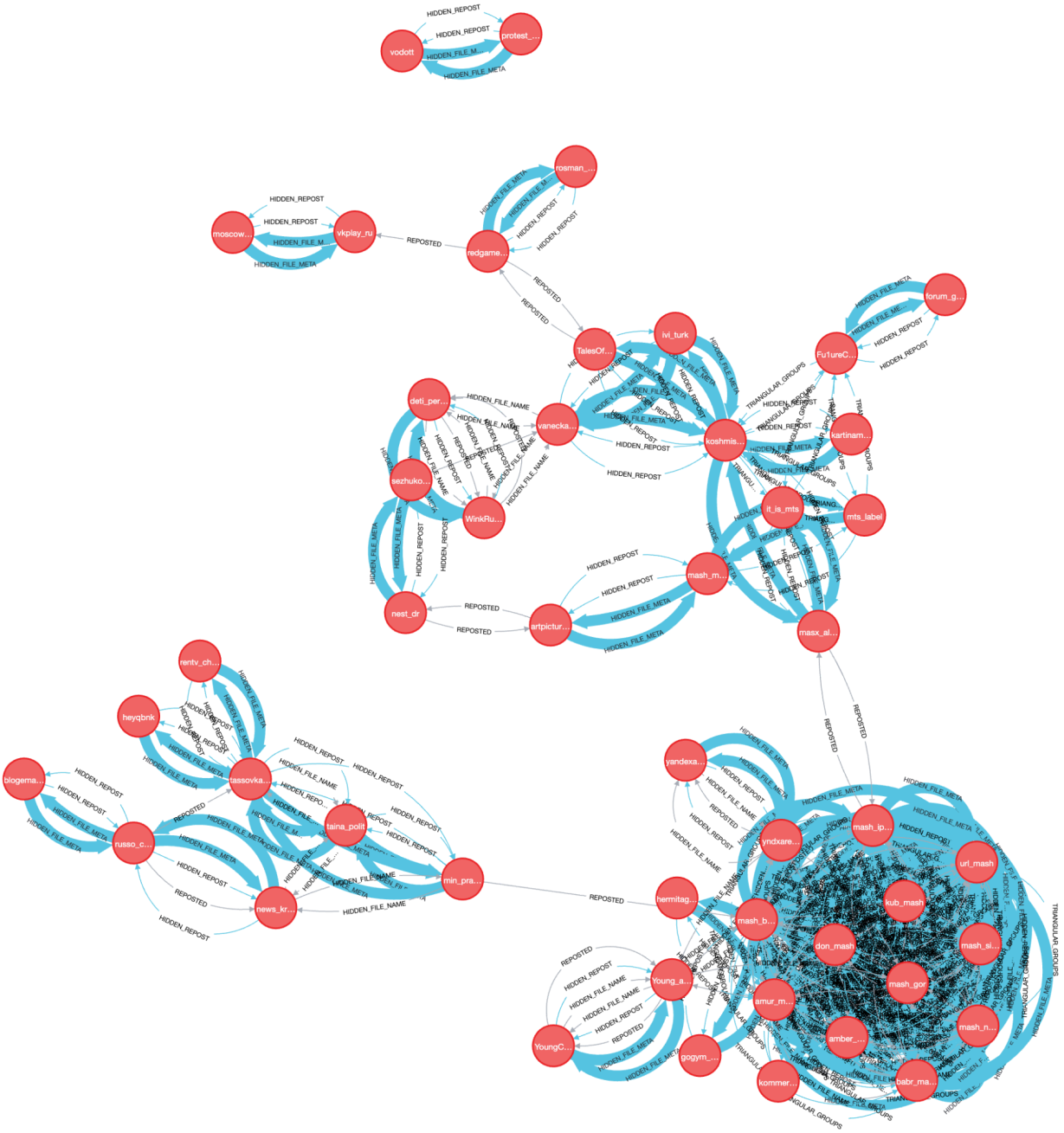

Рисунок 3.4 - Візуалізація прихованих зв’язків на основі метаданих медіафайлів

Граф на рисунку 3.4 ілюструє взаємозв’язки між каналами, виявлені через аналіз файлів із однаковими метаданими. У цьому підході досліджуються характеристики медіафайлів, такі як розмір, тривалість, роздільна здатність та інші атрибути, які залишаються незмінними, незалежно від назви файлу.

Виявлення таких збігів дозволяє встановити потенційні зв’язки між каналами, що використовують ідентичні файли. Це може свідчити про спільні джерела вмісту чи певну форму координації між цими каналами.

3.4.2.3 Ідентифікація прихованих зв'язків через текстові повідомлення

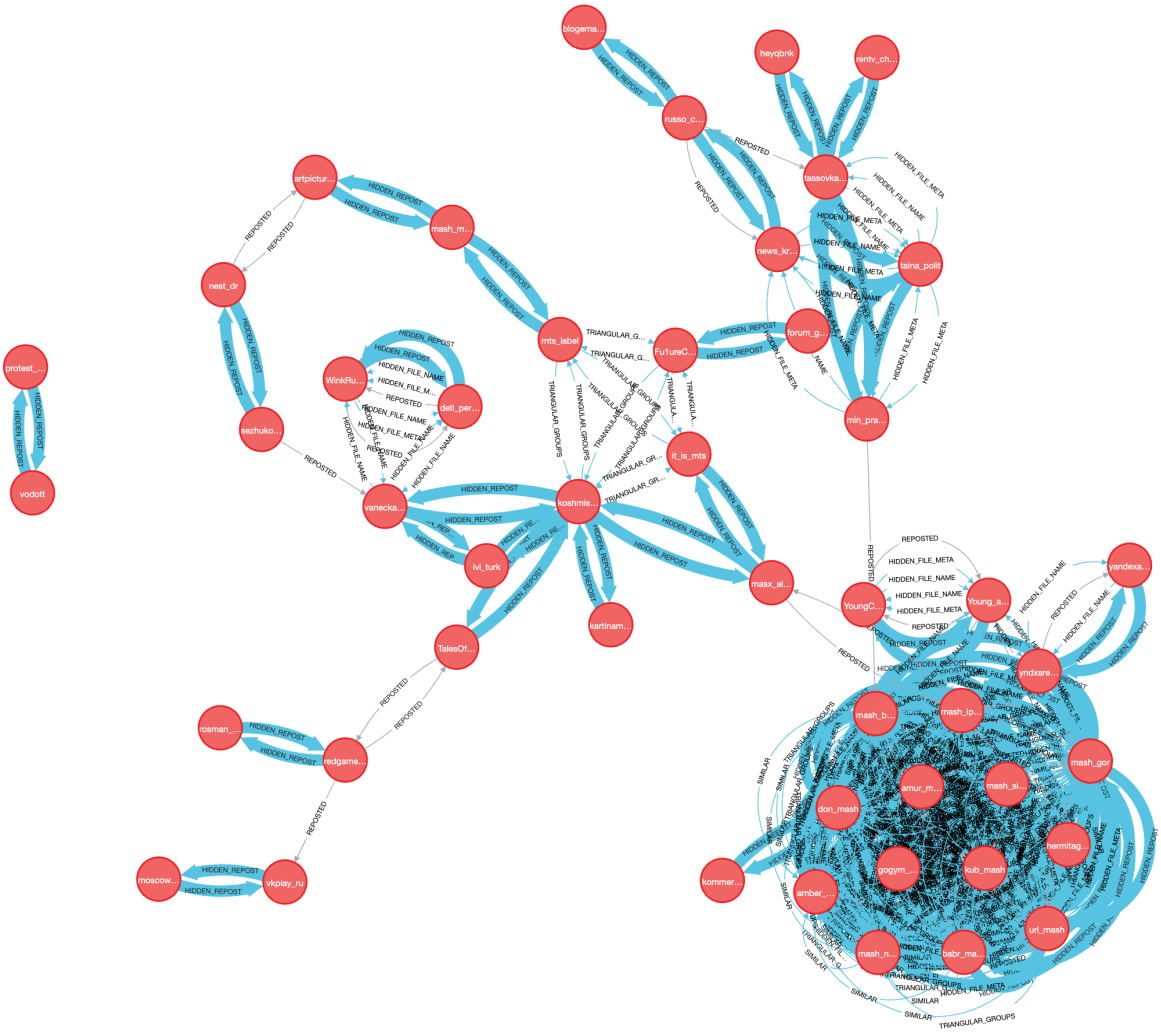

Рисунок 3.5 - Візуалізація прихованих зв'язків на основі текстових повідомлення

Графік на рисунку 3.5 демонструє взаємозв’язки між каналами, виявлені шляхом аналізу текстових повідомлень зі схожим змістом. Канали були пов’язані між собою, якщо їхні повідомлення містили текстові збіги, такі як ідентичні або майже ідентичні формулювання.

Окремі вузли, що мають лише кілька зв’язків, можуть вказувати на канали,

які лише частково беруть участь у мережі взаємодії, тоді як щільні вузли з великою кількістю зв’язків відображають більш інтегровані групи.

3.4.2.4 Виявлення прихованих зв’язків через тріангуляцію каналів

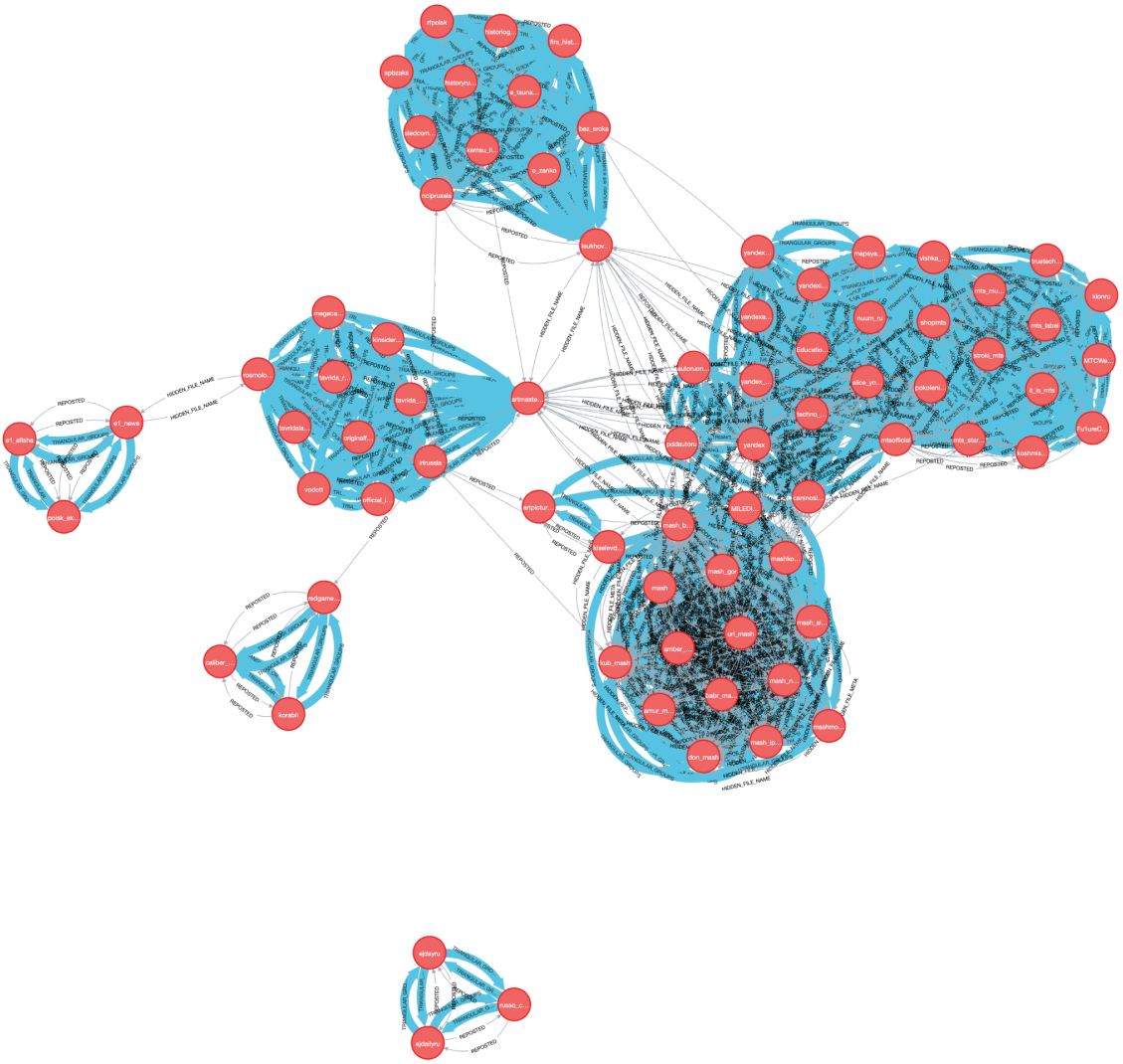

Рисунок 3.6 - Візуалізація прихованих зв’язків на основі тріангуляції каналів

Граф на рисунку 3.6 відображає взаємозв’язки між каналами, виявлені через метод тріангуляції, який дозволяє аналізувати непрямі взаємодії між трьома каналами. Суть методу полягає у відстеженні послідовностей, коли контент одного каналу стає основою для публікацій на двох інших каналах. Завдяки цьому можна розпізнати приховані зв’язки між учасниками інформаційної мережі, навіть

якщо ці зв’язки не є прямими.

На графіку помітно щільно взаємопов’язані кластери, які свідчать про активну координацію між каналами через проміжні точки взаємодії. Поряд із цим видно менш щільні зони, які демонструють окремі групи взаємодій, де канали співпрацюють із меншою частотою або в ізольованих контекстах. Окремі трикутні структури на графіку відображають локалізовані випадки взаємодій, де контент циркулює лише в межах обмеженої групи каналів.

3.4.2.5 Виявлення прихованих зв’язків за допомогою плагіну GDS

Louvain

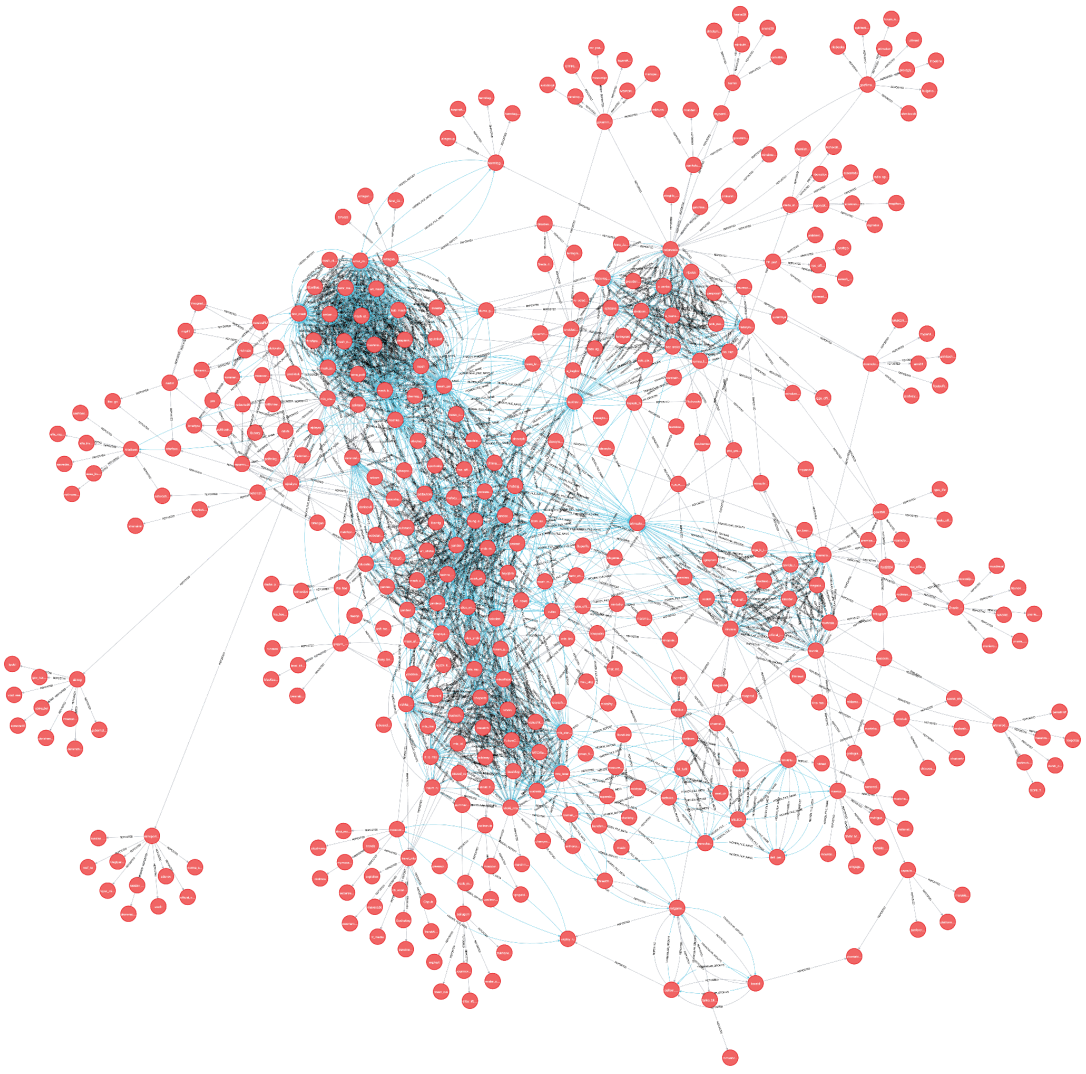

Рисунок 3.7 - Візуалізація прихованих зв’язків за допомогою алгоритму Лувена

Граф на рисунку 3.7 відображає мережу взаємодій між каналами, сформовану на основі спільнот, виявлених за допомогою алгоритму Лувена. Метод кластеризації дозволяє визначити групи каналів, що мають щільні внутрішні зв’язки, на основі аналізу їхньої активності, зокрема взаємних репостів.

На графі помітно декілька великих спільнот, що утворюють ядро взаємодій у мережі. Ці спільноти характеризуються інтенсивною координацією між каналами, що свідчить про активний обмін інформацією та можливу синхронізацію контенту. Окрім цього, видно менш щільні групи, які функціонують автономніше, але все ж мають зв’язки з основною мережею.

Зони з поодинокими зв’язками можуть вказувати на вузли-ретранслятори, які поширюють інформацію між різними спільнотами. Такі канали грають роль мостів, забезпечуючи циркуляцію контенту між віддаленими сегментами мережі.

Strongly Connected Components

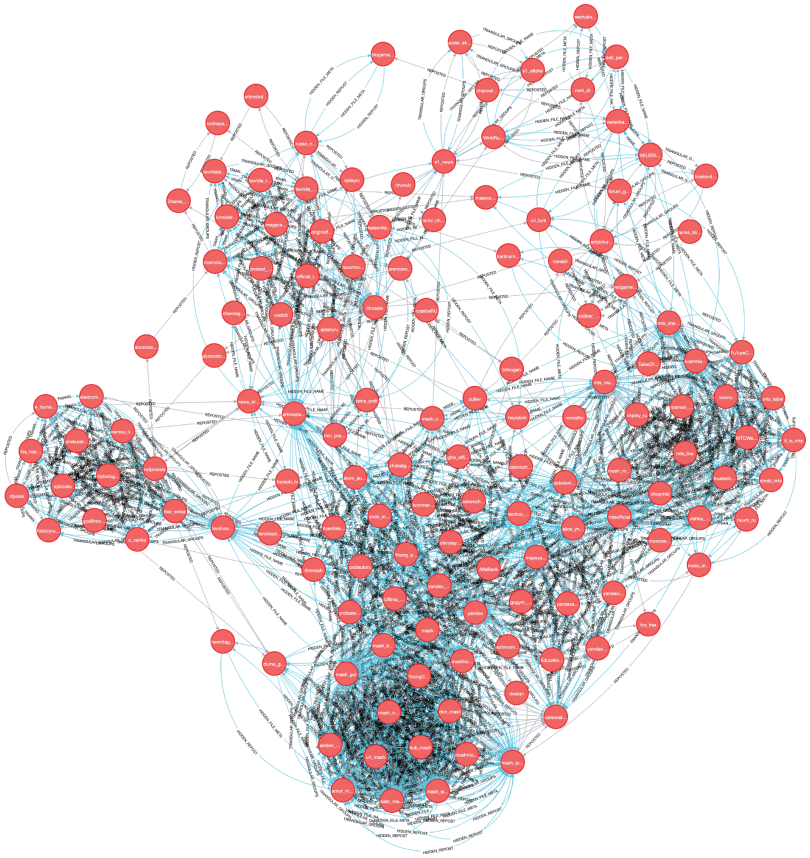

Рисунок 3.8 - Візуалізація прихованих зв’язків за допомогою алгоритму SCC

Граф на рисунку 3.8 відображає зв'язки між каналами, виявлені за допомогою методу сильно зв'язних компонент (SCC), що дозволяє ідентифікувати групи каналів, які взаємодіють між собою таким чином, що кожен канал у межах групи можна досягти з будь-якого іншого каналу. Цей підхід дає змогу виявити замкнені структури взаємодій, що можуть вказувати на координацію або спільну активність.

На графіку видно щільно взаємопов'язані кластери, які демонструють високий рівень взаємодії між каналами. Такі структури свідчать про активну координацію та, ймовірно, спільне поширення інформації в межах цих груп. Окрім цього, на графіку помітні менш щільні області, які відображають локалізовані мережі взаємодій або окремі групи, що діють більш ізольовано

Node Similarity

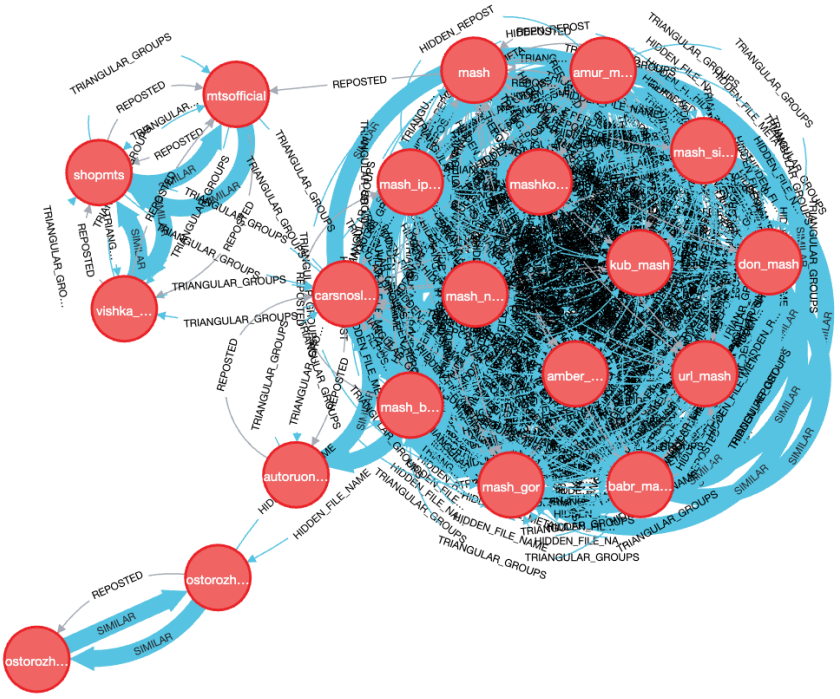

Рисунок 3.9 - Візуалізація прихованих зв'язків за допомогою алгоритму nodeSimilarity

Граф на рисунку 3.9 відображає взаємозв’язки між каналами, виявлені за допомогою алгоритму подібності вузлів (nodeSimilarity) з бібліотеки GDS. Алгоритм визначає схожість між каналами шляхом аналізу їхніх сусідів у графі, тобто тих каналів, із якими вони взаємодіють через репости. Такий підхід дозволяє виявити канали зі спільними інформаційними зв’язками, навіть якщо ці зв’язки є опосередкованими.

На графіку видно кілька щільно взаємопов’язаних кластерів, що свідчать про активну взаємодію між каналами, які мають високу схожість за змістом. Ці кластери вказують на групи каналів, що часто обмінюються контентом або мають спільні інтереси.

Окрім цього, присутні менш щільно пов’язані зони, які показують канали, що мають слабші зв’язки або взаємодіють рідше. Вони можуть бути частиною відокремлених груп або взаємодіяти в обмежених контекстах.

Особливу увагу привертають трикутні структури, що з’являються на графіку, де канали взаємодіють через посередників, утворюючи замкнуті цикли. Це є індикатором того, як контент циркулює між кількома каналами, навіть якщо ці зв’язки не є прямими

3.4.3 Інтерпретація отриманих результатів

Методи виявлення прихованих зв’язків між каналами на основі медіафайлів, метаданих, текстових повідомлень та алгоритмів кластеризації дозволяють створити багатогранну картину взаємодій у мережі. Кожен з підходів має свої унікальні переваги, спрямовані на виявлення різних типів зв’язків. Методи, які орієнтуються на медіафайли, як-от аналіз однакових імен файлів або їхніх метаданих, дозволяють виявити приховані зв’язки, що не відображаються в явних текстових чи репостах. Вони можуть вказувати на спільні джерела інформації або координацію між каналами без безпосереднього посилання один на одного.

Тим часом аналіз текстових повідомлень і тріангуляція каналів відкривають можливості для виявлення непрямих зв’язків, де інформація циркулює між кількома каналами, але не завжди безпосередньо. Це може свідчити про більш складну мережу взаємодій, що сприяє глибшому розумінню того, як інформація поширюється через різні платформи. Алгоритми, такі як Лувен або SCC, дозволяють кластеризувати канали в групи, де внутрішні зв’язки є більш інтенсивними, що дає змогу виявити основні групи впливу і координування в мережі.

Таблиця 3.4 - Порівняння методів виявлення зв’язків

Метод виявлення зв’язків	**Опис**	**Тип зв’язків**	**Переваги**	**Недоліки**
Медіафайли з однаковими іменами	Виявлення зв’язків між каналами через спільне використання медіафайлів з однаковими іменами.	Прямі зв’язки через спільні медіафайли	Виявляє можливу координацію без явних посилань, ефективно для прихованих взаємодій.	Обмежене застосування, залежить від наявності медіафайлів.
Метадані медіафайлів	Аналіз характеристик медіафайлів (розмір, тривалість, роздільна здатність) для виявлення схожих файлів.	Прямі зв’язки через метадані файлів	Дозволяє виявити приховані взаємодії навіть без явних зовнішніх ознак.	Може пропустити важливі зв’язки, якщо файли мають інші характеристики.
Текстові повідомлення	Пошук зв’язків через схожі або ідентичні текстові повідомлення.	Прямі зв’язки через текст	Виявляє спільні інформаційні потоки, корисно для аналізу повторюваного контенту.	Залежить від наявності однакових чи подібних текстів, може пропустити непрямі зв’язки.
Тріангуляція каналів	Виявлення зв’язків між каналами через непрямі взаємодії, коли контент одного каналу використовується двома іншими.	Непрямі зв’язки через проміжні канали	Виявляє приховані зв’язки навіть при відсутності прямих репостів.	Може не виявляти зв’язки у малих групах.

Кінець таблиці 3.4

Алгоритм Лувена (Community Detection)	Виявлення груп каналів із щільними внутрішніми зв'язками на основі алгоритму Лувена для кластеризації.	Групи каналів з активною координацією	Визначає основні центри впливу в мережі, дозволяє виділити більш впливові канали.	Не завжди враховує малочисельні, але важливі зв'язки.
Сильно зв'язні компоненти (SCC)	Виявлення замкнених груп каналів, де кожен канал досяжний з будь-якого іншого в межах групи.	Замкнені структури з високою взаємодією	Виявляє групи з високим рівнем координації, корисно для аналізу синхронізації контенту.	Може ігнорувати менш явні або ізольовані зв'язки.
Node Similarity (Подібність вузлів)	Аналіз схожості каналів через їхніх сусідів у графі на основі алгоритму подібності вузлів.	Схожість через сусідів	Виявляє канали з подібними інтересами, навіть якщо зв'язки не є прямими.	Може не виявити приховані зв'язки у випадку слабкої взаємодії.

Загалом, поєднання цих підходів дає можливість побудувати повнішу картину інформаційної мережі, де можна виявити як очевидні, так і приховані зв'язки між каналами, що важливо для аналізу поширення контенту та вивчення структури взаємодій. Кожен метод доповнює інший, надаючи різні перспективи для розуміння глибини і складності зв'язків у мережі.

3.4.4 Перевірка гіпотез

Аналізуючи різні методи виявлення неявних зв'язків, кожен із них має свої сильні сторони, залежно від контексту і мети дослідження. Однак, найбільш дієвим способом видається **тріангуляція** через її здатність розкривати складні та багаторівневі взаємодії.

Аргументація:

1. **Універсальність**: Тріангуляція дозволяє виявляти не лише прямі зв’язки, як у випадку аналізу спільних файлів чи метаданих, а й непрямі. Це забезпечує глибший огляд інформаційних потоків і дозволяє побудувати більш повну карту взаємодій між каналами.
2. **Виявлення мереж**: Завдяки аналізу трьохсторонніх взаємодій тріангуляція розкриває структури, які могли б залишитися невидимими при використанні інших методів, орієнтованих на прямі співпадіння (наприклад, файли або текстові збіги).
3. **Складність зв’язків**: На відміну від методів, які базуються на простих ознаках, таких як назви файлів або текстові повідомлення, тріангуляція враховує послідовності дій і виявляє зв’язки навіть тоді, коли немає явного контентного перетину.

Хоча інші методи мають свої переваги в конкретних ситуаціях, вони більше підходять для пошуку очевидних зв’язків. Тріангуляція ж виділяється своєю здатністю розкривати приховані, багатошарові зв’язки, які є особливо важливими для аналізу складних інформаційних потоків у мережах.

Висновки до розділу 3

Вибір середовища для збору даних став ключовим етапом у розробці системи для виявлення потоків поширення інформації. Telegram було обрано як основну платформу через її відкритий доступ до публічних даних, розвинуті технічні можливості API та популярність серед користувачів. Це дозволило створити гнучку систему для аналізу великих обсягів інформації.

Збір даних реалізовано за допомогою Telegram API. Було розроблено та впроваджено автоматизовані скрипти, які забезпечують ефективне отримання

текстових повідомлень, метаданих та медіафайлів із публічних каналів. У процесі збору враховано технічні обмеження API, що дало змогу уникнути перевищення лімітів запитів та забезпечити стабільність роботи системи. Зібрані дані піддавалися фільтрації та обробці, включаючи виділення ключових атрибутів.

Для забезпечення ефективної роботи системи алгоритм збору й аналізу даних було оптимізовано. Зменшено кількість дубльованих вузлів і зв'язків, застосовано методи фільтрації нерелевантної інформації.

Для моделювання потоків інформації було створено графову модель, яка відображає взаємодії між вузлами — каналами, користувачами, хештегами, доменами з посилань тощо. У графі побудовано зв'язки, що відображають репости, згадки та інші типи взаємодій. Візуалізація моделі виявила кластеризацію спільнот, центральні вузли та основні маршрути поширення інформації.

Окрім явних зв'язків, система дозволяє ідентифікувати приховані взаємодії між каналами. Аналіз проводився через вивчення медіафайлів, текстових повідомлень, тріангуляцію каналів та застосування алгоритмів кластеризації з використанням плагіна GDS для Neo4j. Це дало змогу виявити як прямі, так і опосередковані зв'язки, що інакше залишалися б непомітними. Алгоритми, такі як Louvain і SCC, дозволили визначити щільно пов'язані спільноти, а аналіз схожості вузлів розкрив канали зі спільними інформаційними інтересами. Найефективнішим методом виявлення прихованих взаємодій визнано тріангуляцію каналів, оскільки вона розкриває багаторівневі зв'язки, які важко ідентифікувати іншими способами.

На основі отриманих даних було проведено комплексний аналіз графу, який показав типову для соціальних мереж структуру з окремими кластерами та впливовими вузлами. Центральні вузли виявилися ключовими точками поширення інформації, тоді як менш зв'язані вузли відігравали роль периферійних елементів. Глибший аналіз показав, що інформація поширюється переважно всередині кластерів із обмеженим обміном між ними.

Таким чином, розроблена система ефективно реалізує виявлення потоків поширення інформації, дозволяючи аналізувати складні взаємодії між каналами, визначати ключові точки впливу та досліджувати особливості інформаційних мереж. Отримані результати демонструють високу ефективність запропонованого підходу і можливість його застосування для моніторингу та дослідження поширення контенту в інформаційних мережах.

4 РОЗРОБКА СТАРТАП-ПРОЄКТУ

4.1 Опис ідеї проекту

Стартап-проект спрямований на створення системи аналізу поширення інформації в месенджерах для потреб кібербезпеки. Ця система дозволить автоматизувати збір і обробку відкритих даних із публічних каналів Telegram, використовуючи API та технології графових БД. Основною метою є виявлення мереж поширення інформації для захисту інформаційного простору від дезінформації та кіберзагроз.

Таблиця 4.1 - Опис ідеї стартап ‑ проєкту

Зміст ідеї	**Напрямки застосування**	**Вигода для користувача**
Розробка системи аналізу інформаційних потоків у месенджерах	Кібербезпека, журналістика, аналіз соціальних медіа	Автоматизація процесів аналізу, підвищення ефективності виявлення дезінформації та інформаційних атак
Створення графових моделей зв’язків каналів	Розвідка у відкритих джерелах (OSINT), моніторинг впливу	Виявлення прихованих мереж, оцінка ризиків поширення шкідливого контенту
Використання Telegram API для автоматичного збору даних	Аналіз тенденцій у соціальних мережах	Швидкість та точність отримання актуальної інформації

Розглянемо техніко-економічні характеристики ідеї, а також проведемо порівняння з конкурентними рішеннями для ідентифікації слабких, сильних та нейтральних сторін.

Таблиця 4.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проєкту

Техніко-економічні характеристики ідеї	**Мій проект**	**Конкуренти**					**W**	**N**	**S**
		CrowdTangle	**SocialBlade**	**Social Bearing**	**Botometer**	**doesfollow**			
Використання Telegram API	Автоматичний збір даних із публічних каналів	Працює лише з платформами Meta	Збирає дані про користувачів YouTube, Twitch	Відстеження ключових слів у Twitter	Аналіз акаунтів Twitter на ботоподібність	Аналізує підписки в Twitter	-	-	+
Графові бази даних	Візуалізація зв’язків між каналами	Немає інструментів для зв’язків	Відсутня підтримка графових БД	Відсутня підтримка графів	Відсутня підтримка графових моделей	Відсутня підтримка графів	-	-	+
Витрати	Початкові витрати на розробку	Потребує підписки або спеціального доступу	Безкоштовний базовий доступ	Безкоштовний або з невеликими витратами	Безкоштовний базовий доступ	Безкоштовний базовий доступ	-	+	-
Прибуток	Потенціал масштабування	Орієнтований лише на бізнес Meta	Обмежена монетизація	Переважно використовується для аналізу	Немає прямої монетизації	Немає прямої монетизації	-	+	-
Пошук фейкових новин	Автоматизоване виявлення	Дозволяє відстежувати лише загальнодоступний контент	Не передбачений	Відстеження ключових слів, обмежена точність	Оцінка акаунтів, а не контенту	Не аналізує контент	+	-	-
Інтеграція з іншими системами	Легка інтеграція через API	Потребує інтеграції лише з платформами Meta	Інтегрується з YouTube, Twitch	Підтримує лише Twitter	Орієнтований на Twitter	Лише Twitter	-	-	+

4.2 Технологічний аудит ідеї проекту

У таблиці 4.3 представлено технологічний аудит ідеї проекту, де перераховані основні технології, які використовуються для реалізації проекту, їх наявність та доступність. Для автоматизації збору даних із Telegram використовується Telegram API в поєднанні з Python, що дозволяє ефективно отримувати інформацію з публічних каналів. Ця технологія є вже розробленою і має високу доступність. Для візуалізації зв'язків між каналами застосовується Neo4j разом з Graph Data Science Library, що дає змогу створювати графи і аналізувати їх. Технології для візуалізації також розроблені і доступні на високому рівні. Аналіз репостів для виявлення мереж поширення здійснюється за допомогою алгоритмів графового аналізу, таких як Node Similarity, Louvain і SCC, що дозволяє виявляти приховані зв'язки між каналами. Ці алгоритми також розроблені, а доступність їх є високою. Для виявлення неявних поширень контенту використовуються спеціально розроблені алгоритми, що аналізують текстовий зміст та медіа для виявлення неявної взаємодії між каналами. Хоча ця технологія також розроблена, її доступність оцінюється як помірна, оскільки потребує додаткових налаштувань і адаптацій.

Для зберігання та обробки великих обсягів даних застосовуються PostgreSQL та Neo4j, що дозволяє зберігати структуровану інформацію та створювати графи для подальшого аналізу. Ці технології є розробленими і мають високу доступність. Інтеграція з іншими системами здійснюється за допомогою REST API та формату JSON, що забезпечує взаємодію між різними компонентами системи. Ці технології також розроблені і мають високу доступність для інтеграції з іншими сервісами.

Таблиця 4.3 - Технологічна здійсненність ідеї проєкту

Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
Автоматизація збору даних із Telegram	Telegram API, Python	Розроблено	Висока
Візуалізація зв'язків між каналами	Neo4j, Graph Data Science Library	Розроблено	Висока
Аналіз репостів для виявлення мереж поширення	Алгоритми графового аналізу (PageRank, Louvain, SCC)	Розроблено	Висока
Виявлення неявних поширень контенту	Спеціально розроблені алгоритми аналізу змісту дописів	Розроблено	Помірна
Зберігання та обробка великих обсягів даних	PostgreSQL, Neo4j	Розроблено	Висока
Інтеграція з іншими системами	REST API, JSON	Потребує розробки	Висока

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Для аналізу ринкових можливостей стартап-проекту наводяться основні показники стану ринку, що допомагають оцінити потенціал проекту на різних етапах його розвитку. Оцінка стану ринку дозволяє зрозуміти, які фактори можуть вплинути на запуск і розвиток стартапу, а також визначити потенційні ризики та можливості для його масштабування.

Таблиця 4.4 - Попередня характеристика потенційного ринку стартап-проєкту

Показники стану ринку	Характеристика
Розмір ринку	Ринок месенджерів, зокрема Telegram, демонструє швидке зростання завдяки зростанню популярності серед користувачів і бізнесу, що активно використовують платформи для комунікацій.
Конкуренція на ринку	На ринку існують компанії, які займаються аналізом даних в соціальних мережах та месенджерах (наприклад, CrowdTangle), проте специфіка аналізу Telegram як каналу поширення інформації залишається недостатньо розробленою.
Попит на технології збору і аналізу даних	Підвищення попиту на аналіз відкритих даних і виявлення фейкових новин створює сприятливі умови для запуску стартапу, оскільки питання довіри до інформації в онлайн-середовищі є важливим.
Інноваційність технології	Використання графових БД для аналізу поширення інформації є інноваційним підходом у порівнянні з іншими методами збору та обробки даних, що відкриває нові можливості для виявлення неявних зв'язків і патернів.
Етап розвитку ринку	Ринок знаходиться на етапі активного розвитку: все більше користувачів та бізнесів розглядають месенджери як канал для поширення новин і комунікацій. Однак існує потреба у вдосконалених інструментах для їх аналізу.
Регулювання і правові обмеження	Враховуючи активні обговорення щодо регулювання даних і захисту конфіденційності, запуск стартапу може зіткнутися з юридичними питаннями, що потребує врахування при розробці продукту.
Технологічний прогрес	Збільшення можливостей для обробки великих обсягів даних, розвиток штучного інтелекту та алгоритмів машинного навчання сприяють створенню ефективних інструментів для аналізу даних і виявлення інформаційних мереж.

Наступна таблиця 4.5 дозволяє зрозуміти потреби та вимоги різних цільових аудиторій для стартапу, а також те, як вони поводяться з аналізованими технологіями. Визначення відмінностей у поведінці та вимогах клієнтів дає можливість краще адаптувати продукт до конкретних потреб ринку.

Таблиця 4.5 - Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	**Цільова аудиторія (цільові сегменти ринку)**	**Відмінності у поведінці різних потенційних цільових груп клієнтів**	**Вимоги споживачів до товару**
Пошук фейкових новин і виявлення маніпуляцій	Журналісти, медіа-агентства, аналітичні центри, громадські організації, державні установи	Журналісти та аналітики потребують точних даних для перевірки фактів, громадські організації — зручних інструментів для виявлення маніпуляцій у контенті	Потреба у швидкості, точності, інтеграції з іншими системами, доступність даних про контент, простота використання інтерфейсу

Наступна таблиця 4.6 дозволяє оцінити основні загрози та можливості для стартапу, що допомагає розробити стратегії для мінімізації ризиків і максимізації потенціалу на ринку

Таблиця 4.6 - Фактори загроз і можливостей

Фактор	**Зміст загроз**	**Можлива реакція компанії**
Загрози		
Конкуренція	Зростаюча кількість конкурентів на ринку інструментів для аналізу даних, таких як CrowdTangle, Botometer, що можуть запропонувати схожі функції.	Розвиток унікальних функціональних можливостей, підвищення якості обслуговування клієнтів, впровадження нових алгоритмів аналізу для підтримки конкурентоспроможності.
Правові обмеження	Можливі зміни у законодавстві, що регулюють використання даних та приватність, зокрема у контексті збору інформації з публічних платформ, таких як Telegram.	Постійне відслідковування змін у законодавстві, адаптація продукту до нових вимог, забезпечення відповідності політикам захисту даних.
Технічні ризики	Висока залежність від зовнішніх технологій (Telegram API, Neo4j), що можуть змінюватися або ставати недоступними.	Розробка резервних рішень та альтернативних технологій для забезпечення стабільної роботи, створення більш гнучкої архітектури продукту.

Кінець таблиці 4.6

Зміни у вимогах споживачів	Зміна пріоритетів і потреб клієнтів у сфері аналітики даних та засобів виявлення фейкових новин, що може призвести до зменшення попиту на поточні функціональні можливості продукту.	Підвищення рівня гнучкості продукту для адаптації до нових вимог клієнтів, швидке реагування на зміни потреб ринку через постійну модернізацію технологій і функцій.
Фінансові ризики	Обмеження в бюджетах потенційних клієнтів через економічні труднощі або несприятливі ринкові умови, що можуть зменшити попит на інструменти для аналізу даних.	Розробка бюджетних варіантів для малих бізнесів, гнучкі моделі цін для залучення широкої аудиторії, орієнтація на різні цінові сегменти.
Можливості		
Зростання популярності месенджерів	Месенджери, такі як Telegram, набирають популярності як канал для комунікацій і поширення інформації, що створює попит на інструменти для аналізу інформаційних потоків.	Активне позиціонування продукту як інструмента для аналізу Telegram, розвиток функціоналу для інших популярних платформ, розширення ринку.
Інновації в області обробки даних	Розвиток нових технологій для обробки великих обсягів даних, таких як штучний інтелект і машинне навчання, що дозволяє вдосконалити алгоритми аналізу інформації та виявлення мереж.	Впровадження новітніх технологій, зокрема AI та машинного навчання, для підвищення точності та швидкості аналізу даних, удосконалення алгоритмів.
Попит на аналітику та виявлення фейкових новин	Збільшення попиту на інструменти для виявлення фейкових новин і аналізу контенту через поширення інформаційних маніпуляцій та необхідність боротьби з дезінформацією.	Зосередження на розвитку інструментів для виявлення фейкових новин, інтеграція нових алгоритмів для покращення точності перевірки контенту, створення нових функцій.
Розвиток партнерств і інтеграцій	Потенціал для інтеграції з іншими платформами, аналітичними системами і інструментами, що розширює можливості стартапу та дозволяє залучити більше користувачів.	Розвиток партнерств з іншими технологічними компаніями, інтеграція з великими аналітичними системами та API для розширення функціональності продукту та доступу до нових ринків.
Нова хвиля інтересу до даних і їх аналізу	Підвищення інтересу до збору та аналізу великих даних, зокрема в бізнес-середовищі, що сприяє розвитку ринку аналітичних інструментів для автоматизації обробки даних.	Активне просування продукту серед бізнесів, пропозиція інструментів для збору та аналізу даних, включаючи аналітику на основі великих даних, автоматизація процесів для користувачів.

4.4 Розроблення ринкової стратегії проекту

Перед випуском товару на ринок, необхідно спроектувати потенційних клієнтів зацікавлених в використанні даного продукту

Таблиця 4.7 - Вибір цільових груп потенційних клієнтів

Опис профілю цільової групи потенційних клієнтів	**Готовність споживачів сприйняти продукт**	**Орієнтовний попит в межах цільової групи (сегменту)**	**Інтенсивність конкуренції в сегменті**	**Простота входу у сегмент**
Медіа-компанії, аналітичні агентства, журналісти	Висока готовність до прийняття інструментів для виявлення фейкових новин, аналізу даних і моніторингу каналів у месенджерах.	Середній до високого попиту, оскільки аналітика в медіа має високу актуальність через проблеми фейкових новин.	Висока: ринок заповнений різними аналітичними інструментами.	Середній: потрібно мати конкурентні переваги щодо точності та швидкості аналізу.
Державні органи, організації, що займаються безпекою	Помірна готовність через бюрократичні вимоги та потенційну недовіру до нових технологій.	Низький до середнього попиту, але вищий у контексті аналізу безпеки та інформаційних потоків.	Низька: організації часто мають обмежений вибір через специфічні вимоги до безпеки.	Складний: необхідно пройти через державні тендери, сертифікацію та відповідність нормам безпеки.
Малі та середні бізнеси в сферах e-commerce, маркетингу	Середня готовність сприймати нові технології для аналізу ефективності рекламних кампаній, аналізу конкурентів.	Середній попит через інтерес до автоматизації та покращення маркетингових стратегій.	Помірна: конкуренція з більшими постачальниками аналітичних рішень.	Легкий: менше обмежень для малого бізнесу, який може швидко інтегрувати інструменти.
IT-компанії та стартапи у сфері кібербезпеки	Висока готовність через постійний інтерес до нових технологій для моніторингу та захисту інформаційного простору.	Високий попит на інструменти для аналізу та виявлення мереж поширення інформації.	Висока: багато компаній розробляють схожі продукти для кібербезпеки.	Середній: необхідно бути на рівні з конкурентами в плані інновацій та безпеки.

Для стартапу найбільш перспективною є група **Медіа-компаній, аналітичних агентств та журналістів**, оскільки вона вже активно шукає інструменти для виявлення фейкових новин, моніторингу інформаційних потоків та аналізу даних у месенджерах. Готовність сприйняти продукт висока, попит на аналітичні інструменти для медіа зростає, а конкуренція, хоча й висока, зумовлена насиченістю ринку, дозволяє знайти нішу для спеціалізованих рішень з унікальними характеристиками.

Таблиця 4.8 - Визначення базової стратегії розвитку

Стратегія охоплення ринку	**Ключові конкурентоспроможні позиції відповідно до обраної альтернативи**	**Базова стратегія розвитку**
Стратегія диференціації	Унікальність продукту через точність і швидкість аналізу, можливість інтеграції з іншими системами, а також підвищена безпека обробки даних.	Зосередитися на інноваційних технологіях, які дозволяють досягати переваги в точності і швидкості виявлення фейкових новин та аналізу даних.

Для стартап-проекту найбільш підходящою базовою стратегією розвитку є **Стратегія диференціації**. Це обумовлено тим, що продукт має унікальні функціональні можливості, такі як інноваційні алгоритми для виявлення фейкових новин і аналізу інформаційних потоків у месенджерах, що дозволяє створювати конкурентні переваги

4.5 Проведення маркетингової програми стартап-проекту

Розроблення маркетингової програми є ключовим етапом запуску стартапу, що забезпечує позиціювання продукту на ринку та його конкурентоспроможність. Вона включає аналіз переваг продукту, визначення моделі, розроблення стратегії ціноутворення, системи збуту та маркетингових комунікацій. Основною метою є

відповідність потребам цільової аудиторії та формування привабливого позиціонування серед конкурентів.

Продукт спрямований на задоволення потреб медіа-компаній, аналітичних центрів, державних установ і бізнесу, що працює з інформаційними потоками. Унікальні можливості, такі як точність аналізу, швидкість обробки даних та інтеграція з іншими системами, створюють стійку конкурентну позицію.

Таблиця 4.9 використовується для формування унікальної ціннісної пропозиції.

Таблиця 4.9 - Визначення ключових переваг концепції потенційного товару

Потреба	**Вигода, яку пропонує товар**	**Ключові переваги перед конкурентами**
Аналіз поширення інформації в месенджерах	Швидке і точне виявлення мереж поширення контенту	Інноваційні алгоритми аналізу, висока точність результатів
Виявлення фейкових новин	Автоматизований аналіз на основі графових БД	Унікальні методики виявлення на основі часових і поведінкових даних
Інтеграція з іншими аналітичними системами	Простота інтеграції через API	Широкий спектр можливостей кастомізації

У таблиці 4.10 наведено трирівневу модель товару: базовий, реальний і розширений рівні. Вона допомагає зрозуміти, як продукт відповідає основним потребам клієнтів та які додаткові цінності створює.

Таблиця 4.10 - Опис трьох рівнів моделі товару

Рівні товару	**Сутність та складові**
Ядро товару	Основна функція продукту – аналіз і візуалізація поширення інформації в месенджерах
Реальний товар	Веб-додаток із дружнім інтерфейсом, підтримка Neo4j і Telegram API
Розширений товар	Додаткові сервіси: інтеграція з аналітичними платформами, технічна підтримка, навчання

Таблиця 4.11 аналізує рівні цін на аналоги та замінники, доходи цільової аудиторії та визначає верхню і нижню межі ціноутворення. Це необхідно для обґрунтування оптимальної цінової стратегії.

Таблиця 4.11 - Визначення меж встановлення ціни

Рівень цін на товари-замінники	**Рівень цін на товари-аналоги**	**Рівень доходів цільової групи споживачів**	**Верхня та нижня межі встановлення ціни на товар/послугу**
300-500 USD за місячний доступ	400-600 USD за місячний доступ	Середній/високий	350–550 USD за місячний доступ

Таблиця 4.12 використовується для визначення найбільш ефективного способу доставки продукту до споживачів.

Таблиця 4.12 - Формування системи збуту

Цільові клієнти	**Специфіка закупівельної поведінки**	**Функції збуту, які має виконувати постачальник товару**	**Глибина каналу збуту**	**Оптимальна система збуту**
Медіа-компанії, аналітичні центри	Пряма комунікація з постачальником	Надання консультацій, технічної підтримки, демонстрації продукту	Неглибокий (прямий збут)	Власна збутова мережа з акцентом на прямі продажі
Державні установи	Тривалий цикл прийняття рішень, орієнтація на безпеку	Інтеграція продукту, післяпродажна підтримка, відповідність законодавчим нормам	Помірний (через партнерів)	Комбінація власної збутової мережі та партнерів для підтримки локальних інтеграцій
Малі та середні бізнеси	Орієнтація на швидке прийняття рішення	Простота доступу до продукту, навчання персоналу	Неглибокий (прямий збут)	Онлайн-продажі через власну платформу, інтегровану з CRM-системами клієнтів

У таблиці описано поведінку клієнтів, канали комунікації, ключові позиції для позиціонування, завдання рекламного повідомлення та концепцію звернення. Це допомагає розробити ефективну стратегію комунікацій з цільовою аудиторією.

Таблиця 4.13 - Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	**Канали комунікацій, якими користуються цільові клієнти**	**Ключові позиції, обрані для позиціонування**	**Завдання рекламного повідомлення**	**Концепція рекламного звернення**
Орієнтація на інновації та технологічні рішення	Соціальні мережі, професійні конференції, вебінари	Інноваційність, швидкість аналітики	Демонстрація технологічних можливостей продукту	Акцент на автоматизацію процесів і унікальність
Інтерес до безпеки та надійності	Галузеві форуми, прямі переговори, спеціалізовані блоги	Захист даних, відповідність нормам	Підкреслення безпеки і відповідності стандартам	Технічна компетентність та надійність
Прагнення до оптимізації витрат і часу	Електронна пошта, цільова реклама, партнерські програми	Ефективність, доступність	Пояснення економії ресурсів	Простота інтеграції та економічна вигода

Висновки до розділу 4

Проведений аналіз показав, що проект має значний потенціал для ринкової комерціалізації. Попит на продукт підтверджується наявністю потреби в інструментах для аналізу інформаційних потоків та забезпечення кібербезпеки, зокрема в медіа-компаніях, аналітичних центрах та державних установах. Динаміка ринку вказує на зростаючу потребу в таких технологіях через постійну актуальність боротьби з фейковими новинами та інформаційними атаками. Рентабельність роботи на ринку є високою, оскільки продукт орієнтований на

задоволення конкретних потреб, і є потенціал для розширення цільової аудиторії.

Щодо перспектив впровадження, проект має хороші шанси на успіх, оскільки націлений на ключові групи клієнтів з високим попитом на подібні рішення. Однак існують певні бар'єри входження, зокрема високий рівень конкуренції та необхідність адаптації продукту до специфічних потреб клієнтів. Проте, завдяки унікальним можливостям продукту та здатності інтегруватися з іншими системами, проект має конкурентні переваги, що дозволяє йому зайняти стійку позицію на ринку.

Для ринкової реалізації проекту доцільно обрати стратегію поступового впровадження з акцентом на інтеграцію з уже існуючими інфраструктурами, що дозволить знизити бар'єри входження та забезпечити високий рівень довіри з боку потенційних клієнтів.

Подальша імплементація проекту є доцільною, оскільки продукт має потенціал для масштабування, а також відповідає вимогам сучасного ринку кібербезпеки та інформаційного аналізу. У разі правильної маркетингової стратегії, розвитку ефективних каналів збуту та належного брендування, проект може стати конкурентоспроможним на ринку і здобути стабільну частку ринку.

ВИСНОВКИ

В результаті виконання роботи було проведено комплексне дослідження методів аналізу потоків поширення інформації в месенджерах, зокрема Telegram, із застосуванням технологій графових БД. Завдання магістерської дисертації було виконано в повному обсязі, включаючи розробку системи для збору даних за допомогою Telegram API, створення моделі графової бази даних для ідентифікації явних та прихованих взаємозв'язків між каналами, а також оцінку ефективності автоматизованого підходу до аналізу інформаційних потоків. У процесі роботи було розроблено прототип програмного забезпечення, здатний здійснювати збір, обробку та аналіз даних у напівавтоматичному режимі.

Результати дослідження демонструють повноту виконання поставлених завдань:

1. **Огляд існуючих методів аналізу інформаційних потоків у месенджерах**. У процесі дослідження було проведено аналіз різних методів, застосовуваних для вивчення поширення інформації через месенджери, зокрема через Telegram. Розглянуті методи включають використання традиційних статистичних та графових підходів для побудови мереж розповсюдження, а також методи аналізу тексту та контенту. Особливу увагу було приділено вивченню обмежень цих методів, таких як складність обробки великих обсягів даних, необхідність адаптації до специфіки месенджерів та наявність прихованих зв'язків між каналами.
2. **Дослідження можливості використання Telegram API для збору даних**. Було досліджено документацію Telegram API, що дозволило зібрати необхідні дані для аналізу потоків інформації в месенджерах. Telegram API надає доступ до публічних каналів і чатів, що дозволяє отримувати інформацію про повідомлення, користувачів та репости. Завдяки API було реалізовано механізм

збору даних, який включає як отримання текстових повідомлень, так і метаданих, що дозволяють відстежувати зв'язки між каналами.

3. **Розробка архітектури системи для аналізу мереж поширення інформації**. Для реалізації аналізу потоків інформації була спроектована багаторівнева архітектура системи. Система включає компоненти для збору даних через Telegram API, збереження даних у реляційній базі (PostgreSQL) і графовій БД (Neo4j), а також модулі для обробки та аналізу зібраної інформації. Архітектура системи передбачає автоматизацію процесу збору та обробки даних, що забезпечує високу ефективність та масштабованість.
4. **Реалізація автоматизованої системи збору даних**. Було розроблено автоматизовану систему для збору інформації з Telegram, що включає збір повідомлень, репостів та метаданих каналів. Система дозволяє здійснювати регулярне оновлення даних та забезпечує інтеграцію з PostgreSQL для збереження первинних даних, а також з Neo4j для побудови графів зв'язків між каналами. Таким чином, автоматизація збору даних значно прискорила процес аналізу і дозволила працювати з великими обсягами інформації.
5. **Тестування розробленої системи на реальних даних та оцінка її ефективності**. Для тестування було використано реальні дані з публічних Telegram-каналів, що дозволило перевірити точність та ефективність системи. Тестування показало, що система успішно виявляє ключові канали, які сприяють поширенню інформації, та їх взаємозв'язки. Оцінка ефективності базувалась на кількісних показниках, таких як точність виявлення зв'язків між каналами та коректність побудови графів. Отримані результати підтвердили високу ефективність розробленої системи.

Серед досягнутих наукових результатів варто відзначити розробку методу виявлення явних і прихованих зв’язків між каналами на основі графових структур. Запропонований підхід дозволяє аналізувати ключові характеристики графу, такі як центральність вузлів, вагомість зв’язків та їх вплив на поширення інформації. В рамках дослідження було розглянуто особливості використання

орієнтованих графів для моделювання потоків у Telegram, а також оцінено ефективність запропонованих алгоритмів у порівнянні з традиційними методами аналізу. Основні результати роботи були представлені на науковій конференції та підготовлені до публікації у вигляді наукової статті, що свідчить про новизну та актуальність отриманих даних.

Подальші дослідження в цьому напрямку можуть включати розширення функціоналу системи для аналізу мультимедійного контенту, який також відіграє важливу роль у поширенні інформації. Іншим перспективним напрямком є впровадження алгоритмів машинного навчання для автоматичної класифікації типів інформаційних потоків та прогнозування шляхів їх поширення. Удосконалення методів виявлення прихованих зв’язків у графах, таких як слабко пов’язані компоненти, також сприятиме підвищенню точності аналізу.

Отримані результати мають широкий спектр застосувань у задачах забезпечення кібербезпеки, зокрема для моніторингу та виявлення джерел дезінформації, оцінки масштабів її поширення та розробки стратегій протидії. Запропонований підхід може бути інтегрований у системи кіберзахисту для ефективного реагування на загрози інформаційній безпеці, що є актуальним завданням сучасного інформаційного простору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Месенджери та реклама: статистика використання в Україні. *Digdata*. URL:https://digdata.com.ua/index.php/mesendzhery-ta-reklama-vykorystannya-v-ukrayini-2018-infohrafyka/.
2. Месенджери та реклама: статистика використання в Україні. *Digdata*. URL:https://digdata.com.ua/index.php/mesendzhery-ta-reklama-vykorystannya-v-ukrayini-2018-infohrafyka/.
3. Список інструментів поширення ворожої дезінформації. Центр Протидії Дезінформації. URL: https://cpd.gov.ua/reports/spysok-instrumentiv-poshyrennya-vorozhoyi-dezinformacziyi/
4. The Anatomy of the Facebook Social Graph. Johan Ugander, Brian Karrer, Lars Backstrom, Cameron Marlow. URL:https://arxiv.org/abs/1111.4503
5. Singh S. Rediscovering the Social Graph. *Medium*. URL: https://medium.com/@sameer_singh17/rediscovering-the-social-graph-8bd3960988ac
6. Слінько, Т. (2021). Сучасні загрози інформаційній безпеці країни та шляхи їх подолання. *Український часопис конституційного права*, (4), 77-84
7. Yan, G., Chen, G., Eidenbenz, S., & Li, N. (2011, March). Malware propagation in online social networks: nature, dynamics, and defense implications. In *Proceedings of the 6th acm symposium on information, computer and communications security* (pp. 196-206).
8. Silic, M., & Back, A. (2016). The dark side of social networking sites: Understanding phishing risks. *Computers in Human Behavior*, *60*, 35-43
9. Telegram API Documentation. *Telegram API Documentation*. URL: https://core.telegram.org.

10. Discord Developer Portal — API docs for bots and developers. (n.d.). Discord Developer Portal. URL: https://discord.com/developers/docs/reference
11. Robinson, Ian, James Webber, and Emil Eifrem. 2015. Graph Databases. Second edition. Beijing: O'Reilly. http://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=3564533.
12. Hodler, A. E., & Needham, M. (2022). Graph data science using Neo4j. In *Massive Graph Analytics* (pp. 433-457). Chapman and Hall/CRC.
13. Landed. V., & Snarskii, A. O. (2020). Мережі, що визначаються динамікою тематичних інформаційних потоків. *Реєстрація, зберігання і обробка даних*, *22*(1), 56-61.
14. Chyrun, L., Gozhyj, A., Evseeva, I., Don, D., Tyhonov, V., & Zakharchuk, M. (2019). Web Content Monitoring System Development. In *COLINS* (pp. 126-142).
15. Moffitt, K., Karabiyik, U., Hutchinson, S., & Yoon, Y. H. (2021, January). Discord forensics: The logs keep growing. In *2021 ieee 11th annual computing and communication workshop and conference (ccwc)* (pp. 0993-0999). IEEE.
16. Chauchard, S., & Garimella, K. Collecting WhatsApp Data for Social Science Research: Challenges and a Proposed Solution
17. Needham, M., & Hodler, A. E. (2018). A comprehensive guide to graph algorithms in neo4j. *Neo4j. com*.

ДОДАТОК А

```
from telethon import TelegramClient, errors, functions
from sqlalchemy import create_engine, text
from sqlalchemy.exc import IntegrityError
from sqlalchemy.orm import sessionmaker
from asyncio import run, sleep, Lock
from sqlalchemy import create_engine
from requests import get as r_get
from neo4j import GraphDatabase
from collections import Counter
from bisect import bisect_left
from bs4 import BeautifulSoup
from telethon.tl import types
from datetime import datetime
from re import findall, sub
from rapidfuzz import fuzz
from json import dump
from time import time
import pandas as pd
import traceback
import argparse
import sys
import os

uri = "bolt://localhost:7687"
username = "neo4j"
password = "password"
driver = GraphDatabase.driver(uri, auth=(username, password))
engine = create_engine('postgresql://kittenrage:123@localhost:5432/posts')
cols = ['channel_id', 'post_id', 'from_id', 'post_date',
        'post_author', 'post_text', 'media1_id', 'media1_size',
        'media1_weight', 'media1_height', 'media1_duration',
        'media1_filename', 'media2_id', 'media2_size',
        'media2_weight', 'media2_height', 'media2_duration',
        'media2_filename', 'fuzz_ratio_1', 'fuzz_ratio_2',
        'fuzz_ratio_3', 'fuzz_ratio_4', 'entity_type']
api_id = os.environ.get('TELEGRAM_API_ID_1')
api_hash = os.environ.get('TELEGRAM_API_HASH_1')
client = TelegramClient('session_name', api_id, api_hash)
takeout_lock = Lock()

def hidden_file_name():
    hidden_file_name_q = """
SELECT
    media_filename,
    JSON_AGG(
        JSON_BUILD_OBJECT('channel_id', channel_id, 'post_date', post_date)
        ORDER BY channel_id, post_date
    ) AS channel_date_pairs
FROM (
    SELECT
        channel_id, media1_size AS media_size, from_id, post_date,
        media1_weight AS media_weight, media1_height AS media_height,
        media1_duration AS media_duration, media1_filename AS media_filename
    FROM telegram_posts
    UNION ALL
    SELECT
        channel_id, media2_size AS media_size, from_id, post_date,
        media2_weight AS media_weight, media2_height AS media_height,
        media2_duration AS media_duration, media2_filename AS media_filename
    FROM telegram_posts
) AS combined_media
WHERE
    (channel_id IS NOT NULL OR media_size IS NOT NULL OR media_weight IS NOT NULL OR media_height IS NOT NULL OR media_duration IS NOT NULL)
    AND from_id IS NULL AND media_filename IS NOT NULL
GROUP BY media_filename
HAVING COUNT(DISTINCT channel_id) > 1;
"""
    groups = []
    df = pd.read_sql(hidden_file_name_q, con=engine).to_dict()
    for key in df['media_filename'].keys():
        if len([channel for channel in df['channel_date_pairs'][key]]) < 30 and not df['media_filename'][key].startswith('IMG_'):
            groups.append(list(set([channel['channel_id'] for channel in df['channel_date_pairs'][key]])))

    final_groups = []
    for i in groups:
        if not any(set(i).issubset(set(j)) for j in final_groups):
            final_groups.append(i)
    return final_groups
```

```
def hidden_file_meta():
    hidden_file_meta_q = """
SELECT
    media_size,
    media_weight,
    media_height,
    media_duration,
    JSON_AGG(
        JSON_BUILD_OBJECT('channel_id', channel_id, 'post_date', post_date)
        ORDER BY channel_id, post_date
    ) AS sorted_channel_date_pairs
FROM (
    SELECT
        channel_id, media1_size AS media_size, from_id,
        media1_weight AS media_weight, media1_height AS media_height,
        media1_duration AS media_duration, post_date
    FROM telegram_posts
    UNION ALL
    SELECT
        channel_id, media2_size AS media_size, from_id,
        media2_weight AS media_weight, media2_height AS media_height,
        media2_duration AS media_duration, post_date
    FROM telegram_posts
) AS combined_media
WHERE
    (channel_id IS NOT NULL OR media_size IS NOT NULL OR media_weight IS NOT NULL OR media_height IS NOT NULL OR media_duration IS NOT NULL)
    AND from_id IS NULL AND media_size IS NOT NULL AND media_weight IS NOT NULL AND media_height IS NOT NULL AND media_duration IS NOT NULL
GROUP BY media_size, media_weight, media_height, media_duration
HAVING COUNT(DISTINCT channel_id) > 1;
"""
    channels = []
    df = pd.read_sql(hidden_file_meta_q, con=engine)
    sorted_groups = [sorted(data, key=lambda x: x['post_date']) for data in df['sorted_channel_date_pairs'].to_list()]
    for group in sorted_groups:
        channels.append(list(set([post['channel_id'] for post in group])))

    final_groups = []
    for i in channels:
        if not any(set(i).issubset(set(j)) for j in final_groups):
            final_groups.append(i)
    return final_groups

def hidden_repost():
    groups = []
    test_q = """
SELECT
  ARRAY_AGG(post_text) AS texts,
  ARRAY_AGG(channel_id) AS channel_ids
FROM telegram_posts
WHERE post_text IS NOT NULL
  AND post_text <> ''
  AND fuzz_ratio_1 <> 0
  AND fuzz_ratio_2 <> 0
  AND fuzz_ratio_3 <> 0
  AND fuzz_ratio_4 <> 0
  AND from_id IS NULL
GROUP BY fuzz_ratio_1, fuzz_ratio_2, fuzz_ratio_3, fuzz_ratio_4
HAVING COUNT(DISTINCT channel_id) > 1
   AND MAX(LENGTH(post_text)) <= 1.1 * MIN(LENGTH(post_text));
"""
    unique_data = [list(x) for x in set(tuple(x) for x in pd.read_sql(test_q, con=engine)['channel_ids'].to_list())]
    groups = []
    for i in unique_data:
        if not any(set(i).issubset(set(j)) for j in groups):
            groups.append(i)
    return groups
```

```
def triangular_groups():

    test_q = """
SELECT
    t1.channel_id AS first_channel,
    t2.channel_id AS second_channel,
    t3.channel_id AS third_channel
FROM
    telegram_posts t1
JOIN
    telegram_posts t2 ON t1.from_id = t2.channel_id
JOIN
    telegram_posts t3 ON t2.from_id = t3.channel_id
WHERE
    t3.from_id = t1.channel_id
    AND t1.from_id != t2.from_id
    AND t2.from_id != t3.from_id
    AND t3.from_id != t1.from_id;
"""
    df = pd.read_sql(test_q, con=engine).drop_duplicates().values.tolist()

    groups = [sorted({item for lst in df if element in lst for item in lst})
              for element, _ in sorted(Counter([item for sublist in df for item in sublist]).items(), key=lambda x: x[1], reverse=True)]

    final_groups = []
    for i in groups:
        if not any(set(i).issubset(set(j)) for j in final_groups):
            final_groups.append(i)

    return final_groups

def create_node(tx, label, properties):
    if properties.get('name'):
        query = "MATCH (n {name: $name}) WITH n, labels(n) AS existing_labels WHERE NOT ANY(label IN existing_labels WHERE label = $label) RETURN n"
        result = tx.run(query, name=properties['name'], label=label)
        if result.peek(): pass
        else:
            query = f"MERGE (n:{label} {{name: $name}})"
            tx.run(query, name=properties['name'])
    else: pass

def update_node_label(tx, old_label, new_label, properties):
    query = f"MATCH (n:{old_label} {{name: $name}}) REMOVE n:{old_label} SET n:{new_label}"
    tx.run(query, name=properties['name'])

def create_relationship(tx, start_node, end_node, relationship_type, count=None):
    query = f"""MATCH (a:{start_node['label']}), (b:{end_node['label']})
                WHERE a.name = $start_name AND b.name = $end_name
                MERGE (a)-[r:{relationship_type}]->(b)
                ON MATCH SET r.count = COALESCE(r.count, 0) + $count
                ON CREATE SET r.count = $count"""
    tx.run(query, start_name=start_node['name'], end_name=end_node['name'], count=count)

def create_relationships_in_group(tx, channels, relationship_type):
    for i, channel_1 in enumerate(channels):
        for j, channel_2 in enumerate(channels):
            if i != j:
                query = f"MATCH (a:CHANNEL {{name: $name1}}), (b:CHANNEL {{name: $name2}}) MERGE (a)-[r:{relationship_type}]->(b) ON CREATE SET r.count = 1 ON MATCH SET r.count = r.count + 1"
                tx.run(query, name1=channel_1, name2=channel_2)

def get_all_nodes(tx):
    query = "MATCH (n) RETURN labels(n) AS labels, properties(n) AS properties"
    result = tx.run(query)
    nodes = []
    for record in result:
        nodes.append({
            "labels": record["labels"],
            "properties": record["properties"]
        })
    return nodes

def get_deadend_nodes(tx):
    query = "MATCH (c:CHANNEL) WHERE NOT EXISTS ((c)-[:REPOSTED]->()) RETURN c.name"
    result = tx.run(query)
    return [record["c.name"] for record in result]
```

```python
def n4j_runner(foo):
    with driver.session() as session:
        return session.execute_read(foo)

def emtpy_nodes(tx):
    query = "MATCH (n) WHERE NOT (n)--() RETURN n.name AS name"
    result = tx.run(query)
    nodes = []
    for record in result:
        nodes.append(record['name'])
    return nodes

def get_nodes_by_label(tx, label):
    query = f"MATCH (n:{label}) RETURN n.name AS name"
    result = tx.run(query)
    names = [record["name"] for record in result]
    return names

def build_graph(key, value):
    if key:
        with driver.session() as session:
            if isinstance(value, str):
                session.execute_write(update_node_label, "CHANNEL", value.upper(), {'name': key})
            elif isinstance(value, dict):
                session.execute_write(create_node, "CHANNEL", {'name': key})
                if value:
                    for channel in list(value.keys())[:-2]:
                        test_q = "SELECT entity_type FROM telegram_posts WHERE entity_type IN ('bot', 'small', 'user', 'chat' ) and channel_id='evorog_bot';"
                        df = pd.read_sql(test_q, con=engine).values
                        if df.size > 0:
                            session.execute_write(create_relationship, {'label': 'CHANNEL', 'name': key}, {'label': df[0][0].upper(), 'name': channel}, 'REPOSTED', value[channel])
                        else:
                            session.execute_write(create_node, "CHANNEL", {'name': channel})
                            session.execute_write(create_relationship, {'label': 'CHANNEL', 'name': key}, {'label': 'CHANNEL', 'name': channel}, 'REPOSTED', value[channel])
                    if value['web_domains']:
                        for domain in value['web_domains']:
                            session.execute_write(create_node, "DOMAIN", {'name': domain})
                            session.execute_write(create_relationship, {'label': 'CHANNEL', 'name': key}, {'label': 'DOMAIN', 'name': domain}, 'REFERENCES', value['web_domains'][domain])
                    if value['hashtags']:
                        for hashtag in value['hashtags']:
                            session.execute_write(create_node, "HASHTAG", {'name': hashtag})
                            session.execute_write(create_relationship, {'label': 'CHANNEL', 'name': key}, {'label': 'HASHTAG', 'name': hashtag}, 'TAGGED', value['hashtags'][hashtag])
                    else: pass

def create_data(channel_id, post_text, entity_type, from_id=None):
    data = {
            'channel_id': channel_id, 'post_id': 0,
            'from_id': from_id, 'post_date': None,'post_author': None, 'post_text': post_text,
            'media1_id': None, 'media1_size': None, 'media1_weight': None, 'media1_height': None,
            'media1_duration': None, 'media1_filename': None, 'media2_id': None, 'media2_size': None,
            'media2_weight': None, 'media2_height': None, 'media2_duration': None, 'media2_filename': None,
            'fuzz_ratio_1': fuzz.ratio(str(post_text), ''.join(['a' for _ in str(post_text)])),
            'fuzz_ratio_2': fuzz.ratio(str(post_text), ''.join(['e' for _ in str(post_text)])),
            'fuzz_ratio_3': fuzz.ratio(str(post_text), ''.join(['n' for _ in str(post_text)])),
            'fuzz_ratio_4': fuzz.ratio(str(post_text), ''.join(['r' for _ in str(post_text)])),
            'entity_type' : entity_type
            }
    df = pd.DataFrame([data], columns=cols)
    try: df.to_sql('telegram_posts', engine, index=False, if_exists='append', method='multi')
    except IntegrityError: pass

def postgres_execute(command):
    Session = sessionmaker(bind=engine)
    session = Session()
    try:
        session.execute(text(command))
        session.commit()
    except Exception as e:
        print(f"Error: {e}")
        session.rollback()
    finally:
        session.close()
```

```python
def get_title_by_hash(hash):
    response = r_get(f'https://t.me/{hash}')
    if response.status_code == 200:
        soup = BeautifulSoup(response.text, 'html.parser')
        meta_tag = soup.find('meta', property="og:title")
        title = soup.find('title')
        if 'chat' not in str(title).lower():
            if meta_tag:
                if 'Join group chat on Telegram' not in meta_tag.get('content'): return meta_tag.get('content')

def trim_ids(dict_ids):
    new_pairs, old_pairs = {}, {}
    known_ids = pd.read_sql("SELECT channel_id FROM id_to_username;", con=engine)['channel_id'].to_list()
    for key, value in dict_ids.items():
        if str(key) not in known_ids: new_pairs[key] = value
        else: old_pairs[key] = value
    return old_pairs, new_pairs

if not pd.read_sql("SELECT EXISTS ( SELECT 1 FROM information_schema.tables WHERE table_name = 'telegram_posts' AND table_schema = 'public');", con=engine)['exists'].to_list()[0]:
    postgres_execute("""CREATE TABLE IF NOT EXISTS telegram_posts (
    row_number SERIAL PRIMARY KEY, channel_id TEXT NOT NULL, post_id BIGINT NOT NULL,
    from_id TEXT, post_date TIMESTAMP, post_author VARCHAR(255),
    post_text TEXT, media1_id BIGINT, media1_size BIGINT,
    media1_weight FLOAT, media1_height FLOAT, media1_duration FLOAT,
    media1_filename VARCHAR(255), media2_id BIGINT, media2_size BIGINT,
    media2_weight FLOAT, media2_height FLOAT, media2_duration FLOAT,
    media2_filename VARCHAR(255), fuzz_ratio_1 FLOAT, fuzz_ratio_2 FLOAT,
    fuzz_ratio_3 FLOAT, fuzz_ratio_4 FLOAT, entity_type TEXT,
    CONSTRAINT channel_id_post_id_from_id UNIQUE (channel_id, post_id, from_id)
                );""")

if not pd.read_sql("SELECT EXISTS ( SELECT 1 FROM information_schema.tables WHERE table_name = 'id_to_username' AND table_schema = 'public');", con=engine)['exists'].to_list()[0]:
    postgres_execute("""CREATE TABLE IF NOT EXISTS id_to_username (
    row_number SERIAL PRIMARY KEY, channel_id TEXT NOT NULL, channel_name TEXT NOT NULL,
    CONSTRAINT uix_id_name UNIQUE (channel_id, channel_name)
                );""")

if not pd.read_sql("SELECT EXISTS ( SELECT 1 FROM information_schema.tables WHERE table_name = 'hash_to_data' AND table_schema = 'public');", con=engine)['exists'].to_list()[0]:
    postgres_execute("""CREATE TABLE IF NOT EXISTS hash_to_data (
    row_number SERIAL PRIMARY KEY, hash TEXT NOT NULL, title TEXT, username TEXT,
    CONSTRAINT uix_title_username UNIQUE (title, username)
                );""")

def process_forwarded_messages(post):
    try: return post.fwd_from.from_id.channel_id
    except AttributeError as err: pass
    except Exception as err: print(type(err), err)

def process_urls(post, message_urls_handled = False, hashtags_handled = False):
    urls = []
    hashtags = []
    CustomEmoji = []
    for entity in post.entities:
        if isinstance(entity, types.MessageEntityTextUrl):
            urls.append(entity.url.split('?')[0])
        elif isinstance(entity, types.MessageEntityUrl) and not message_urls_handled:
            urls.extend([url.split('?')[0] for url in findall(r'(https?://[^\s]+)', post.message)])
            message_urls_handled = True
        elif isinstance(entity, types.MessageEntityCustomEmoji):
            CustomEmoji.append(entity.document_id)
        elif isinstance(entity, types.MessageEntityHashtag) and not hashtags_handled:
            hashtags = [tag[1:] for tag in findall(r'#\w+', post.message)]
            hashtags_handled = True
    return list(filter(None, urls)), hashtags

def set_data_row(post, data_row):
    if post.fwd_from and post.fwd_from.from_id:
        try: from_id = post.fwd_from.from_id.channel_id
        except AttributeError: from_id = None
    else: from_id = None
    data_row[0] = post.peer_id.channel_id
    data_row[1] = post.id
    data_row[2] = from_id
    data_row[3] = post.date
    data_row[4] = post.post_author
```

```
def process_media(media, data_row):
    if isinstance(media, types.MessageMediaPhoto):
        data = media.photo
        data_row[6] = data.id
        data_row[7] = data.sizes[1].size
        data_row[8] = data.sizes[1].w
        data_row[9] = data.sizes[1].h
    elif isinstance(media, types.MessageMediaDocument):
        data = media.document
        attrs = [item for item in data.attributes if isinstance(item, types.DocumentAttributeFilename)]
        data_row[6] = data.id
        data_row[7] = data.size
        try:
            data_row[8] = data.attributes[0].w
            data_row[9] = data.attributes[0].h
        except AttributeError: pass
        try: data_row[10] = data.attributes[0].duration
        except AttributeError: pass
        if attrs: data_row[11] = attrs[0].file_name
        try:
            alt_data = media.alt_document
            alt_attrs = [item for item in data.attributes if isinstance(item, types.DocumentAttributeFilename)]
            data_row[12] = alt_data.id
            data_row[13] = alt_data.size
            try:
                data_row[14] = alt_data.attributes[0].w
                data_row[15] = alt_data.attributes[0].h
            except (AttributeError, IndexError): pass
            try: data_row[16] = alt_data.attributes[0].duration
            except AttributeError: pass
            if alt_attrs: data_row[17] = alt_attrs[0].file_name
        except AttributeError: pass

def process_messages(post, data_row):
    if post.message:
        text = str(sub(r'http[^\s]*', '', ''.join(e for e in ' '.join(post.message.lower().split()) if e.isalnum() or e == ' ')))
        if len(text) > 10:
            data_row[5] = text
            data_row[18] = fuzz.ratio(str(text), ''.join(['o' for _ in str(text)]))
            data_row[19] = fuzz.ratio(str(text), ''.join(['e' for _ in str(text)]))
            data_row[20] = fuzz.ratio(str(text), ''.join(['n' for _ in str(text)]))
            data_row[21] = fuzz.ratio(str(text), ''.join(['p' for _ in str(text)]))

def finalize_reposts(reposts, old_pairs, channels_data_by_id, ids_count, tags_count, domains_count, hashtags_count, channel):
    for tag, count in tags_count.items():
        try: create_data(channel_id=channel, post_text=None, entity_type=None, from_id=tag)
        except Exception as err: print(err)
        reposts[tag] = reposts.get(tag, 0) + count
    for data in channels_data_by_id:
        username = data.username or (data.usernames[0].username if data.usernames else None)
        if username:
            try:
                df = pd.DataFrame([[data.id, username]], columns=['channel_id', 'channel_name'])
                try: df.to_sql('id_to_username', engine, index=False, if_exists='append', method='multi')
                except IntegrityError: pass
            except Exception as err: print(err)
            reposts[username] = reposts.get(username, 0) + tags_count.get(data.id, 0) + ids_count.get(data.id, 0)
    for key, value in old_pairs.items():
        result = pd.read_sql(f"SELECT channel_name FROM id_to_username where channel_id='{key}';", con=engine)
        username = result['channel_name'].to_list()[0]
        reposts[username] = reposts.get(username, 0) + tags_count.get(username, 0) + ids_count.get(key, 0) + value
    reposts['web_domains'] = dict(domains_count)
    reposts['hashtags'] = dict(hashtags_count)

async def process_addlists(addlists, tags):
    for folder in addlists:
        while True:
            try:
                async with takeout_lock, client.takeout() as takeout:
                    lists = await takeout(functions.chatlists.CheckChatlistInviteRequest(slug=folder))
                    tags.extend([i.username for i in lists.chats if not isinstance(i, types.ChannelForbidden)])
                    break
            except errors.rpcbaseerrors.BadRequestError: break
            except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
                print('Must wait ar process_addlists',  e.seconds * 5, 'before takeout')
                await sleep(e.seconds * 5)
```

```python
async def fetch_channel_entities(dict_ids):
    async with takeout_lock, client.takeout() as takeout:
        channels_data_by_id = []
        while True:
            try:
                channels_data_by_id = await takeout.get_entity(list(dict_ids.keys()))
                break
            except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
                print('Must wait at fetch_channel_entities 1', e.seconds * 5, 'before takeout')
                await sleep(e.seconds * 5)
            except Exception:
                sorted_keys = sorted(dict_ids.keys())
                for key in sorted_keys:
                    index = bisect_left(sorted_keys, key)
                    if index < len(sorted_keys) and sorted_keys[index] == key:
                        while True:
                            try:
                                data = await takeout.get_entity(key)
                                channels_data_by_id.append(data)
                                break
                            except ValueError: break
                            except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
                                print('Must wait at fetch_channel_entities 2', e.seconds * 5, 'before takeout')
                                await sleep(e.seconds * 5)
                            except Exception as err: print(err)
                        break
        return channels_data_by_id

async def process_hashes(hashes):
    async with takeout_lock, client.takeout() as takeout:
        reposts = {}
        titles = []
        for hash, count in Counter(hashes).items():
            df = pd.read_sql(f"SELECT title, username FROM hash_to_data WHERE hash='{hash}'", con=engine).to_dict()
            if df['title'] or df['username']:
                if df['title']:
                    title = df['title']
                    if title: titles.append(title)
                if df['username']:
                    username = df['username']
                    reposts[username] = reposts.get(username, 0) + count
            else:
                try:
                    while True:
                        try:
                            hash_data = await takeout(functions.messages.CheckChatInviteRequest(hash=hash))
                            break
                        except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
                            print('Must wait at process_hashes', e.seconds * 5, 'before takeout')
                            await sleep(e.seconds * 5)
                    title, username = None, None
                    if hasattr(hash_data, 'chat'):
                        if hasattr(hash_data.chat, 'username') or hasattr(hash_data.chat, 'usernames'):
                            if hash_data.chat.username: username = hash_data.chat.username
                            elif hash_data.chat.usernames: username = hash_data.chat.usernames[0].username
                        if hasattr(hash_data.chat, 'title'):
                            if hash_data.chat.title: title = hash_data.chat.title
                    else: pass
                    try:
                        df = pd.DataFrame([[hash, title, username]], columns=['hash', 'title', 'username'])
                        df.to_sql('hash_to_data', engine, index=False, if_exists='append', method='multi')
                    except IntegrityError: pass
                    except Exception as err: print(err)
                    if username:
                        reposts[username] = reposts.get(username, 0) + count
                except (errors.InviteHashExpiredError, ValueError): continue
        return reposts, titles

async def fetch_entities_by_id_and_hashes(dict_ids, hashes):
    reposts = {}
    channels_data_by_id = []
    titles = []
    try:
        channels_data_by_id = await fetch_channel_entities(dict_ids)
        reposts, titles = await process_hashes(hashes)
    except (errors.ChannelPrivateError, errors.rpcerrorlist.UsernameInvalidError, errors.rpcerrorlist.InviteHashExpiredError, TypeError, ValueError): pass
    return reposts, channels_data_by_id, titles
```

```python
def categorize_urls(urls):
    hashes, addlists, tags, domains = [], [], [], []
    for url in urls:
        if 'http' in url: path_parts = url.split('/')[2:]
        else: path_parts = url.split('/')
        if len(path_parts):
            if path_parts[0] == "t.me":
                if path_parts[1].startswith('+'): hashes.append(path_parts[1][1:])
                elif path_parts[1] == "joinchat": hashes.append(url[-1])
                elif path_parts[1] == "addlist": addlists.append(path_parts[2])
                elif path_parts[1] == "boost": tags.append(path_parts[2])
                elif path_parts[1] in ["c", "addstickers", "addemoji"]: pass
                else: tags.append(path_parts[1])
            else: domains.append(path_parts[0])
    return hashes, addlists, tags, domains

def process_posts(all_posts):
    ids, urls, hashtags = [], [], []
    data = []
    for post in all_posts:
        if post.fwd_from and post.fwd_from.from_id:
            ids.append(process_forwarded_messages(post))
        if post.entities:
            procceses_urls, hashtags = process_urls(post)
            urls.extend(procceses_urls)
        if post.media or post.message:
            data_row = [None] * len(cols)
            set_data_row(post, data_row)
            if post.message:
                process_messages(post, data_row)
            if post.media:
                process_media(post.media, data_row)
            data_row[-1] = 'channel'
            data.append(data_row)
    df = pd.DataFrame(data, columns=cols)
    try: df.to_sql('telegram_posts', engine, index=False, if_exists='append', method='multi')
    except IntegrityError:
        for _, row in df.iterrows():
            try:
                df_row = pd.DataFrame([row], columns=cols)
                df_row.to_sql('telegram_posts', engine, index=False, if_exists='append', method='multi')
            except IntegrityError: pass
    return ids, urls, hashtags

async def fetch_channel_data(channel, limit, deepen=False):
    while True:
        try:
            async with takeout_lock, client.takeout() as takeout:
                newest_id = pd.read_sql(f"SELECT MAX(post_id) FROM telegram_posts WHERE channel_id='{channel}'", con=engine)['max'].to_list()[0]
                if deepen:
                    oldest_id = pd.read_sql(f"SELECT MIN(post_id) FROM telegram_posts WHERE channel_id='{channel}' AND post_id<>0", con=engine)['min'].to_list()[0]
                    try:
                        all_posts = await takeout.get_messages(channel, limit=limit, offset_id=oldest_id)
                    except TypeError:
                        if newest_id: all_posts = await takeout.get_messages(channel, limit=limit, min_id=newest_id)
                        else: all_posts = await takeout.get_messages(channel, limit=limit)
                else:
                    if newest_id: all_posts = await takeout.get_messages(channel, limit=limit, min_id=newest_id)
                    else: all_posts = await takeout.get_messages(channel, limit=limit)
                return all_posts
        except ValueError: break
        except errors.rpcerrorlist.UsernameInvalidError: break
        except errors.rpcerrorlist.UsernameNotOccupiedError: break
        except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
            print('Must wait at fetch_channel_data', e.seconds * 5, 'before takeout')
            await sleep(e.seconds * 5)
        finally: all_posts = []

    return all_posts
```

```python
async def check_channel(channel):
    try:
        if pd.read_sql(f"select count(channel_id) from telegram_posts where channel_id='{channel}'", con=engine)['count'].item(): return None
        if channel.lower().endswith('bot'):
            try:
                data = create_data(channel_id=channel, post_text=None, entity_type='bot')
            except Exception as err: print(err)
            return 'bot'
    except Exception as err: print("some problem reading postgres", err)
    try:
        async with takeout_lock, client.takeout() as takeout:
            while True:
                try:
                    channel_details = await takeout(functions.channels.GetFullChannelRequest(channel))
                    break
                except (errors.TakeoutInitDelayError, errors.FloodWaitError) as e:
                    print('Must wait at check_channel', e.seconds * 5, 'before takeout')
                    await sleep(e.seconds * 5)
            try:
                df = pd.DataFrame([[channel_details.full_chat.id, channel]], columns=['channel_id', 'channel_name'])
                df.to_sql('id_to_username', engine, index=False, if_exists='append', method='multi')
            except IntegrityError: pass
            except Exception as err: print(err)
            if channel_details.full_chat.default_send_as:
                try: data = create_data(channel_id=channel, post_text=None, entity_type='chat')
                except Exception as err: print(err)
                finally: return 'chat'
            if channel_details.full_chat.participants_count < 1000:
                try: data = create_data(channel_id=channel, post_text=None, entity_type='small')
                except Exception as err: print(err)
                finally: return 'small'
            try:
                desc = ' '.join(channel_details.full_chat.about.splitlines())
                data = create_data(channel_id=channel, post_text=desc, entity_type='channel')
            except Exception as err: print(err)
            urls = [url.split('?')[0] for url in findall(r'(\b(?:https?://)?[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+(?:/[^\s]*)?)', desc)]
            tags = [tag[1:] for tag in findall(r'@\w+', desc)]
            data = channel_details.chats[0]
            username = data.username or (data.usernames[0].username if data.usernames else None)
            df = pd.DataFrame([[data.id, username]], columns=['channel_id', 'channel_name'])
            try: df.to_sql('id_to_username', engine, index=False, if_exists='append', method='multi')
            except IntegrityError: pass
            return tags, urls
    except (errors.rpcerrorlist.UsernameInvalidError, ValueError): pass
    except TypeError as err:
        if 'InputPeerUser' in str(err):
            try: data = create_data(channel_id=channel, post_text=None, entity_type='user')
            except Exception as err: print(err)
            finally: return 'user'
        else: pass
    return None

async def process_channel(channel, limit, deepen):
    print('-'*60, channel, '-'*(80-len(channel)))
    tags_from_desc, urls_from_desc, hashes, addlists, tags, domains = [], [], [], [], [], []
    check_result = await check_channel(channel)
    if not check_result: pass
    elif isinstance(check_result, str):
        build_graph(channel, check_result)
        return check_result
    else: tags_from_desc, urls_from_desc = check_result
    all_posts = await fetch_channel_data(channel, limit, deepen)
    ids, urls, hashtags = process_posts(all_posts)
    try: urls.extend(urls_from_desc)
    except Exception as err: print(err)
    hashes, addlists, tags, domains = categorize_urls(urls)
    old_pairs, new_pairs = trim_ids(Counter(ids))
    reposts, channels_data_by_id, titles = await fetch_entities_by_id_and_hashes(new_pairs, hashes)
    tags.extend(titles)
    try: tags.extend(tags_from_desc)
    except Exception as err: print(err)
    await process_addlists(addlists, tags)
    finalize_reposts(reposts, old_pairs, channels_data_by_id, Counter(ids), Counter(tags), Counter(domains), Counter(hashtags), channel)
    if channel in reposts: del reposts[channel]
    build_graph(channel, reposts)
    return reposts
```

```
async def build(entry_points, limit=100, depth=3, deepen=False):
    print(f"{'-'*20} Обробка каналів для {entry_points} {'-'*20}")
    await client.start()
    reposts = {}
    for entry_point in entry_points:
        print(f"{'-'*20} {entry_point} {'-'*20}")
        new_keys = [entry_point]
        for level in range(depth):
            results = []
            print(new_keys)
            start_time = time()
            for key in new_keys:
                if (key not in reposts) or (not reposts[key].keys()):
                    result = await process_channel(key, limit, deepen)
                    results.append(result)
            print(f"{level} рівень завершено, {len(new_keys)} каналів оброблено --- {time() - start_time:.2f} секунд ---")
            reposts.update({key: result for key, result in zip(new_keys, results)})
            new_keys = list(set([key for result in results if isinstance(result, dict) for key in list(result.keys())[:-2] if key not in reposts]))
            for key in new_keys:reposts[key] = {}
            print(f"{len(new_keys)} каналів на наступному рівні")
            with open(f"all_entry_points_sample_{str(datetime.now()).replace(' ', '_')}.json", "w", encoding="utf-8") as outfile:
                dump(reposts, outfile, indent=4, ensure_ascii=False)

async def deepen(entry_points, limit = 500, depth = 3, deepen=True):
    await build(entry_points, limit, depth, deepen)

async def extend():
    deaddend_nodes = n4j_runner(get_deadend_nodes)
    for node in deaddend_nodes:
        await build([node], limit=100, depth=1)

async def analyze():
    print('\n', '-'*60, 'Виявлення прихованих зв'язків через тріангуляцію каналів' ,'-'*60, '\n')
    groups = triangular_groups()
    for group in groups:
        with driver.session() as session:
            session.execute_write(create_relationships_in_group, group, "TRIANGULAR_GROUPS")

    print('\n', '-'*60, 'Ідентифікація прихованих зв'язків через текстові повідомлення' ,'-'*60, '\n')
    hidden_repost_groups = hidden_repost()
    for group in hidden_repost_groups:
        with driver.session() as session:
            session.execute_write(create_relationships_in_group, group, "HIDDEN_REPOST")

    print('\n', '-'*60, 'Виявлення прихованих зв'язків через метадані медіафайлів' ,'-'*60, '\n')
    hidden_file_meta_groups = hidden_file_meta()
    for group in hidden_file_meta_groups:
        with driver.session() as session:
            session.execute_write(create_relationships_in_group, group, "HIDDEN_FILE_META")

    print('\n', '-'*60, 'Виявлення прихованих зв'язків на основі медіафайлів' ,'-'*60, '\n')
    hidden_file_name_groups = hidden_file_name()
    for group in hidden_file_name_groups:
        with driver.session() as session:
            session.execute_write(create_relationships_in_group, group, "HIDDEN_FILE_NAME")
    pass


try:
    parser = argparse.ArgumentParser(description="Обробка даних з Telegram каналів.")
    parser.add_argument('-a', '--analyze', action='store_true', help="аналіз мережі")
    parser.add_argument('-b', '--build', action='store_true', help="побудова мережі")
    parser.add_argument('-d', '--deepen', action='store_true', help="поглиблення мережі")
    parser.add_argument('-e', '--extend', action='store_true', help="розширення мережі")
    parser.add_argument('entry_points', nargs='*', help="Список точок входу для обробки.")
    args = parser.parse_args()
    entry_points = [arg for arg in sys.argv[1:] if '-' not in arg]
    if args.build:
        run(build(entry_points))
    elif args.deepen:
        run(deepen(entry_points))
    elif args.analyze:
        run(analyze())
    elif args.extend:
        run(extend())
except Exception as err:
    print(err)
    traceback.print_exc()
finally:
    df = pd.read_sql('select channel_id, channel_name from id_to_username', con=engine).to_dict()
    for key in list(df['channel_id'].keys()):
        try: postgres_execute(f"UPDATE telegram_posts SET channel_id='{df['channel_name'][key]}' WHERE channel_id='{df['channel_id'][key]}';")
        except: pass
        try: postgres_execute(f"UPDATE telegram_posts SET from_id='{df['channel_name'][key]}' WHERE from_id='{df['channel_id'][key]}.0';")
        except: pass
```