КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «магістр»

НА ТЕМУ:

Розробка та дослідження методів генерації запитань до українськомовних текстів

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітньо-наукова програма «Технології штучного інтелекту»

Виконала:
студентка 2 курсу магістратури, групи ТШІ-21 — Дуліч Олександра Романівна

Науковий керівник: — Тмєнова Наталія Пилипівна
Кандидат фізико-математичних наук, доцент

Засвідчую, що в цій кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студентка _____
підпис

Кваліфікаційна робота допущена до захисту
рішенням кафедри *інтелектуальних технологій*

Протокол № ____ від «____» травня 2021 р.

Зав. кафедри _____ доц. Іларіонов О.Є.
підпис

Київ 2021

ВСТУП

Ми живемо в часи, коли обсяги згенерованої людством інформації більші, ніж будь-коли, і кількість цих даних зростає з кожним днем. Однак значну користь з цієї інформації можна отримати лише при правильній обробці і аналізі цих даних. З іншого боку, кількість задач, яку людство довіряє розв'язувати комп'ютерам, зростає з неймовірним темпом. Усе більше й більше процесів піддається автоматизації. Таким чином, постають проблеми комп'ютерного аналізу та синтезу природної мови, а потреба в удосконаленні методів обробки природної мови зростає та користується невичерпним інтересом. Особливо актуальним це є для українських реалій, оскільки для обробки української мови, на жаль, не вистачає таких інструментів, як бібліотек для мов програмування, розмічених корпусів, словників, тезаурусів тощо.

Генерація запитань до тексту — завдання генерації природної мови, що має життєво важливе значення для самостійного навчання. Отже, актуальність даної роботи обумовлена невичерпним інтересом та зростаючою потребою в удосконаленні обробки природної мови. Особливо актуальним це є для українських реалій, адже для розвитку даного напрямку для української мови було зроблено не так багато.

Метою даної роботи є дослідження й порівняння реалізації модифікованого методу генерації запитань на основі текстового корпусу та модифікованого методу за шаблонами, визначення їх ефективності.

Об’єкт дослідження – процес автоматичної генерації запитань до українськомовних текстів.

Предмет дослідження – методи та підходи генерації запитань до українськомовних текстів.

Наукова новизна роботи:

- реалізовано систему автоматичної генерації запитань до українськомовних текстів за допомогою модифікованого методу генерації запитань на основі текстового корпусу та модифікованого методу за шаблонами;

- надано порівняльну характеристику реалізації використаних методів генерації запитань для системи автоматичної генерації запитань та оцінено ефективність кожного.

Практичним результатом роботи є реалізація системи автоматичної генерації запитань до українськомовних текстів з використанням модифікованого методу генерації запитань на основі текстового корпусу та модифікованого методу за шаблонами.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Комп’ютерна лінгвістика та задача генерації запитань

Комп’ютерна лінгвістика – напрям в прикладній лінгвістиці, орієнтований на використання комп’ютерних інструментів – програм, комп'ютерних технологій організації та обробки даних – для моделювання функціонування мови в тих чи інших умовах, ситуаціях, проблемних сферах тощо, а також вся сфера застосування комп'ютерних моделей мови в лінгвістиці та суміжних дисциплінах.

До комп'ютерної лінгвістики певною мірою можуть бути віднесені роботи в області створення гіпертекстових систем, що розглядаються як особливий спосіб організації тексту і навіть як принципово новий вид тексту, протиставлений за багатьма своїми властивостями звичайного тексту. До компетенції комп’ютерної лінгвістики відноситься й автоматичний переклад.

В рамках комп’ютерної лінгвістики виник порівняно новий напрям, що активно розвивається з 1980-90-х років, – корпусна лінгвістика, де розробляються загальні принципи побудови лінгвістичних корпусів даних (зокрема, корпусів текстів) з використанням сучасних комп’ютерних технологій. Корпуси текстів – це колекції спеціально підібраних текстів книг, журналів, газет тощо, перенесені на машинні носії і призначені для автоматичної обробки. Один з перших корпусів текстів був створений для американського варіанту англійської мови в Браунівському університеті, так званий Браунський корпус, в 1962-1963 під керівництвом У. Френсіса. Крім конструювання корпусів даних, корпусна лінгвістика займається створенням

комп’ютерних інструментів і програм, призначених для вилучення різноманітної інформації з текстових корпусів. З точки зору користувача, до корпусів текстів висуваються вимоги репрезентативності, повноти і економічності.

Однією з актуальних задач комп'ютерної лінгвістики є автоматична генерація запитань природною мовою [1]. Системи, що мають подібну функціональність, як правило, використовуються в сфері освіти для перевірки знань учнів, а саме при складанні запитань з теоретичного матеріалу [2].

Створення QA-систем – це особливий тип інформаційних систем, які є гібридом пошукових та інтелектуальних систем (часто вони розглядаються як інтелектуальні пошукові системи). QA-система повинна бути здатна приймати запитання природною мовою, тобто це система з природномовним інтерфейсом. Інформація надається на основі документів з мережі Інтернет або з локального сховища. Сучасні розробки QA-систем дозволяють обробляти множину варіантів запитів фактів, списків, дефініцій, запитань типу Як, Чому, гіпотетичних, складних та міжмовних.

1.2. Аналіз сучасного стану питання. Обґрунтування актуальності обраної теми

Ідея автоматичного генерування запитань з речень була запропонована та реалізована ще в 1976 р. Оскільки технології синтаксичного аналізу на той час ще не були зрілими, замість синтаксичного аналізу використовувалось узгодження рядків [3].

У 1993 році Кунічіка та ін. розробили мультимедійну систему навчання мови для викладання англійської мови японським студентам [4]. Система є

інтегрованим середовищем для авторів підручників та студентів. Система обробляє дані, які автори вводять через компонент NLP, та надає певні вказівки під час використання студентом системи. Однією з інтелектуальних функцій системи є автоматичне створення тестів на розуміння на основі текстового матеріалу. Детальний підхід описаний у наступній роботі [5]. Запитання генерувалися до змісту тексту, спочатку аналізуючи англійські речення на рівні семантики, а потім замінюючи одне слово чи фразу питальним словом, що відповідає цьому конкретному семантичному класу, і, нарешті, змінюючи речення на форму запитання. Хоча викладач задає запитання усно, студент може відповісти на них лише текстом.

Юші Сю, Анна Голді, Стефані Сенефф розробили досить визначні технології обробки мови. Вони використали систему розуміння мови TINA [6] та систему генерації мови GENESIS [7] для різних застосувань. Дві системи можна каскадувати через представлення інтерлінгви, для чого дослідники використовують термін "лінгвістичним фрейм", тобто TINA створює лінгвістичний фрейм із рядка, а GENESIS генерує рядок із фрейму. Ученими були розроблені загальні граматики та правила генерації як для англійської, так і для китайської мови. Залежно від вибору граматики та правил генерації, каскадні системи можуть виконувати переклад на іншу мову або міжмовний переклад. Група ентузіастів розробила мовні ігри з перекладом [8], використовуючи ці системи. На рисунку 1.1 представлена загальна структура гри.

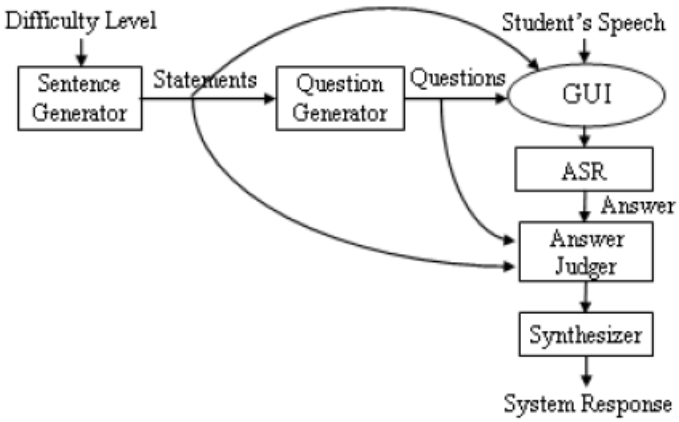

Рисунок 1.1 – Загальна структура гри [9]

Генератор речень створює список паралельних тверджень відповідно до поточного рівня складності з використанням рекурсивної граматики. Для системи був розроблений спеціальний синтаксис з прив'язаними правилами для кодування двомовної лексики у шаблонах уроків, а також для забезпечення різних порядків слів в англійських та китайських реченнях. Твердження у списку складаються випадковим чином як з поточного рівня, так і з попередніх рівнів. Система відображає китайські речення та використовує їх для генерації запитань. Англійська половина використовується лише тоді, коли студент звертається за допомогою. Генератор запитань випадковим чином вибирає одне китайське твердження, генерує запитання та відправляє його до графічного інтерфейсу для синтезу та відтворення. Коли студент вимовляє свою відповідь, розпізнавач мови фіксує звукову форму і перетворює її в текст. На основі початкового твердження та запитання суддя, який відповідає за відповіді, вирішує, чи є відповідь студента правильною чи ні.

Генерація запитань у галузі обчислювальної лінгвістики привертає величезну увагу дослідників. Двадцять років тому потрібні були години чи тижні, щоб отримати відповіді на ті самі запитання, що і людина, за якою полювали за документами в бібліотеці. У майбутньому електронні підручники та джерела інформації будуть загальнодоступними, і вони будуть

супроводжуватися складними засобами для запитань та відповідей. Як результат, вважається, що покоління Google має більш допитливий розум, ніж покоління, які покладались на пасивне читання та бібліотеки [10]. В останні кілька років з’являються нові проблеми для автоматичного формування запитань. У роботі Andrenucci та Shneiders було запроваджено підхід за шаблонами для створення запитань щодо чотирьох типів сутностей [11]. McGough та інші використовували WTML (Web Testing Markup Language), що є розширенням HTML, для вирішення проблеми представлення студентам динамічно згенерованих іспитів в браузері зі значним інженерним математичним змістом [12]. Wang та його колеги автоматично генерували запитання за шаблонами запитань, що створюються шляхом навчання за багатьма медичними статтями [13]. Цікавий підхід до автоматичного формування запитань для оцінки словникового запасу був описаний Брауном та іншими дослідниками [14].

QA-системи, які забезпечують відповіді природною мовою на запити природною мовою, є предметом швидко прогресуючих досліджень, що охоплюють як академічне навчання, так і комерційні програми, найвідоміша з яких – пошукова система Ask Jeeves. Відповіді на запитання спираються на різні галузі та технології, включаючи обробку природною мовою, пошук інформації, генерацію пояснень та взаємодію людини з комп’ютером. Відповіді на запитання створюють важливий новий метод доступу до інформації і можуть розглядатися як природний крок за межі таких стандартних методів веб-пошуку, як запит ключових слів та пошук документів.

З моменту появи перших прототипів QA-систем їх область застосування значно розширилася [15]. Наприклад, їх використовують у відповідях на запитання, пов'язані з часом, геолокаційні запитання, запитання визначення

понять, бібліографічні, багатомовні запитання, запитання, пов'язані з мультимедіа (візуальною, аудіо- та відео- інформацією). Вивчаються додаткові області: побудова інтерактивних систем QA, повторне використання відповідей і уявлення знань, використання виведення з наявною інформацією для отримання відповідей на запитання тощо, прогнозування запитань, які можуть бути задані, аналіз настрою.

Інтернет-користувачам все важче й важче орієнтуватися у великій кількості наявної в даний час інформації, а отже потреба в автоматизованих системах відповідей на запитання стає більш актуальною. Нам потрібні системи, які дозволяють користувачеві задавати запитання повсякденною мовою та отримувати відповідь швидко та стисло, з достатнім контекстом для підтвердження відповіді. Зрозуміло, що сучасні пошукові системи можуть повертати рейтингові списки документів, але вони не дають відповіді користувачеві [16].

Пандемія 2020-го року змінила темп і образ життя всієї планети: маска стала невід’ємним аксесуаром, а навчання та робота перемістилися з офісів та учбових закладів до осель працівників та учнів. Але дистанційний формат навчання тягне за собою більшу кількість перевірок знань і зусиль з боку викладачів, яким необхідно вдаватися до нових методів контролю в сучасних реаліях. Викладачам доводиться витрачати багато часу на підготовку онлайн-матеріалів, в тому числі і тестів, зі створенням яких випливають нові проблеми, насамперед це різноманіття типів запитань тестів і час, який буде затрачено на їх створення. У сучасному світі час є головною проблемою. Будь-який продукт, який може ефективно зменшити час та енергоспоживання, приймається та цінується. Система автоматичної генерації запитань володіє перерахованими функціями і звертається до вирішення проблеми нестачі часу,

а отже є дуже актуальною, а саме в українськомовних реаліях, адже подібних систем не існує.

Генератори запитань і тестових завдань стають важливим елементом системи контролю знань у ВНЗ. Вони використовуються при видачі індивідуальних завдань студентам, проведенні тестових іспитів і контрольних робіт, створення різних тренажерів, комп'ютерних навчальних програм і мультимедійних підручників.

Якщо для англійської мови дані збиралися та оброблялися протягом останніх 60 років, то для української мови тільки зараз можна спостерігати за розвитком обробки природної мови та появою нових інструментів та корпусів саме для української мови: Браунський корпус української мови – відкритий, збалансований за жанрами та в майбутньому проанотований корпус сучасної української мови, який побудований на засадах, що були покладені в основу корпусу англійської мови Brown; ner-uk – NER-анотація українського корпусу; ВЕСУМ – великий електронний словник української мови, що містить слова та їхні парадигми з відповідними тегами.

Таким чином, реалізація системи автоматичної генерації запитань до українськомовних текстів буде гарним поштовхом для подальших розробок в даному напрямі, в тому числі й для реалізації QA-систем.

1.3. Проблеми реалізації систем автоматичної генерації запитань

Дослідники описують низку проблем, з якими можна зіткнутися при реалізації системи автоматичної генерації запитань. Одна з них – це визначення типів запитань та методи їх побудови, адже різні типи запитання потребують відмінних підходів до генерації структури запитання.

Обробка згенерованих запитань є наступною проблемою, оскільки до одного тексту можна поставити відмінні запитання за структурою, але не за значенням.

Отже, постає необхідність створення ефективних методів розуміння та обробки семантики речення. Важливо, щоб програма була здатна розпізнавати еквівалентні за змістом запитання, незалежно від використаних слів, стилю, синтаксичних взаємозв'язків та ідіом. Система повинна розділяти складні запитання на кілька простих і правильно трактувати контекстно-залежні фрази. Один і той же інформаційний запит може бути виражений різними способами. Потрібна семантична модель розуміння та обробки запитань, яка б розпізнавала еквівалентні запитання, незалежно від мовленнєвого акту чи слів, синтаксичних взаємозв’язків чи ідіоматичних форм. Ця модель дозволить перекласти складне запитання на низку простих запитань, визначить неоднозначності та розгляне їх у контексті або за допомогою інтерактивних роз’яснень [17].

Центральним вузлом генератора є база знань, яка забезпечує формалізоване уявлення знань деякої предметної області. Структура цієї бази визначає основні характеристики генератора: варіантність, складність, надійність, перерахунок. Варіантність визначає загальне число запитань і тестових завдань, яке може бути отримано даними генератором. Надійність визначає коректність завдань, що генеруються, тому необхідно мати генератор, який генерує тільки коректні завдання. Перерахунок визначає можливість однозначної відповідності між деякими номером і конкретним запитанням, отриманим генератором.

Але основна проблема автоматичної генерації запитань до українськомовних текстів полягає в тому, що перед тим як це запитання

згенерувати, потрібно пройти ряд перетворень зі вхідним текстом для того, щоб машина зрозуміла людську мову, а вже потім генерувала запитання.

Не виникає жодної проблеми виконати ці дії для англійської мови, а ось з українською вже виникають складності. Якість розуміння залежить від безлічі факторів: від мови, від національної культури, від самого співрозмовника тощо. Ось деякі приклади складнощів, з якими стикаються системи розуміння текстів:

1) складнощі з розкриттям анафор (розпізнаванням, що мається на увазі при використанні займенників): речення "Ми віддали банани мавпам, тому що вони були голодні" і "Ми віддали банани мавпам, тому що вони були перестиглі" схожі за синтаксичною структурою. В одному з них займенник "вони" відноситься до мавп, а в іншому – до бананів. Правильне розуміння залежить від знань комп'ютера, якими можуть бути банани і мавпи;
2) вільний порядок слів може призвести до абсолютно іншого пояснення фрази: "Буття визначає свідомість" - що визначає що?
3) в українській мові вільний порядок компенсується розвиненою морфологією, службовими словами і знаками пунктуації, але в більшості випадків для комп'ютера це створює додаткову проблему.
4) У промові можуть зустрітися неологізми. Система повинна вміти відрізняти такі випадки від помилок і правильно їх розуміти;
5) Правильне розуміння омонімів – ще одна проблема. При розпізнаванні мови, крім інших, виникає проблема фонетичних омонімів.

Система автоматичної генерації запитань на основі певного набору документів повинна згенерувати до них найбільш повний список запитань і варіантів відповідей. При цьому основна складність полягає в складанні складних за структурою запитань [18]. Як правило, такі запитання

ґрунтуються на інформації з різних частин тексту, яка пов'язана між собою тільки за змістом. Існуючі системи не дозволяють складати складні запитання, оскільки дана задача тягне за собою безліч складнощів і проблем, яких можна уникнути при складанні простих запитань з декількох слів і на основі одного речення.

1.4. Методи генерації запитань

Загалом, зустрічаються наступні методи генерації запитань в існуючих дослідженнях:

- методи, засновані на деревах І/АБО;
- методи генерації запитань за шаблонами;
- методи, засновані на перебудові речення;
- методи на основі текстового корпусу.

Розглянемо детальніше ці методи, щоб визначитися які з них можуть бути застосовані для розв’язання задач, поставлених у рамках даного дослідження.

1.4.1. Метод генерації запитань на основі дерев І/АБО

Метод генерації запитань на основі дерев І/АБО дозволяє автоматично генерувати набір тестових запитань заданого типу. Він широко використовується при складанні завдань для підручників [19]. Його особливістю є використання вже наявних даних про структуру запитання, що

дозволяє складати складні запитання, які складаються з необмеженого числа слів і речень.

Дерево І/АБО – дерево, що складається з вузлів двох типів: І-вузол (рис. 1.2) і АБО-вузол (рис. 1.3).

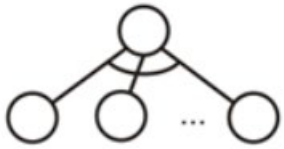

Рисунок 1.2 – І-вузол

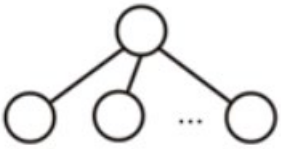

Рисунок 1.3 – АБО-вузол

Варіантом дерева І/АБО називається дерево, отримане з даного шляхом відсікання всіх дуг, крім однієї, у АБО-вузлів. Коренем варіанту буде корінь вихідного дерева (рис 1.4).

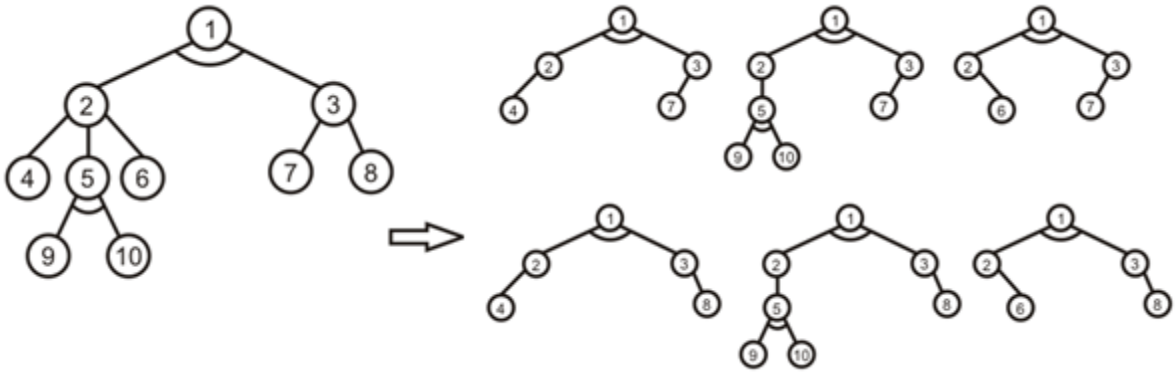

Рисунок 1.4 – дерево І / АБО і все його варіанти

Потужністю дерева І/АБО називається кількість варіантів, яке воно містить. Таким чином, потужність дерева, представленого на рисунку, дорівнює 6.

Ще однією чудовою властивістю дерева І/АБО є можливість ідентифікації варіанту (отримання варіанту за його номером) [19].

Метод включає в себе наступні етапи:

1. Побудова дерева І/АБО, що описує структуру запитання: усі вершини дерева позначаються як постійні або ті, що змінюються. У постійних вузлах міститься незмінна інформація, а для змінних вузлів задається множина можливих значень.
2. Обхід дерева з метою перебору всіх можливих комбінацій його вузлів: до кожної комбінації застосовуються певні алгоритми, що перевіряють її коректність, тобто використання відповідних до змісту запитань числових значень і якісних характеристик. У разі якщо комбінація такою не є, то при побудові запитання вона не враховується.
3. Формування набору тестових запитань з отриманих комбінацій вершин.

Таблиця 1.1 – Переваги і недоліки методу генерації запитань на основі дерев І/АБО

Переваги	Недоліки
- висока швидкість генерації запитань; - велика кількість варіантів тестових запитань одного типу.	- кожний тип запитань вимагає побудови окремого дерева; - формування структури дерева запитання, як і заповнення його вузлів виконується вручну; - складання алгоритмів для перевірки коректності запитань виконується вручну для кожного типу запитань [20].

1.4.2. Генерація запитань за шаблонами

Метод генерації запитань за шаблонами дозволяє автоматично генерувати запитання, використовуючи певний набір шаблонів [21]. Застосування такого шаблону до певної частини тексту дозволяє виконати її перебудову в запитання. Як правило, даний метод використовується для побудови запитань до текстів, що знаходяться в структурованому форматі, таким як словники, представлені у вигляді масиву пар: <Термін, Визначення>.

Таблиця 1.2 – Переваги і недоліки методу генерації запитань за шаблонами

Переваги	Недоліки
- висока швидкість генерації запитання; - складання запитань різних типів.	- обмеження кількості типів запитань через використання обмеженого числа шаблонів; - можливість генерації запитань тільки до текстових даних певного формату [22].

1.4.3. Генерація запитань шляхом перебудови речення

Суть методу генерації запитань шляхом перестроювання речення полягає в перестроюванні стверджувального речення в питальне [1]. Він включає в себе наступні етапи:

1. Вибір об'єкта запитання. Як правило, запитання задається до підмета, присудка, означення або обставини.
2. Підбір питального слова.
3. Перебудова речення в залежності від об'єкта запитання.

Таблиця 1.3 – Переваги і недоліки методу генерації запитань шляхом перебудови речення

Переваги	Недоліки
- проста реалізація.	- помилки у визначенні роду і відмінка питального слова; - обмежене число об'єктів запитання [23].

1.4.4. Генерація запитань на основі текстового корпусу

Метод генерації запитань на основі текстового корпусу дозволяє автоматично генерувати набір різних запитань відразу до всього текстового корпусу, тобто набору текстів, об'єднаних певною спільною тематикою [24].

Стандартна реалізація даного методу передбачає наступні три основні етапи (рисунок 1.5):

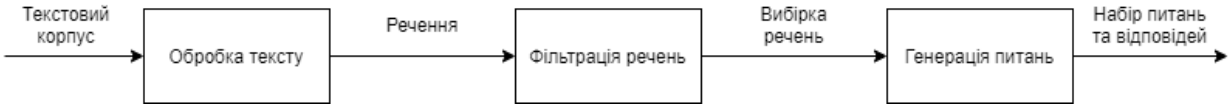

Рисунок 1.5 – Основні етапи методу генерації запитання на основі текстового корпусу

Таблиця 1.4 – Переваги і недоліки методу генерації запитань на основі текстового корпусу

Переваги	Недоліки
- генерує запитання до текстового корпусу; - помилки при побудові запитань практично відсутні.	- складнощі в реалізації; - невисока продуктивність.

1.5. Огляд існуючих систем

1.5.1 Система Quillionz

Quillionz створений Harbinger Group. Harbinger Group є світовим лідером у галузі електронного навчання та інженерії програмних продуктів. Harbinger постійно намагається кинути виклик статусу кво в електронному навчанні з інноваційними новаторськими продуктами, такими як конструктор інтерактивності Raptivity, інтерактивне відео Exaltive та Quillionz – унікальний автоматизований генератор запитань. Основна місія Harbinger – підтримувати та давати можливість фахівцям, які навчаються та навчають, створювати більше заохочення, залучення до навчання та удосконалювати досвід навчання [25].

Користувачу системи необхідно виконати наступні кроки:

1) надати текст для обробки (рис. 1.5);

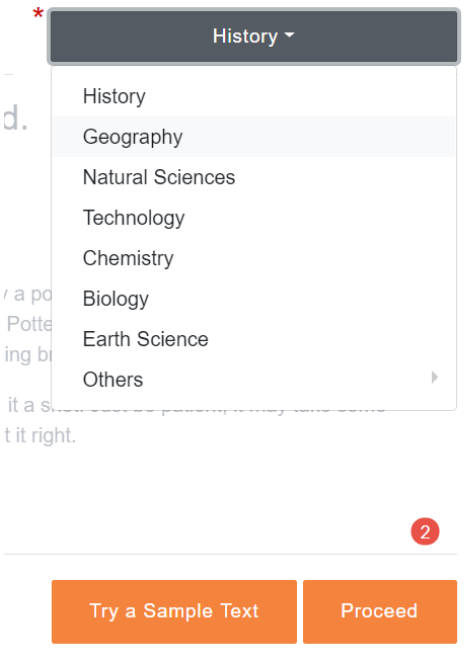

Рисунок 1.6 – Інтерфейс вибору предметної області тексту з можливістю використати шаблонний текст

2) обрати ключові слова, які допоможуть системі скласти кращі запитання і вдосконалити результат (рис. 1.6);

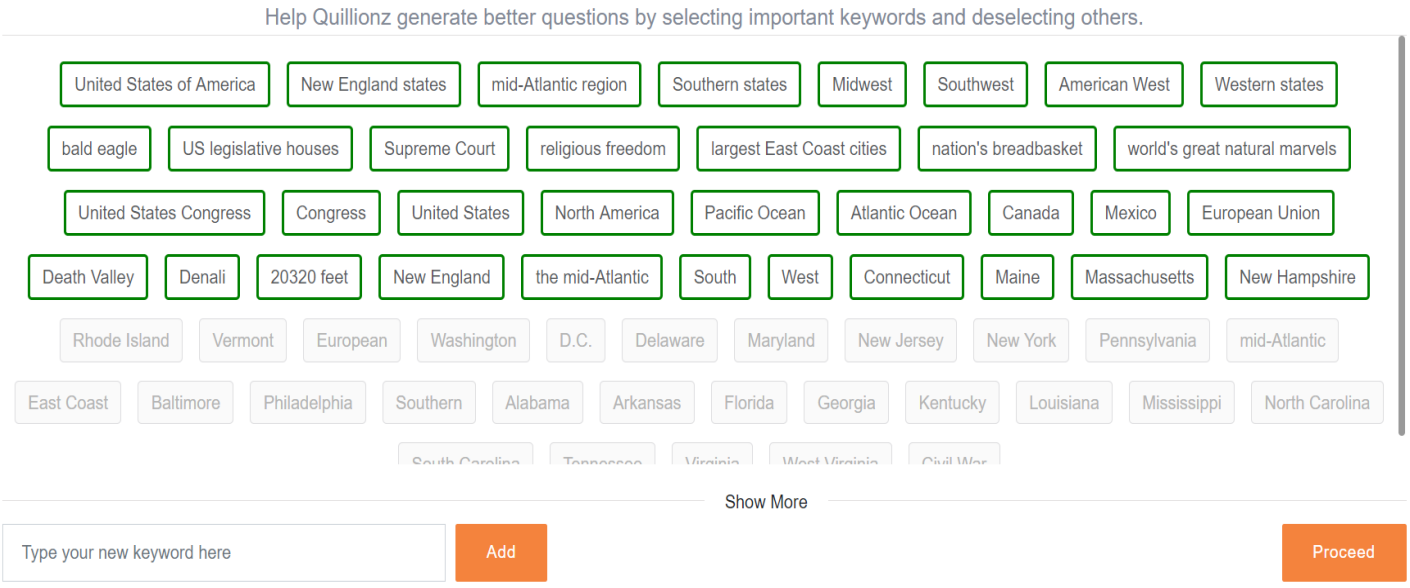

Рисунок 1.7 – Інтерфейс додавання ключових слів

3) перевірити контент і отримати запропоновані запитання (рис. 1.7).

Рисунок 1.8 – Можливість переглянути згенеровані запитання або пропустити цей крок

У результаті отримуємо запитання різних типів, як видно на рисунку 1.8.

Рисунок 1.9 – Згенеровані запитання, згруповані по типах запитань

Таблиця 1.5 – Переваги та недоліки системи генерації запитань Quillionz

Переваги	Недоліки
- підтримка різних типів запитань; - наявність безкоштовної версії; - можливість корекції запитань; - можливість покращення результату завдяки функції додавання ключових слів; - граматична коректність створених запитань; - гарний і зрозумілий інтерфейс.	- обробка тексту лише англійської мови; - обмеження в кількості слів – максимум 3000; - обмеження в предметних областях; - обмеження безкоштовної версії.

1.5.2 Нейромережева модель ParaQG

ParaQG – це нейромережева модель, яка генерує запитання на основі змісту тексту. У ParaQG є демо-версія інтерактивного сервісу для генерації запитань з абзацу тексту. ParaQG використовує трюки для фільтрації запитань, на які неможливо дати відповідь. Фільтруються запитання за допомогою BERT архітектури. Модель надає згенеровані запитання в згрупованому форматі. В одній групі знаходяться запитання зі схожими відповідями.

ParaQG генерує запитання з пропозицій і абзаців за допомогою 4-х кроків (рис. 1.9):

- Аналіз абзацу;
- Вибір відповіді;
- Генерація запитання з оцінкою впевненості в запитанні;
- Фільтрація запитань на підставі угруповання за схожими відповідей.

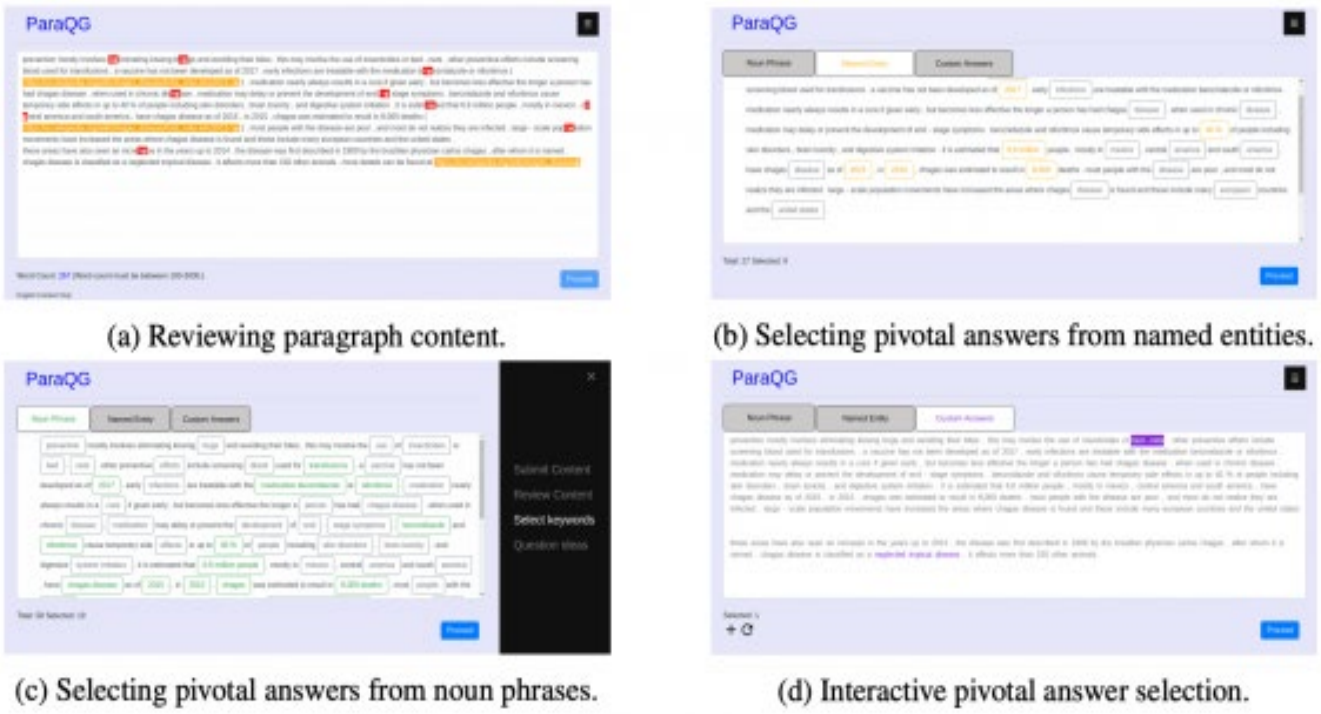

Рисунок 1.10 – Ключові кроки системи для генерації запитань [26]

Таблиця 1.6 – Переваги та недоліки системи ParaQG

Переваги	Недоліки
- Користувач має можливість вказати область тексту для генерації запитань; - Наявність фільтрації запитань; - Веб-додаток, що налаштовується користувачем	- Підтримка однієї мови

1.6. Постановка задачі

Постановка задачі:

- дослідити методи генерації запитань;
- реалізувати систему автоматичної генерації запитань;
    - реалізувати модифікований метод генерації запитань на основі текстового корпусу в поєднанні з методом генерації запитань шляхом перебудови;
    - реалізувати модифікований метод генерації запитань за шаблонами з використанням NER-моделі;
- виконати порівняльний аналіз та визначити ефективність реалізованих методів.

1.7 Висновки до розділу 1

У цьому розділі були зазначені основні задачі, які розглядає комп'ютерна лінгвістика, і було визначено перспективний напрям розвитку, а саме QA-системи, які мали би великий попит серед викладачів в сучасних реаліях пандемії та дистанційного формату навчання. Особливо актуальним це є для української мови, оскільки усі дослідження цієї галузі загалом зосереджені на реалізації систем для англійської мови у той час. Були зазначені основні методи генерації запитань природною мовою: на основі дерев І/АБО, за шаблонами, шляхом перебудови речення, на основі текстового корпусу. Для кожного методу були детально описані його принципи роботи, а також істотні

переваги та недоліки. Також були наведені приклади існуючих програмних застосунків і зазначені їх переваги та недоліки. У ході цього дослідження вдалося виявити, що наразі на ринку прикладних аналітичних програмних продуктів немає повноцінного інструменту для автоматичної генерації запитань до україномовних текстів. З огляду на цю проблему було вирішено дослідити та покращити існуючі методи генерації запитань, і на їх основі побудувати систему автоматичної генерації запитань до україномовних текстів.

РОЗДІЛ 2
ОГЛЯД ОБРАНИХ МЕТОДІВ ТА ПІДХОДІВ ЇХ РЕАЛІЗАЦІЇ ДЛЯ АВТОМАТИЧНОЇ ГЕНЕРАЦІЇ ЗАПИТАНЬ

2.1 Обрані методи генерації запитань для програмної реалізації

2.1.1 Модифікований метод генерації запитань на основі текстового корпусу

Першим методом для реалізації генерації запитань оберемо метод генерації запитань на основі текстового корпусу, адже він гарантує майже цілкову відсутність помилок при побудові запитань. А також він включає в себе реалізацію методу шляхом перебудови речення, адже для генерації остаточного запитання буде необхідно перебудовувати початкове речення, до якого генерується запитання.

Стандартна реалізація даного методу передбачає наступні три основні етапи:

1. Попередня обробка тексту. На даному етапі виконуються підготовчі дії по обробці тексту, необхідні для проведення наступних етапів, а саме:
    1.1. Розбиття тексту на речення. Основна складність на даному етапі полягає у визначенні меж речення. Аналізований текст може містити різні абревіатури і скорочення, що істотно ускладнює пошук меж речення. Для вирішення даної проблеми використовуються різні регулярні вирази і правила, згідно з якими шаблони застосовуються до тексту.

1.2. Побудова дерева речення. Полягає в перебудові речення зі своєї лінійної форми в деревоподібну [24]. Отримане дерево розмічується стандартним набором тегів [24], які в подальшому використовуються в процесі формування запитання.

1.3. Дроблення складних речень. Дерево речення дозволяє з легкістю визначити, чи описує воно просте речення або складне. Суть даного етапу полягає в розбитті складних речень на прості, що дозволяє спростити подальшу обробку тексту, поділивши його на атомарні структури.

1.4. Перерахунок тегів. Даний процес необхідний, оскільки при перебудові дерев значення певних тегів могло змінитися.

2. Фільтрація речень. Процес ґрунтується на ідеї систем автореферування, які видаляють з тексту "неважливі" в певному контексті речення. "Значимість" речення залежить від кількості набраних ним очок. На основі їх кількості видаляються ті речення, число очок яких не увійшло в заданий діапазон. При цьому необхідно, щоб кожне речення було оцінено рівно один раз. Очки речення розраховуються як середнє серед очок всіх вхідних в нього слів, очки яких обумовлюються їх частотою входжень в текстовий корпус. Далі всі речення сортуються відповідно до кількості набраних очок і видаляються речення з найбільшими і найменшими балами.

3. Генерація запитання. На даному етапі вирішуються наступні завдання:

3.1. Підготовка речення. Щоб полегшити процес перебудови речення, на даному етапі опускаються або замінюються деякі частини речення, що представляють неважливі деталі. До них відносяться: заміна скорочень на відповідні довгі форми; видалення знаків

пунктуації; написання першого слова в реченні з маленької літери, якщо воно не є власним ім'ям.

3.2. Визначення об'єкта запитання.

3.3. Розмітка дерева. Позначаються всі частини дерева, які в процесі побудови запитання повинні залишатися незмінними. Це допомагає уникнути вживання в запитанні зайвої інформації.

3.4. Виділення можливих відповідей на поставлене запитання. Полягає в знаходженні частини речення, яка може бути відповіддю на поставлене запитання. Варіантів відповіді може бути кілька. Далі вони розмічаються аналогічно розмітці дерев. Кожному отриманому дереву для даного речення відповідає унікальна відповідь. В даному процесі можна задати біля шести типів запитань.

3.5. Перебудова слів у реченні.

3.6. Генерація відповіді. Вона відома, оскільки на основі неї ми генерували запитання.

3.7. Додавання в початок і кінець запитання питальних слів.

3.8. Видалення відповіді з дерева.

Для реалізації модифікації даного методу використаємо допоміжні теги, які генерує морфологічний аналізатор pymorphy2, який позбавить нас від необхідності створювати дерево речення. Використанням даної бібліотеки буде забезпечено визначення характеристик слова на основі того, як це слово пишеться. Даний аналізатор впорається з задачею частиномовної розмітки, а також буде повертати перелік тегів до кожної леми. Перелік тегів, які можуть бути використані для реалізації модифікованого методу генерації запитань, наведені в таблиці 2.1.

Таблиця 2.1 – Перелік тегів морфологічного аналізатору pymorphy2

Назва тегу	Значення
POS (Part of Speech)	частина мови
animacy	живий/неживий предмет
aspect	вид дієслова
case	відмінок
gender	рід
mood	способ дієслова (дійсний, умовний, наказовий)
number	число
person	Особа (1, 2, 3)
tense	час

Теги та грамеми в pymorphy2 записуються латиницею, наприклад, NOUN. Але бібліотека також надає можливість використовувати кириличні назви грамем. Для перетворення довільних рядків з тегами/грамеми між кирилицею та латиницею існують спеціальні методи.

Pymorphy2 вміє схиляти слова. Щоб провідмінювати слово, потрібно спочатку зрозуміти, в якій формі воно є зараз і яка в нього лексема. Іншими словами, потрібно спершу розібрати слово та вибрати із запропонованих варіантів розбору правильний.

Також аналізатор спроможний приводити слова до початкової форми, узгоджувати з чисельниками, обирати правильний вибір розбору слова за показником score – оцінка P (tag|word), оцінка ймовірності того, що даний

розбір правильний. На практиці це означає, що перший розбір з тих, що повертають методи MorphAnalyzer.parse() і MorphAnalyzer.tag(), більш ймовірні, ніж інші. Але, на жаль, даний показник поки що не доступний для українського словника.

Було вирішено виключити з алгоритму модифікованого методу другий етап стандартної реалізації методу, заснованого на текстовому корпусі, а саме – фільтрацію речень. Оскільки в модифікації даного методу була виключена побудова дерева, постає необхідність в новому підході для реалізації заключного етапу – генерації запитань. Для вирішення даної проблеми використаємо ідею методу генерації запитань шляхом перебудови речення, адже маючи інформацію, надану морфологічним аналізатором pymorphy2, не постане труднощів виконати всі необхідні етапи даного методу.

2.1.2 Модифікований метод генерації запитань за шаблонами

Другим методом оберемо метод генерації за шаблонами, який демонструє високу швидкість генерації запитання, а також надає можливість складати запитання різних типів.

Шаблони запитань дають можливість задавати запитання, які не настільки тісно пов’язані з точним формулюванням вихідного тексту, як методи, засновані на синтаксисі та семантиці. Шаблон запитання — це будь-який заздалегідь визначений текст із змінними-заповнювачами, який слід замінити вмістом із вихідного тексту. Шаблони запитань дозволяють системам генерації запитань використовувати людський досвід у створенні мови.

Очевидною силою методів на основі шаблонів є їх здатність генерувати запитання з довільними формулюваннями. Отже, вони в принципі не

обмежуються генеруванням лише певних типів запитань, таких як хто, що, коли, де тощо.

Було вирішено, що при реалізації модифікованого методу за шаблонами, будемо будувати шаблони на основі NER-сутностей, визначених в реченні. Після чого до даних сутностей буде підібрано шаблон запитання та згенеровано його.

NER – це підзадача вилучення інформації, яка спрямована на пошук та класифікацію іменованих сутностей, згаданих у неструктурованому тексті, за заздалегідь визначеними категоріями, такими як імена осіб, організації, місцезнаходження, медичні коди, часові вирази, кількість, грошові значення, відсотки тощо.

Будемо використовувати наступні шаблони для ідентифікації речень, до яких можна згенерувати запитання, показані в таблиці 2.1.

Таблиця 2.2 – Відповідність шаблонів запитання та NER-сутностей

NER-сутність	Шаблон речення для запитання	Запитання
[PERS]	[PERS] зробив щось.	Хто зробив щось?
[LOC]	Хтось зробив щось у [LOC].	Де хтось зробив щось?

Для реалізації розпізнавання сутностей налаштуємо модель BERT на основі даних проекту lang-uk [27]. Корпус NER-анотацій від lang-uk включає 229 текстів з українського корпусу Брауна з 217 381 лексемами та 6751 коментованими іменованими сутностями.

BERT – це алгоритм глибокого навчання, що базується на технологіях обробки природної мови на основі нейронної мережі. Дана методика машинного навчання, яка ґрунтується на Трансформері, розроблена Google.

BERT – це загальноприйнята мовна модель, яка навчається на великому наборі даних. Цю попередньо навчену модель можна налаштувати та використовувати для різних завдань, таких як QA, сентиментальний аналіз, класифікація речень тощо. BERT – це найсучасніший метод трансферного навчання (англ. TL – Transfer Learning) в NLP.

Трансферне навчання – це метод, що застосовується в області машинного навчання (ML), метою якого є збереження знань, отриманих під час вирішення певної проблеми, та їх застосуванні до іншої, але пов'язаної з даною, задачі [28].

При використанні моделей глибинного навчання зазвичай використовують саме цей підхід, особливо популярним він виявляється серед рішень, де попередньо натреновані моделі використовуються як відправна точка для задач в області NLP.

Враховуючи величезні часові та обчислювальні ресурси, необхідні для розробки моделей нейронних мереж для подібних задач, такий підхід дозволяє досягти високих результатів при мінімальних затратах для вирішення схожих проблем.

Загалом, підхід включає наступні кроки:

1. Вибір початкової моделі. Відбувається вибір доступної попередньо натренованої вихідної моделі. Нині багато науково-дослідних організацій створюють моделі на великих і складних наборах даних у різноманітних доменах, які можуть бути використані для задач різного плану.
2. Повторне використання моделі. Заздалегідь підготовлена модель може бути використана в якості вихідної точки для іншого завдання. Є

можливість як використати початкову модель повністю, так і лише її частину, залежно від обраної методики моделювання.

3. Налаштування моделі. За бажанням модель можна адаптувати або удосконалити на даних, доступних для поставленої задачі.

Класичним прикладом трансферного навчання у комбінації з архітектурою трансформерів є моделі BERT – двонаправлені представлення енкодерів у моделі трансформерів [29]. Як і рекурентні нейронні мережі (RNN), Трансформери призначені для обробки послідовних вхідних даних, таких як природна мова. Однак, на відміну від RNN, Трансформери не вимагають обробки послідовних даних за порядком. Наприклад, якщо дані входу є реченням природної мови, то Трансформерові не потрібно обробляти його початок, перед тим, як взятися за обробку його кінця. Через цю його особливість Трансформер уможливлює набагато більше розпаралелювання, ніж RNN, і відтак знижує тривалості тренування [30].

2.2 Підхід NLP QA-систем

Загалом існує три підходи до реалізації QA-систем. Розглянемо детальніше NLP QA-системи, адже будемо застосовувати саме такий підхід.

QA-системи, що базуються на NLP, забезпечують найнадійніший діалог та відповіді між людиною та комп’ютером, що досягається за допомогою подання тексту як семантичного представлення. Основним напрямком діяльності NLP були природномовні інтерфейси для баз даних та експертних систем у вузькій області. D. Molla та співавтори [31] показують, що вузькі та технічні домени вимагають ретельного та глибокого NLP, на відміну від

інтенсивного забезпечення якості IR (Information Retrieval). База знань системи NLP створює труднощі з її перенесенням. На рисунку 2.3 показано шість компонентів (лінгвістичний інтерфейс), які змінюються при зміні мови введення, і три компоненти (знання домену), які змінюються при зміні домену знань.

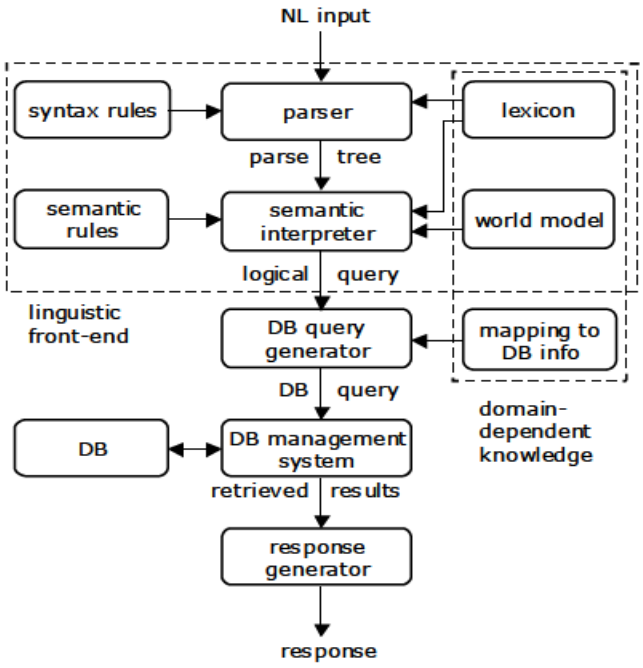

Рисунок 2.3 – Архітектура типової QA-системи [31]

Зміни вимагають залучення кваліфікованого персоналу: програмістів, інженерів знань та адміністраторів баз даних [31]. W. Winiwater [32] підвищує портативність за допомогою двох різних лексиконів: залежного від домену та незалежного. W. Chu та F. Meng [33] покращують портативність за допомогою семантичного графіку, який представляє структуру бази даних, ключові слова, вручну присвоєні вузлам та атрибутам на графіку. По суті, це рух до шаблонного підходу.

У таблиці 1 узагальнено три підходи до побудови QA-систем з точки зору застосування. У великому домені, такому як Інтернет, IR QA-системи є

найбільш доцільними. Поглиблений NLP технічно можливий, але витрати на формування офіційного семантичного представлення інформації у всій Мережі досить високі. Ask Jeeves намагався охопити Інтернет великою кількістю шаблонів запитань. Розриви в охопленні були значними, тому відповіді на запитання підтримувалися пошуком ключових слів. Шаблони запитань в Інтернеті є найбільш корисними, якщо вони пов’язані з онлайновими базами даних (федеративний підхід, [34]).

Малим доменам підходять QA-системи на основі NLP та шаблонів. Обидві методики також можуть утворювати інтерфейс структурованої бази даних. Вилучення фактів з тексту здійснюється за допомогою NLP (малі домени) та IR QA (великі домени), але не за допомогою QA за шаблонами, оскільки цей підхід не обробляє текст.

Таблиця 2.3 – Порівняння підходів до QA-систем [11]

	NLP	IR & NLP	Шаблонний
Весь Інтернет	Ні	Так	Так
Структуровані дані	Так	Ні	Так
Факти з тексту	Так	Так	Ні
Поради	Так	Ні	Так
Надійність, %	≈100	>70	>80
Малі домени	Так	Ні	Так

Поради означають відповідати на запитання без зосередження уваги на фактах. Відповіді на поширені запитання зазвичай передбачають поради. NLP та шаблонні системи підходять для цього, тоді як експерименти в рамках IR QA не дали високої якості [35].

Отже, немає найкращого чи найгіршого підходу для побудови QA-систем. Кожний підхід має свою нішу, середовище застосування та завдання. Якщо якість відповідей має вирішальне значення, слід застосовувати NLP. Якщо з тексту потрібно витягти факти, слід використовувати IR QA. Обмежені домени, покрокове зростання баз знань (наприклад, Ask Jeeves) або відсутність висококваліфікованої робочої сили можуть скористатися перевагами шаблонного підходу. Майбутня тенденція досліджень повинна передбачати поєднання підходів в єдиній системі та адаптувати методи до сфери застосування.

Оскільки NLP-підхід є більш надійним та надає найякісніші результати, буде доцільним звернутися саме до цього підходу при реалізації методу генерації запитань на основі текстового корпусу та методу генерації запитань за шаблонами.

Для реалізації модифікованого методу на основі текстового корпусу було вирішено використовувати підхід на основі правил.

Даний підхід традиційно передбачає, що людина бере участь у процесі поетапного розвитку та вдосконалення системи. Найбільша перевага формальної граматики полягає в тому, що завжди існує спосіб перевірити, чи може система обробити запит, розміщений користувачем, і як це зробити. А оскільки всі правила написані людьми, будь-яку повідомлену помилку легко локалізувати та виправити, скоригувавши правила у відповідному модулі.

Такий підхід заснований на розробці та розширенні існуючих правил, тому система не потребує масивного навчального корпусу, порівняно з підходом до машинного навчання.

Найбільш очевидним недоліком підходу, заснованого на правилах, є те, що він вимагає кваліфікованих експертів: лінгвісту або інженеру знань потрібно вручну закодувати кожне правило в NLP. Правила потрібно постійно розробляти і вдосконалювати також власноруч. Більше того, система може стати настільки складною, що деякі правила можуть почати суперечити один одному.

У цілому система, заснована на правилах, добре сприймає певне мовне явище: воно розшифрує мовні зв’язки між словами для інтерпретації речення. Тому вона може дуже добре впоратися із завданнями на рівні речень, такими як розбір та вилучення [36].

Таблиця 2.4 – Переваги та недоліки підходу, заснованого на правилах

Переваги	Недоліки
- гнучкий; - легко налагоджувати; - не потребує масивного навчального корпусу; - висока точність.	- потребує кваліфікованих експертів; - повільний аналіз парсера; - середній відклик.

Для реалізації модифікованого методу за шаблонами звернемося до глибинного навчання, адже для його реалізації було вирішено реалізувати NER-модель.

Тривалий час більшість методів, що використовуються для вивчення проблем NLP, використовували неглибокі моделі машинного навчання та трудомісткі функції, виготовлені вручну. Це призводить до таких проблем, як прокляття розмірності, оскільки мовна інформація була представлена високовимірними ознаками. Однак, з недавньою популярністю та успіхом вбудовування слів, моделі, які використовують нейронні мережі, досягли найкращих результатів у різних завданнях, пов’язаних з мовою, порівняно з традиційними моделями машинного навчання, такими як SVM або логістична регресія.

Даний підхід застосовує векторне представлення слів, нейронні мережі прямого поширення, рекурентні і згорткові мережі тощо. Глибинні нейронні архітектури досягли передових результатів у багатьох задачах обробки природної мови, таких як розбір складників, аналіз тональності, отримання інформації, розуміння усного мовлення, машинний переклад, контекстне зв'язування об'єктів та інших.

Таблиця 2.5 – Переваги та недоліки глибинного навчання в задачах NLP

Переваги	Недоліки
- висока продуктивність; - мала або відсутня необхідність знань системи про галузь;	- відсутність теоретичного підґрунтя; - відсутність інтерпретаційної моделі;

Продовження таблиці 2.5

Переваги	Недоліки
- переважно застосування навчання з вчителем; добре вирішує задачі розпізнавання образів	- потреба великої кількості даних та потужних обчислювальних ресурсів; - неможливість тренувальних даних охопити всі виключення; - не може безпосередньо обробляти символи; - модель, як правило, чорна скринька, яку важко зрозуміти; відсутність методів навчання без вчителя

2.5. Алгоритми обраних методів генерації запитань

Беручи до уваги необхідні етапи попередньої обробки тексту, алгоритм методу генерації запитань на основі текстового корпусу представлений на рисунку 2.4. Він включає до себе крок токенізації за реченнями та словами, лематизацію, частиномовну розмітку і генерацію запитання.

А алгоритм методу за шаблонами зображено на рисунку 2.5. До нього входять наступні етапи: токенізація, NER-розмітка, пошук шаблону, генерація запитання.

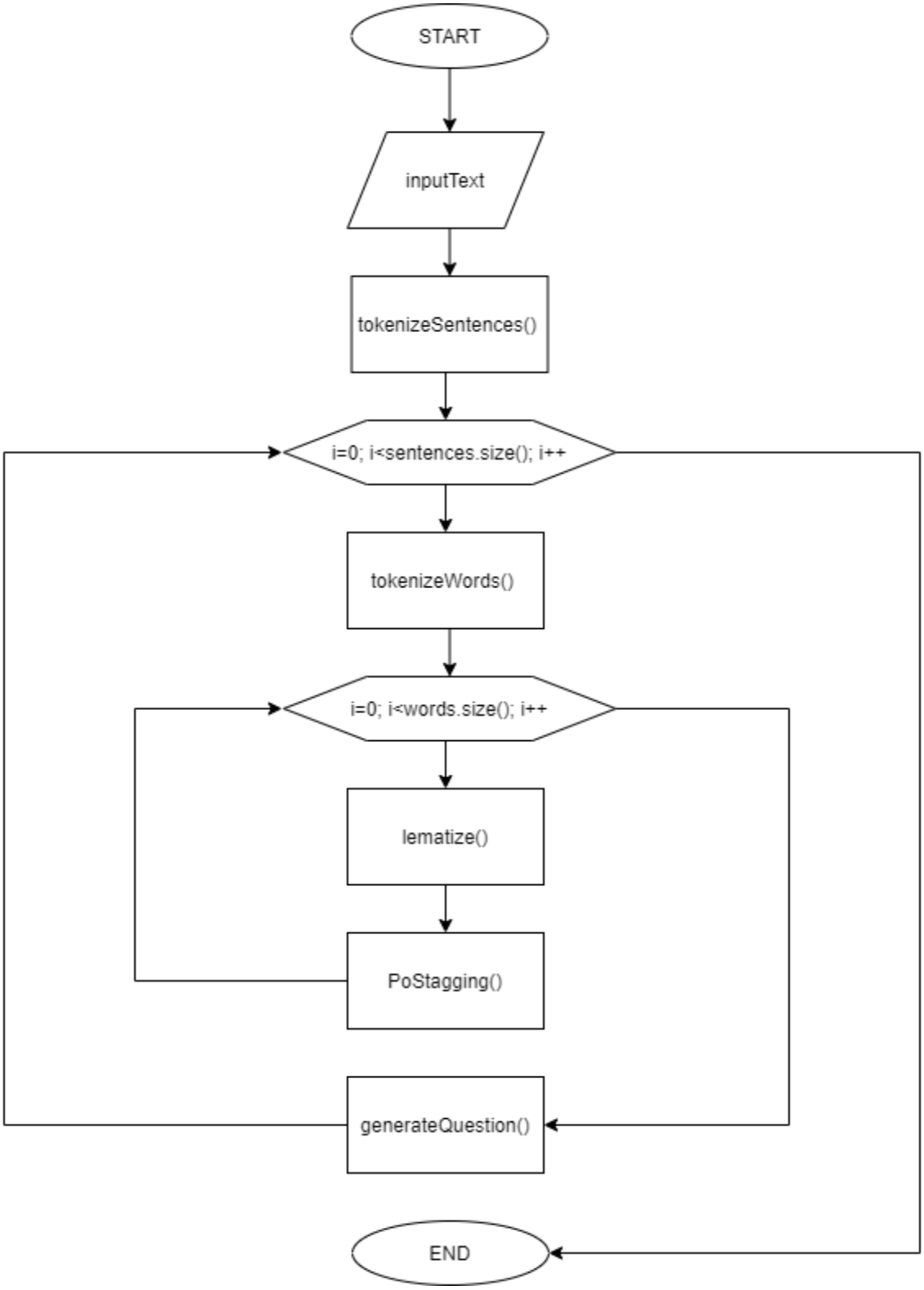

Рисунок 2.4 – Узагальнений алгоритм методу генерації запитань на основі текстового корпусу

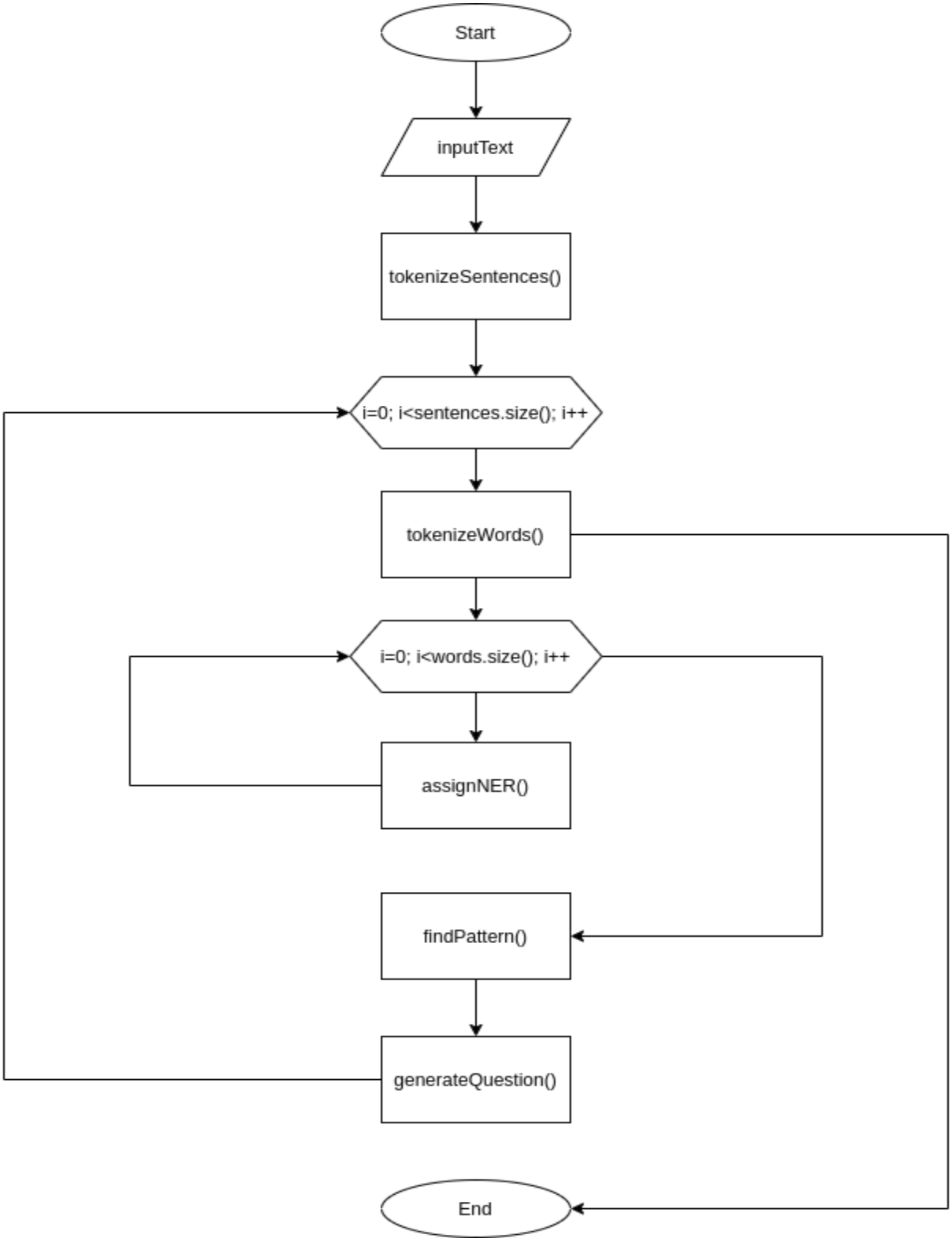

Рисунок 2.5 – Узагальнений алгоритм методу генерації запитань на основі текстового корпусу

На рисунку 2.6 наведена діаграма прецедентів системи:

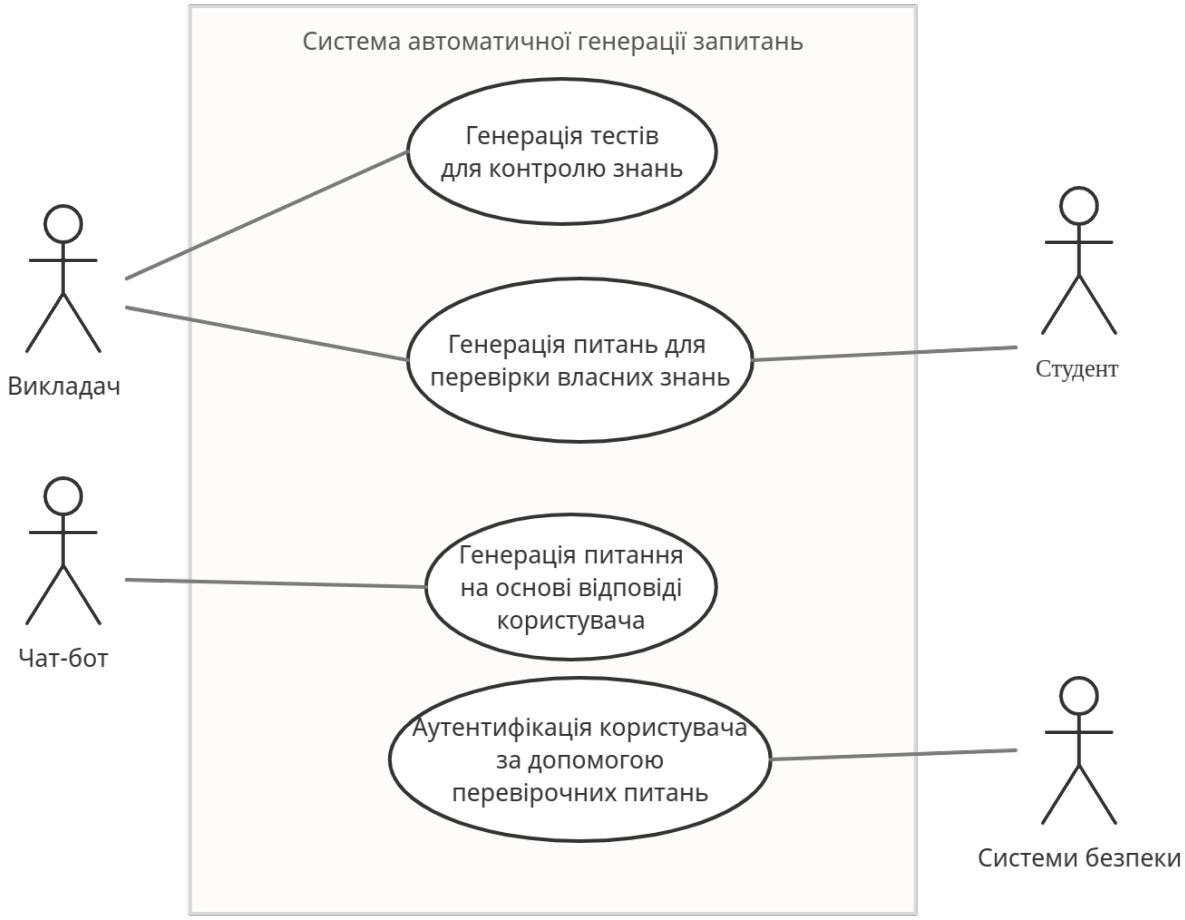

Рисунок 2.6 – Діаграма прецедентів системи

Діаграма діяльності продемонстрована на рисунку 2.7.

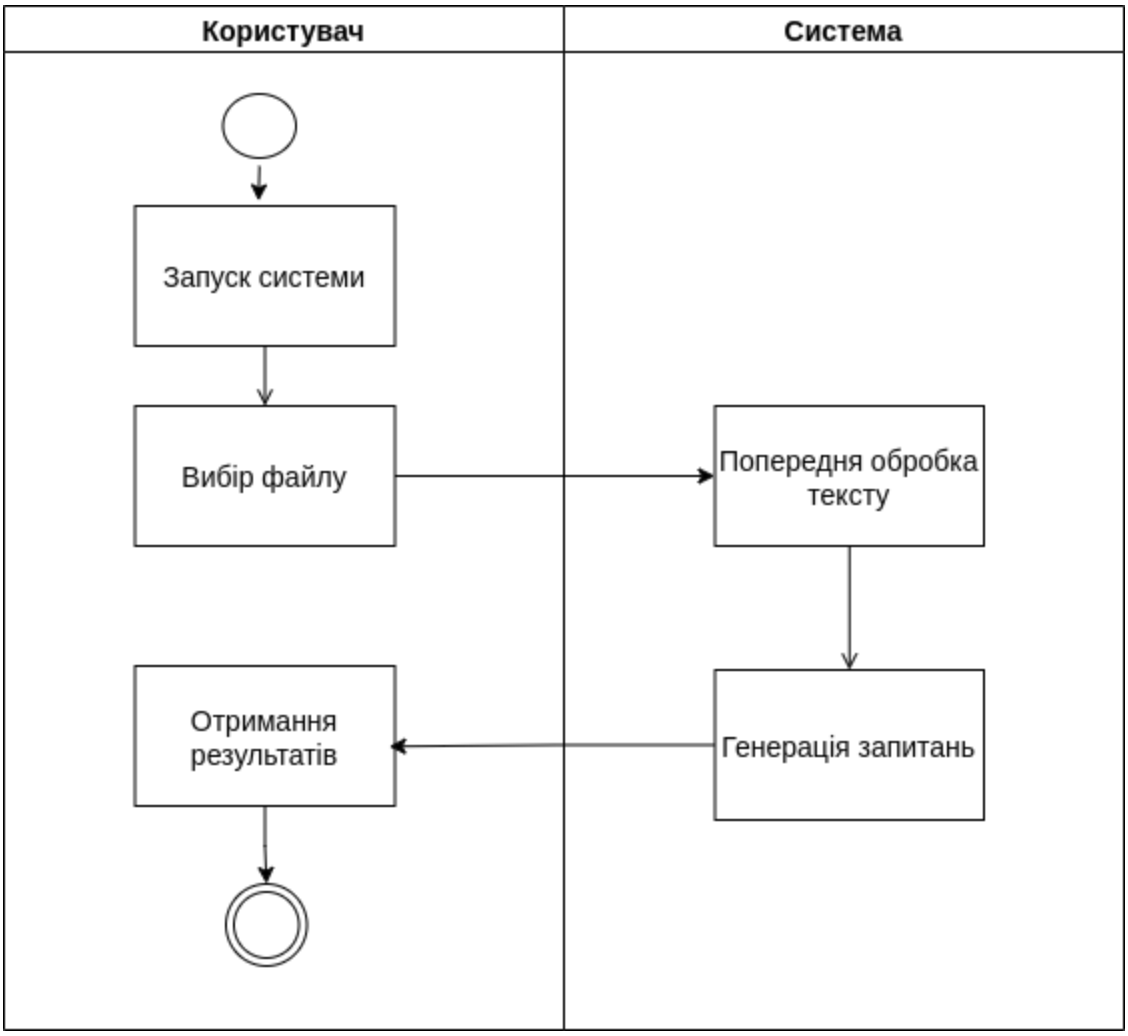

Рисунок 2.7 – Діаграма діяльності

2.6 Використані програмні засоби

Для реалізації обох методів генерації запитань було обрано таке програмне середовище, як PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладчик, інструмент для запуску юніт-тестів і підтримує веб-розробку на Django. PyCharm розроблена компанією JetBrains на основі IntelliJ IDEA.

Важливу роль відіграє саме вибір мови програмування, адже Python був наче створений для задач NLP. Для вирішення багатьох задач існують різні бібліотеки. Особливу увагу потрібно приділити бібліотеці NLTK (Natural Language Toolkit).

Дана безкоштовна бібліотека для мови програмування Python є однією з кращих для створення різних програмних продуктів на цій мові. Вона надає великий набір інструментів, корпус тексту [37]. Містить графічні уявлення і приклади даних, а також супроводжується великою документацією.

NLTK добре підходить для студентів, які вивчають комп'ютерну лінгвістику або близькі предмети, такі як емпірична лінгвістика, когнітивістики, штучний інтелект, інформаційний пошук і машинне навчання. NLTK з успіхом використовується як навчальний посібник, як інструмент індивідуального навчання і в якості платформи для прототипування і створення науково-дослідних систем.

NLTK в даному проекті допоможе розбити текст на речення, а згодом – на слова, а також надасть можливість додати власний список українських стоп-слів. А з задачею розмітки частин мов впорається морфологічний аналізатор pymorphy2 та виконає:

1) приведення слова до нормальної форми;
2) перетворення слова до початкової форми;
3) надання граматичної інформації про слово (число, рід, відмінок, частина мови тощо)

Завдяки ньому кожний токен отримає так звані теги, на які й будемо орієнтуватися при генеруванні запитань. Якщо запитувана характеристика для даного тега не визначена, то повертається None.

Для реалізації NER-моделі за допомогою алгоритму BERT стануть в нагоді бібліотеки bert та tensorflow, а саме його модуль keras. Хоча TensorFlow

– це інфраструктурний рівень для диференційованого програмування, роботи з тензорами, змінними та градієнтами, keras – це користувальницький інтерфейс для глибокого навчання, роботи з шарами, моделями, оптимізаторами, функціями втрат, метриками тощо. Для компіляції моделі keras був використаний оптимізатор Adam зі швидкістю навчання $1e-4$.

2.7 Висновки до розділу 2

У даному розділі були детально описані обрані методи генерації запитань та особливості їх модифікованої реалізації. Було розглянуто два підходи: на основі правил та глибинного навчання. На основі першого підходу реалізовано модифікований метод на основі текстового корпусу, а елементи другого, а саме навчання NER-моделі, використовуються в реалізації модифікованого методу за шаблонами. Також було наведено схеми алгоритмів роботи реалізованих методів, а також описано програмні засоби, які були використані.

РОЗДІЛ 3
ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

3.1. Розробка і реалізація інтерфейсу користувача

Графічний інтерфейс програми реалізований за допомогою бібліотеки PySimpleGUI і представлений на рисунку 3.1.

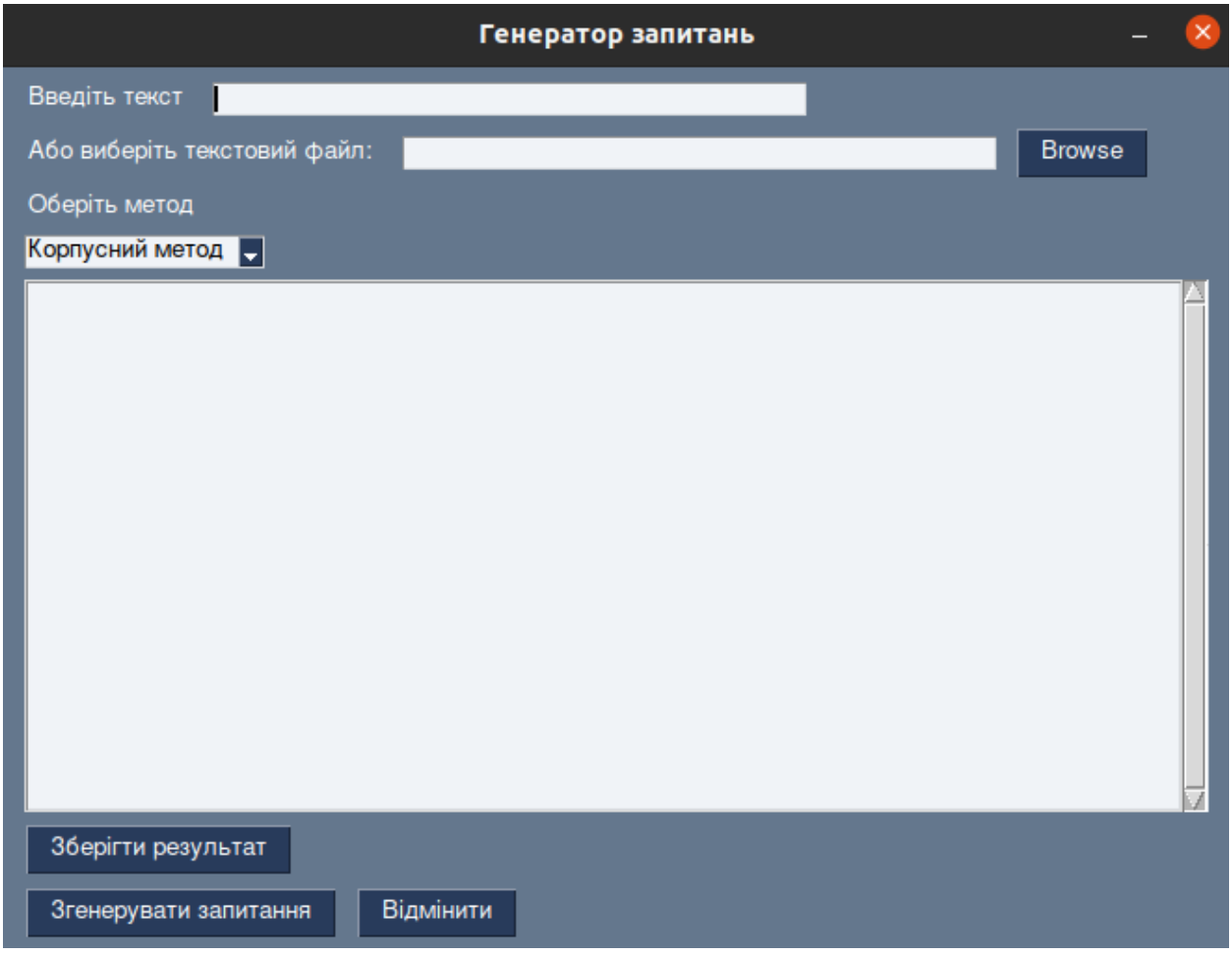

Рисунок 3.1 – Графічний інтерфейс програмного модулю

Опція вибору методу "Корпусний метод" позначає метод на основі текстового корпусу, а "Шаблонний метод" – метод за шаблонами.

3.2 Демонстрація результатів реалізації обраних модифікованих методів генерації запитань

Наведемо результати реалізації методу на основі текстового корпусу, а саме основних етапів програми на прикладі речення "Олегу раніше доводилося робити домашнє завдання.".

Таблиця 3.2 – Результат токенізації речення

Вхідні дані	Вихідні дані
Олегу раніше доводилося робити домашнє завдання.	['Олегу', 'раніше', 'доводилося', 'робити', 'домашнє', 'завдання', '.']

Далі продемонструємо результати роботи морфологічного аналізатору pymorphy2.

Таблиця 3.3 – Результат морфологічного аналізу слів речення

Вхідне слово	Варіанти розбору слова
"Олегу"	tag=OpencorporaTag('NOUN,masc,anim datv'), normal_form='олег'
"раніше"	tag=OpencorporaTag('ADJF,COMP neut,nomn'), normal_form='раніший'; tag=OpencorporaTag('ADVB COMP'), normal_form='раніше'
" доводилося"	tag=OpencorporaTag('VERB,Refl,impf neut,past'), normal_form='приходитися'

Продовження таблиці 3.3

Вхідне слово	Варіанти розбору слова
"робити"	tag=OpencorporaTag('VERB,impf infn'), normal_form='робити'
"домашнє"	tag=OpencorporaTag('ADJF neut,nomn'), normal_form='домашній'
"завдання"	tag=OpencorporaTag('NOUN,inan neut,nomn'), normal_form='завдання'

Як бачимо, через відсутність оцінки ймовірності правильності розбору для українського словника, перший розбір для прислівника "раніше" виявився неправильним. Але для іменників, дієслів та прикметників аналізатор добре показує себе. Отже, буде доречним скласти правила побудови речення з запитанням до перелічених частин мов.

Опишемо логіку побудови правил для кожної частини мови.

1) Іменник: як тільки зустрічається іменник, мі перевіряємо, живий предмет це чи ні. Якщо живий, то ми ставимо слово "хто" у відповідний відмінок, якщо неживий – відмінюємо слово "що". Перевіряємо, чи стоїть перед іменником якийсь прикметник, адже у разі його наявності нам потрібно прибрати його з речення. Далі прибираємо наш іменник з речення та ставимо "хто" або "що" на початок.
2) Дієслово: коли зустрічається дієслово, ми ставимо слово робити в такий же час, шукаємо об’єкт дії та будуємо запитання.
3) Прикметник: коли зустрічається прикметник, ми ставимо слово "який" в такий же рід, що й у прикметника; шукаємо іменник, пов’язаний з цим словом, та визначаємо відмінок, який також застосовуємо до слова "який". Щоб перетворити речення на запитання, прибираємо

прикметник, ставимо "який" на початок, а після нього вставляємо іменник.

Наведемо тестові приклади.

Згенеруємо запитання до речення "Олегу раніше доводилося робити домашнє завдання.". Отримуємо наступні результати, зображені на рисунку 3.2.

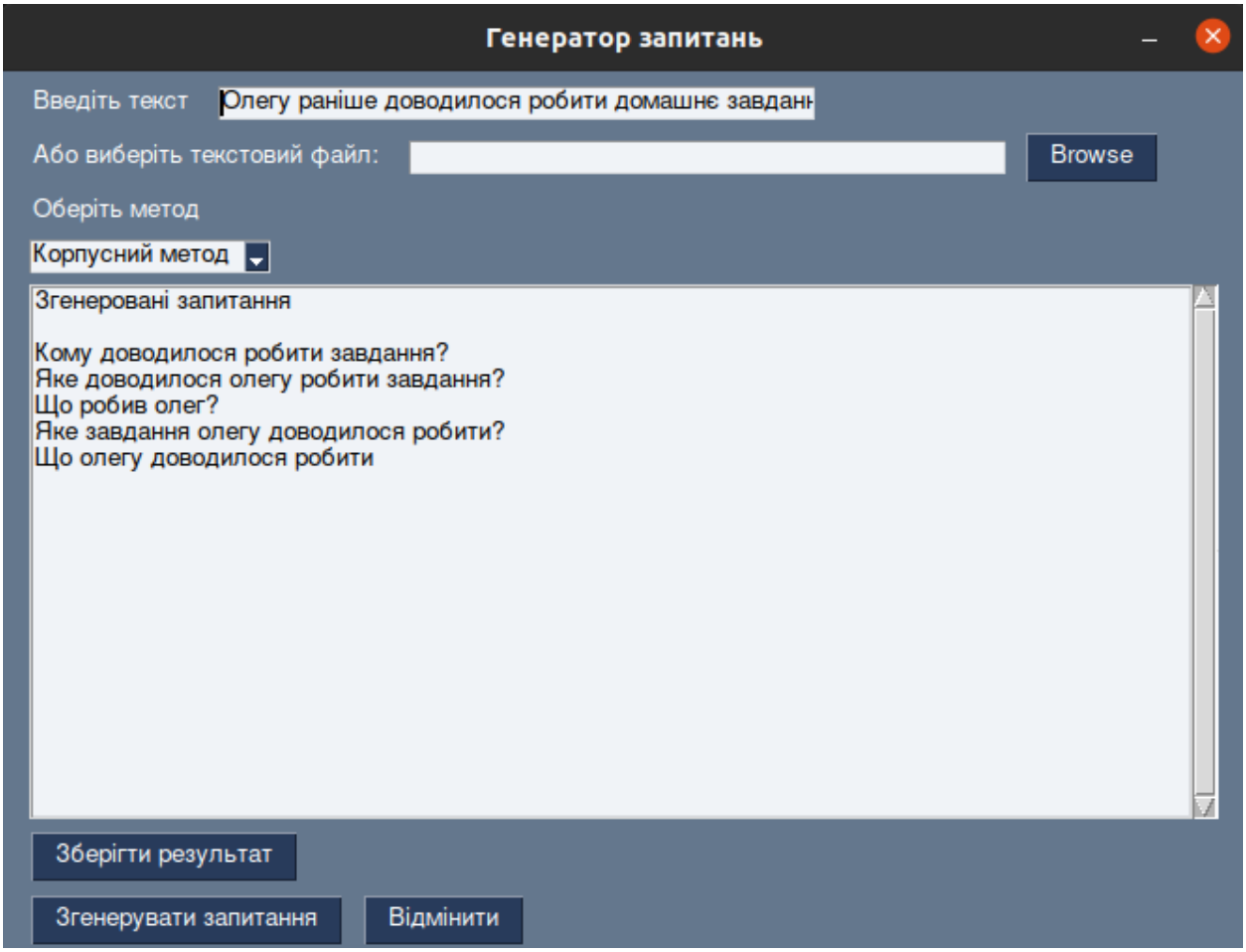

Рисунок 3.2 – Результат роботи методу на основі текстового корпусу

В результаті обробки речення було отримано 5 запитань, серед яких задовільними є чотири з п’яти. У другому реченні було неправильно підібрано питальне слово до слова "раніше". Також бачимо, що система ігнорує верхній регістр для імен.

Розглянемо наступний приклад з реченням "Батько замовив піцу на вечерю.". Результати роботи програми, відображені на рисунку 3.3

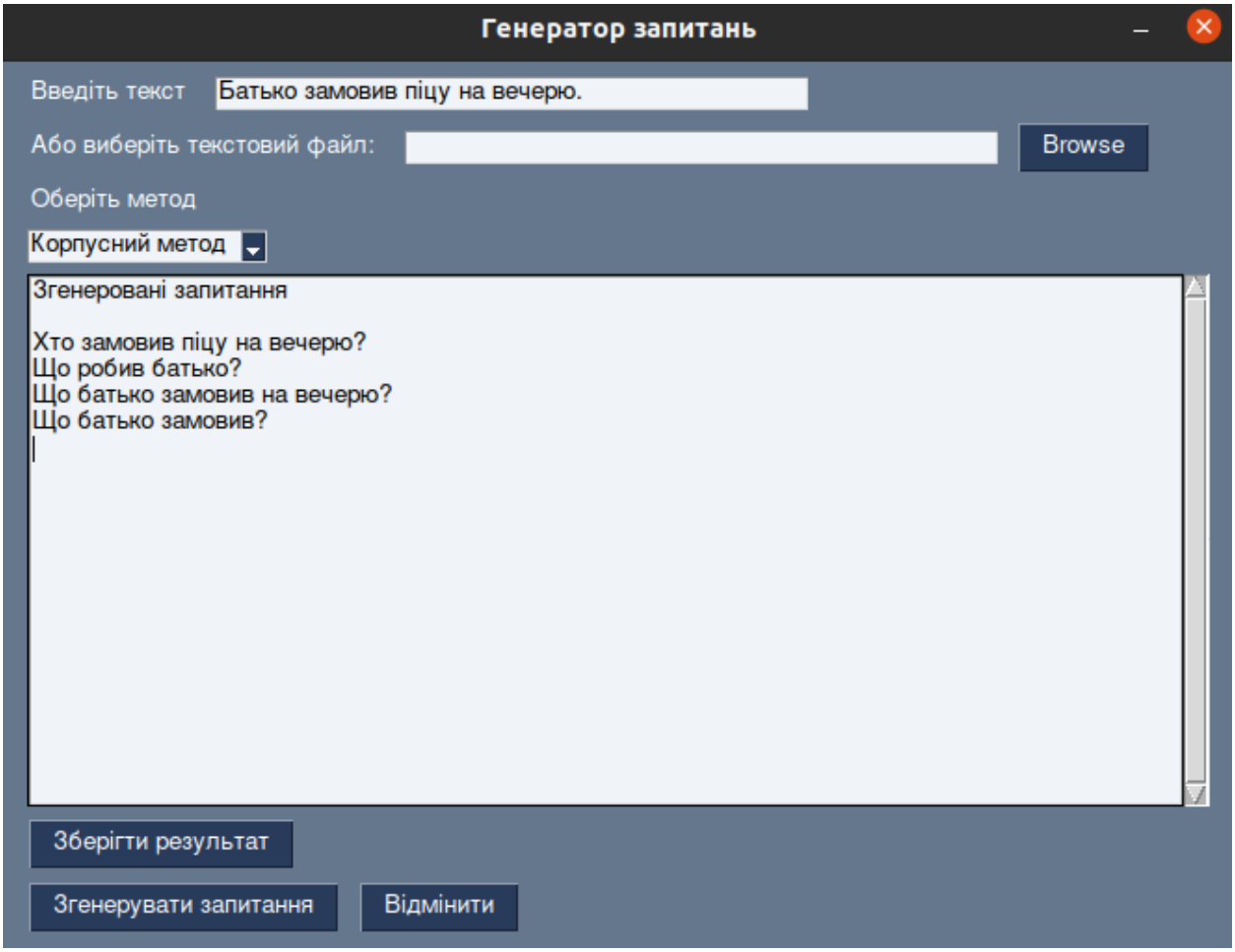

Рисунок 3.3 – Результат роботи методу на основі текстового корпусу

З даним реченням система впоралася чудово. Усі запитання мають сенс, а також коректні з точки зору граматики.

Згенеруємо запитання до вище згаданих речень за допомогою методу за шаблонами. Результати представлені на рисунках 3.4 та 3.5.

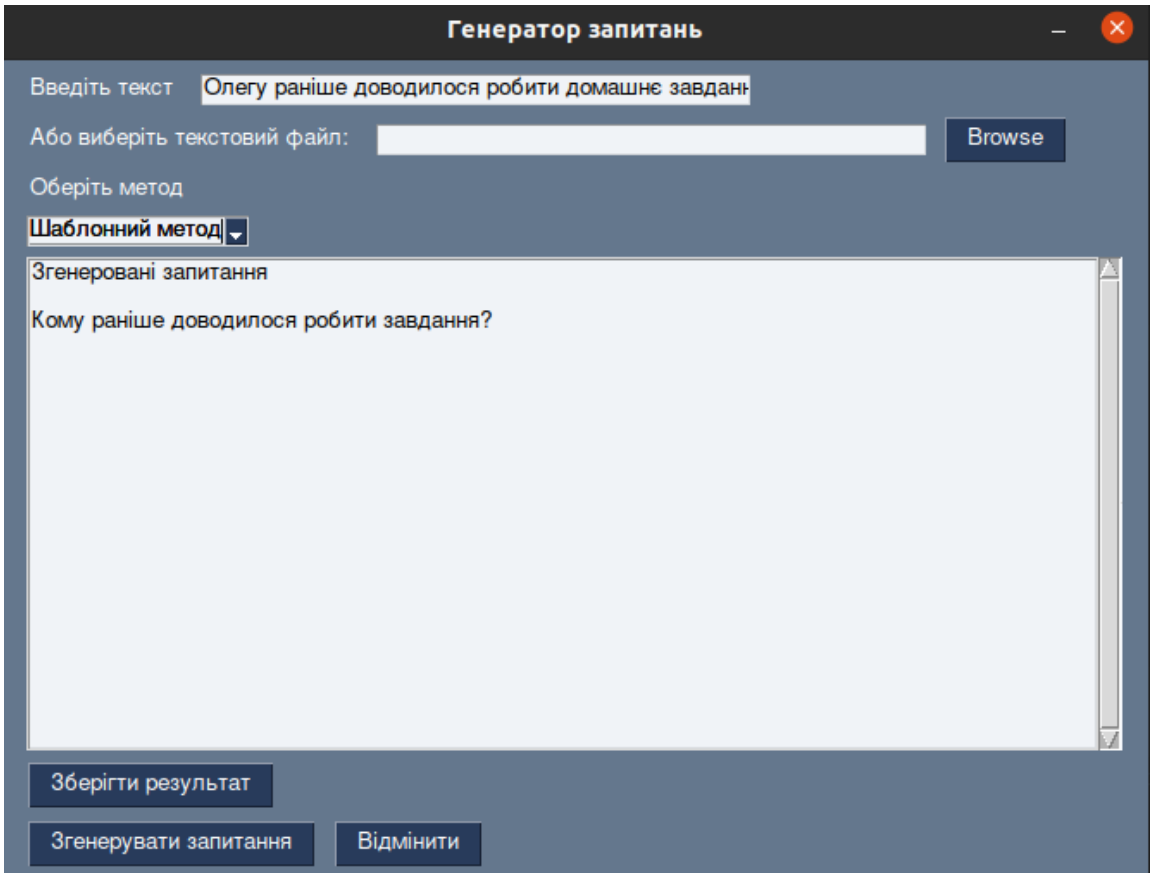

Рисунок 3.4 – Результат роботи методу за шаблонами

З рисунку 3.4 бачимо, що системою була розпізнана лише одна NER-сутність, й, відповідно, було згенероване лише одне запитання. Отже, хоча й результат отримано, але він дуже програє кількісно в порівнянні з методом на основі текстового корпусу, який видав п’ять запитань, чотири з яких були правильні.

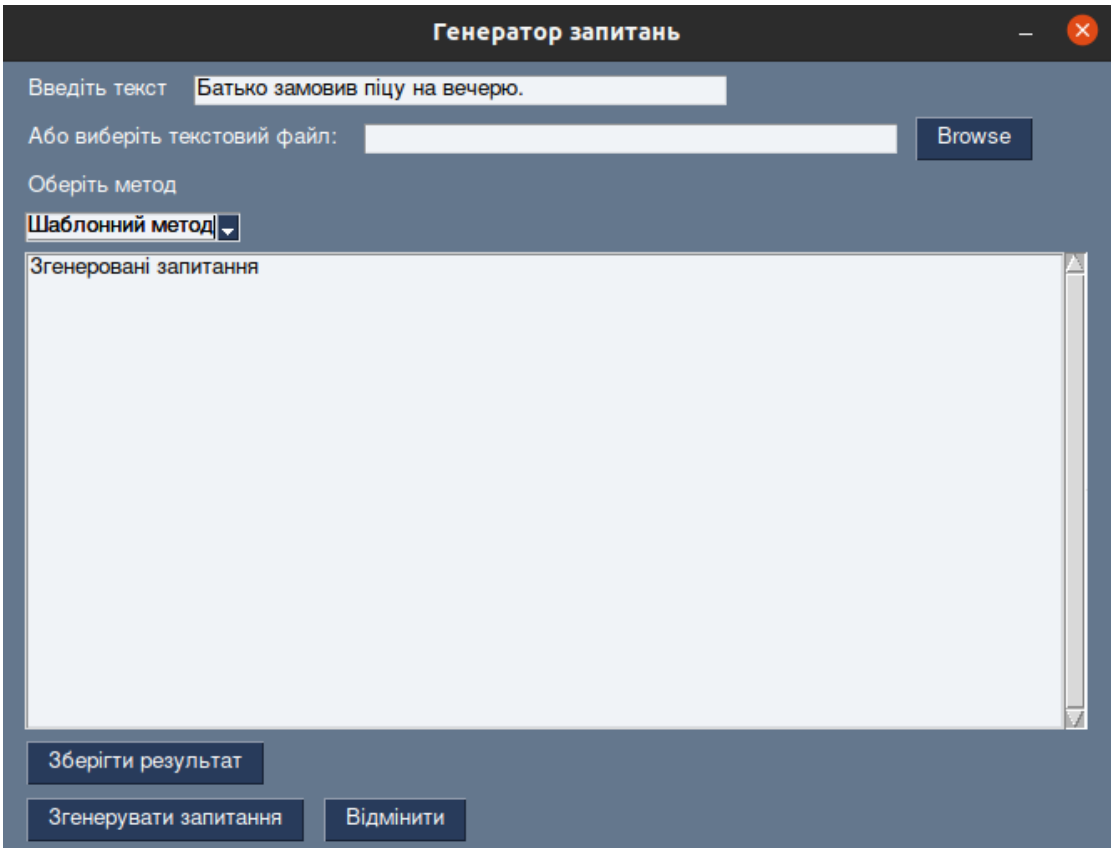

Рисунок 3.5 – Результат роботи методу за шаблонами

На жаль, метод за шаблонами не впорався з даним прикладом (рис. 3.5). Даний результат обумовлений відсутністю NER-сутностей, розпізнаних системою. Відповідно, без наявності сутностей не було віднайдено шаблон до генерації запитання. Наведемо ще по декілька прикладів для кожного методу в таблицях 3.4. та 3.5.

Таблиця 3.4 – Тестові приклади для методу на основі текстового корпусу

Вхідне речення	Отримані запитання
Олегу раніше доводилося робити домашнє завдання.	Кому доводилося робити завдання? Яке доводилося Олегу робити завдання?

Продовження таблиці 3.4

Вхідне речення	Отримані запитання
Олегу раніше доводилося робити домашнє завдання.	Що робив Олег? Яке завдання Олегу доводилося робити? Що Олегу доводилося робити?
Батько замовив піцу на вечерю.	Хто замовив піцу на вечерю? Що робив батько? Що Батько замовив на вечерю?
Подруга приготувала сніданок.	Хто приготував сніданок? Що робила подруга? Що подруга приготувала?
Марія полетіла до Парижу.	Хто полетів до Парижу? Що зробила Марія? Куди полетіла Марія?

Розглянемо результати підбору шаблону на основі NER-нотації та генерування запитання.

Таблиця 3.5 – Тестові приклади для методу за шаблонами

Вхідне речення	Підібраний шаблон	Отримані запитання
Олегу раніше доводилося робити домашнє завдання	{PERS} зробити щось.	Кому раніше доводилося робити завдання?
Батько замовив піцу на вечерю.	-	-

Продовження таблиці 3.5

Вхідне речення	Підібраний шаблон	Отримані запитання
Подруга приготувала сніданок.	-	-
Марія полетіла до Парижу.	{PERS} зробити щось в {LOC}	Хто полетів до Парижу? Куди полетіла Марія?

Як бачимо, даний метод привласнює сутність {PERS}лише до власних назв. Відповідно до речень із загальними словами “батько”, “подруга” не було підібрано шаблону. Але метод гарно впорався з останнім реченням, хоча метод на основі текстового корпусу впорався дещо краще, адже було згенеровано на одне запитання більше.

3.3. Порівняльний аналіз реалізованих методів

Одна з методологій оцінювання систем генерації запитань спирається на порівняльну оцінку результатів роботи системи (Q_g) та запитань, які створили люди (Q_h) з того самого корпусу. Ми можемо сприймати це як непряму оцінку людини. Ali та інші [38], Kalady та інші [39] використовують два підходи, які були оцінені за допомогою цієї методології. Для вхідного документа або корпусу можна обчислити точність і відклик, де створені людиною запитання вважаються золотим стандартом. Очевидно, що такий тип оцінки можливий лише в тому випадку, якщо генеровані людиною запитання вже доступні або є можливість, щоб люди склали запитання.

$$precision = \frac{Q_g \cap Q_h}{Q_g}, \quad (3.1)$$

$$recall = \frac{Q_g \cap Q_h}{Q_h}, \quad (3.2)$$

де Q_g – питання, згенеровані системою,

Q_h – питання, складені людьми.

Наведемо результати застосування формул 3.1 та 3.2 до показників роботи реалізованих методів. Оцінка була проведена на основі 50 речень.

Таблиця 3.6 – Порівняння точності та відклику реалізованих методів

	Точність	Відклик
Модифікований метод на основі текстового корпусу	0,72	0,64
Модифікований метод за шаблонами	0,54	0,21

З таблиці 3.6 видно, що модифікований метод на основі текстового корпусу має більші точність та відклик. Низький відклик модифікованого методу за шаблонами пояснюється тим, що не до всіх речень вдається підібрати хоча б одну NER-сутність, на основі якої буде згенеровано запитання. Отже, необхідно налагоджувати NER-модель та розширювати спектр сутностей для розпізнавання, щоб унеможливити відсутність згенерованих запитань до речення. Стосовно методу на основі текстового корпусу - потрібно додавати нові правила та вдосконалювати існуючі, щоб система генерувала більш коректні запитання, а також покривала більшу кількість.

Оцінимо реалізовані методи за загальними критеріями, що представлені у таблиці 3.6:

Таблиця 3.7 – Порівняльний аналіз реалізованих методів

	Текстовий корпус	Шаблони
Швидкість	Задовільна	Задовільна
Граматичні помилки	Так	Так
Кількість запитань	Задовільна	Незадовільна
Відсутність згенерованих запитань до речення	Ні	Так
Нерелевантні запитання	Так	Ні

Отже, обидва методи демонструють гарну швидкість, але метод на основі текстового корпусу виграє як кількісно, так і якісно, оскільки він не тільки генерує більше запитань, але й не допускає ймовірності відсутності згенерованого запитання, хоча й може містити дещо некоректно згенеровані результати.

Безумовно, обидва методи є куди вдосконалювати та налагоджувати. Для методу на основі текстового корпусу це й розширення існуючих правил, налагодження граматичної структури згенерованих запитань, удосконалення відсіювання нерелевантних запитань. Щодо методу за шаблонами, то в першу чергу необхідно забезпечити краще навчання для NER-моделі і розширити спектр анотацій, які можуть бути визначені. У свою чергу це унеможливлює появу нових шаблонів, що спричинить кількісний зріст результатів генерації запитань.

3.4 Інструктивний матеріал користувача

3.4.1 Призначення та умови використання системи автоматичної генерації запитань

Система призначена для автоматичної обробки текстових файлів з метою автоматичної генерації запитань до вхідного тексту. Програма працюватиме коректно за умов надання тексту, в якому можна буде виділити речення та слова за допомогою токенізації. Це означає, що кожне речення повинно закінчуватися крапкою, а між словами повинен бути присутній пробіл.

3.5 Опис операцій користувача

Після запуску програми на екран виводиться інтерфейс системи генерації запитань, зовнішній вигляд якого представлений на рисунку 3.1. На ньому присутні наступні елементи керування, доступні користувачеві, які описані в таблиці 3.7.

Таблиця 3.7 - Елементи керування графічного інтерфейсу користувача

Елемент керування	Функція
Обрати файл	Функція вибору файлу з текстом, до якого будуть згенеровані запитання
Обрати метод	Функція вибору методу генерації запитань до тексту
Поле для вводу тексту	Надає можливість згенерувати запитання без вхідного файлу. Має перевагу над обраним файлом. Повинно бути пустим для обробки текстового файлу.
Згенерувати запитання	Функція запуску обробки обраного файлу або введеного тексту для генерації запитань обраним методом
Зберегти результат	Функція збереження списку згенерованих запитань
Відмінити	Функція відміни дії
Вийти	Функція виходу з програми

Опишемо загальний алгоритм використання системи.

1. Натиснути клавішу «Обрати файл», після чого у відкритому вікні вибрати файл або вписати шлях до файлу з розширенням .txt, до тексту

якого потрібно згенерувати запитання. Або є можливість вставки тексту в поле без необхідності вибору файла.

2. Обрати метод для генерації запитань. У системі наявні два мутоди: на основі текстового корпусу та за шаблонами.
3. В залежності від розміру тексту та потужності комп'ютера може знадобитися до двох хвилин, доки система обробить вхідний текст.
4. Після видачі згенерованих запитань, їх можна зберігти на локальному комп'ютері.

3.6 Можливі збої в роботі системи

При роботі системи з використанням методу на основі текстового корпусу можуть зустрічатися некоректно згенеровані запитання. Причиною тому може бути недостатня кількість продукційних правил, помилки роботи морфологічного аналізатору pymorphy, який може не впоратися з розпізнавання частини мови для слова. Значний вплив на результат роботи системи демонструє й складність та нестандартність побудови речення. Отже, буде доцільним перебудова таких речень або їх розбиття на менш складні речення, адже система демонструє гарні результаті при обробці односкладних речень.

При виборі методу генерації запитань за шаблонами може спостерігатися відсутність згенерованих запитань до окремих речень, де система не змогла розпізнати NER-сутності й відповідно підібрати шаблони запитань до них.

ВИСНОВКИ

У даній роботі була розглянута проблема автоматичної генерації запитань до українськомовного тексту. Актуальність даної проблеми обумовлена зростаючою потребою в удосконаленні обробки природної мови, а саме для української мови. Кожного дня розробляються нові чат-боти, нові курси дистанційного навчання, системи безпеки, які, безумовно, можуть бути вдосконалені системами автоматичної генерації запитань.

Основною метою даного дослідження були дослідження і розробка методів генерації запитань до українськомовного тексту. У ході виконання даної роботи вдалося виконати наступні задачі:

- здійснено пошук та дослідження методів генерації запитань та їх реалізацію до українськомовних текстів;
- модифіковано модель BERT для застосування NER-моделі для реалізації методу, заснованого на шаблонах;
- проведено аналітичний огляд існуючих програмних застосунків, що мають функцію генерації запитань;
- виконано порівняльний аналіз реалізованих методів генерації запитань.

Для реалізації системи генерації запитань були розглянуті різні методи, серед яких були обрані метод генерації запитань на основі текстового корпусу та метод заснований на шаблонах. Перший метод дозволяє уникнути залежностей від форми подання, обсягу і розміру початкового тексту, а також дозволяє генерувати запитання практично до будь-якого значимого члену речення. Для реалізації даного методу був обраний підхід NLP на основі

правил. Такий підхід заснований на розробці та розширенні існуючих правил, тому система не потребує масивного навчального корпусу, що, безперечно, є перевагою даного методу, як і гнучкість підходу, точність та можливість легко налагоджувати.

Модифікований метод генерації запитань за шаблонами було реалізовано з використанням NER-моделі та генерації шаблонів на основі NER-анотації. Хоча даний метод демонструє високу швидкість генерації запитань та можливість використання різних типів запитань, але

Отже, результатом проведеної роботи став модуль генерації запитань до українськомовного тексту з використанням двох методів. Обидва методи демонструють гарну швидкість, але метод на основі текстового корпусу виграє як кількісно, так і якісно, оскільки він не тільки генерує більше запитань, але й не допускає ймовірності відсутності згенерованого запитання, хоча й може містити дещо некоректно згенеровані результати.

Безумовно, обидва методи є куди вдосконалювати та налагоджувати. Для методу на основі текстового корпусу це й розширення існуючих правил, налагодження граматичної структури згенерованих запитань, удосконалення відсіювання нерелевантних запитань. Щодо методу за шаблонами, то в першу чергу необхідно забезпечити краще навчання для NER-моделі і розширити спектр анотацій, які можуть бути визначені. У свою чергу це унеможливлює появу нових шаблонів, що спричинить кількісний зріст результатів генерації запитань. Зважаючи на стрімкий розвиток наукових досягнень у сфері обробки природної мови, подальше вивчення даних технологій дозволить значно покращити результати отримані у даному дослідженні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Куртасов А.М. Программа генерации учебных тестов на основе семантического подхода/ Андрей Михайлович Куртасов. – Труды Международной научно-методической конференции "Информатизация инженерного образования", ИНФОРИНО. М.: Издательский дом МЭИ, 2012 – С. 71-74.

2. Братчиков И.Л. Генерация тестовых заданий в экспертно-обучающих системах/ Игорь Леонидович Братчиков. – Санкт-Петербург: Вестник Российского университета дружбы народов. Серия: Информатизация образования, 2012. – С. 47-60.

3. Wolfe J. H. Automatic Question Generation From Text – An Aid to Independent Study. ACM SIGCUE Outlook. – 1976. – Vol. 10, No. SI. – pp. 104-112.

4. Kunichika H. A Multimedia Language Learning Environment with Intelligent Tutor/ H. Kunichika, A. Takeuchi, and S. Otsuki // International Conference on Computers in Education, Taiwan. – 1993.

5. Kunichika H. Automated Question Generation Methods for Intelligent English Learning Systems and its Evaluation / H. Kunichika, T. Katayama, T. Hirashima, and A. Takeuchi // ICCE2004. – 2003.

6. Seneff S. TINA: A Natural Language System for Spoken Language Applications. Computational Linguistics, 1992. – Vol. 18, No. 1. – p. 61-86.

7. Baptist L. Genesis-II: A Versatile System for Language Generation in Conversational System Applications / L. Baptist, S. Seneffin // ICSLP. – Beijing, China. – 2000. – p. 271-274.

8. Xu Y. Mandarin Learning Using Speech and Language Technologies: A Translation Game in the Travel Domain," / Y. Xu, S. Seneff // ISCSLP. – Kunming, China. – 2008. – p. 29-32.

9. Xu Y.. Automatic Question Generation and Answer Judging: A Q&A Game for Language Learning / Y. Xu, A. Goldie, S. Seneff // Spoken Language Systems Group MIT Computer Science and Artificial Intelligence Laboratory. – United States. – 2009. – p. 2.

10. Rus V. The Question Generation Shared Task and Evaluation Challenge / V. Rus, A. C.Graesser // In Workshop on the Question Generation Shared Task and Evaluation Challenge, Final Report, The University of Memphis : National Science Foundation. – 2009.

11. Andrenucci A. Automated Question Answering : Review of the Main Approaches / A. Andrenucci , E. Sneiders // In Proceedings of the 3rd International Conference on Information Technology and Applications (ICITA'05). – Sydney, Australia. – 2005.

12. McGough J. A Web-based Testing System with Dynamic Question Generation / J. McGough, J. Mortensen, J. Johnson, S. Fadali // In ASEE/IEEE Frontiers in Education Conference. – 2001.

13. Wang W. Automatic Question Generation for Learning Evaluation in Medicine / Wang W., Tianyong H. & Wenyin L. // In LNCS – 2008. – Volume 4823.

14. Brown J. C. Automatic Question Generation for Vocabulary Assessment / J. C. Brown, G. A. Frishkoff, M. Eskenazi // In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. – Vancouver, British Columbia, Canada. – 2005.

15. Maybury M. T. New Directions in Question Answering. – 2004. – p. 320.

16. Hirchman L. Natural language question answering: the view from here / L. Hirchman, R.Gaizaukas. // Natural Language Engineering 7 (4): – 2001. – p. 275–300.

17. Burger J. Issues, Tasks and Program Structures to Roadmap Research in Question Answering (QA) / J. Burger, C. Cardie, V. Chaudhri, R. Gaizauskas, S. Harabagiu, D. Israel, C. Jacquemin, C-Y. Lin, S. Maiorano, G. Miller, Moldovan D. Ogden, B. Prager, J. Riloff, E. Singhal, A. Shrihari, R. Strzalkowski, T. Voorhees, E. Weishedel // Document Understanding Conference. – 2003.

18. Максимов В.И. Русский язык и культура речи: учебник / Максимов В.И., Голубева А.В: 2-е изд. СПб.: Златоуст, 2014. – 384 с.

19. Кручинин В.В. Использование деревьев И/ИЛИ для генерации вопросов и задач / Владимир Викторович Кручинин. – Томск: Вестник Томского государственного университета, 2004. – № 284. – С. 182-186.

20. Кручинин В.В. Использование графов для построения генераторов вопросов в компьютерных учебных программах / Владимир Викторович Кручинин. – Новосибирск: Вестник НГТУ, 2004. – № 3(18). – С. 187-194.

21. Кручинин В.В. Модели генераторов вопросов для компьютерного контроля знаний / В.В. Кручинин, Ю.В.Морозова – Томск: Научно-методический журнал. Открытое и дистанционное образование, 2004. № 2. – С. 36-42.

22. Сулейманов Д.Ш. Семантические технологии генерации учебных вопросов. / Сулейманов Д.Ш., Аюпов М.М., Невзорова О.А., Прокопьев Н.А.

Казань: Четырнадцатая национальная конференция по искусственному интеллекту с международным участием КИИ-2014., 2014. – С. 84-93.

23. Yao X. Semantics-based question generation and implementation / X. Yao, G. Bouma, Y. Zhang // Dialogue & Discourse. – 2012. – Vol. 3. – Is. 2. – p. 11-42.

24. Aquino J.F. Text2Test: Question Generator Utilizing Information Abstraction Techniques and Question Generation Methods for Narrative and Declarative Text / Chua D.D., Kabiling R.K., Pingco J.N. // Proceedings of the 8th National Natural Language Processing Research Symposium. – Manila. – 2011. – p. 29-34.

25. Система Quillionz [Електронний ресурс] // Quillionz. Режим доступу до ресурсу: https://www.quillionz.com/

26. Нейромережа ParaQG [Електронний ресурс] // ParaQG. Режим доступу до ресурсу: https://neurohive.io/ru/novosti/paraqg-nejroset-generiruet-voprosy-k-tekstu/

27. Lang-uk projects // Lang-uk. Режим доступу до ресурсу: https://lang.org.ua/en/

28. Spring Research Presentation: A Theoretical Foundation for Inductive Transfer, – 2017. – (Brigham Young University, College of Physical and Mathematical Sciences)

29. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding / J. Devlin, M-W. Chang, K. Lee, K. Toutanova // – 2019.

30. Polosukhin I. Attention Is All You Need / I. Polosukhin, K. Lukasz, G. Aidan, J. Llion, U.Jakob, P.Niki, S. Noam, V. Ashish // – 2017.

31. Molla D. NLP for Answer Extraction in Technical Domains. Proc. of EACL – Morgan Kaufmann, USA. – 2003.

32. Winiwater W. An Adaptive Natural Language Interface Architecture to Access FAQ Knowledge Bases. Proc. of the 4th Int. Conf. on Applications of N L to Information Systems. – 1999.

33. Chu W. Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation / W. Chu, and F. Meng // Technical Report 990003. – UCLA, USA. – 1999.

34. Lin J. The Web as a Resource for Question Answering: Perspective and Challenges. Proc. of LREC. – 2002.

35. Burke R. Question Answering from Frequently Asked Question Files: Experiences with the FAQ Finder System. AI Magazine. – 1997. – 18 (2). – p. 57-66.

36. Порівняння машинного навчання та систем, заснованих на правилах [Електронний ресурс] // Medium. – 2017. Режим доступу до ресурсу:

https://medium.com/friendly-data/machine-learning-vs-rule-based-systems-in-nlp-5476de53c3b8

37. Natural Language Toolkit [Електронний ресурс] // NLTK. Режим доступу до ресурсу:

http://www.nltk.org/

38. Husam A. Automation of question generation from sentences / A. Husam, C. Yllias, A.H. Sadid // In Proceedings of QG2010: The Third Workshop on Question Generation. – 2010. p. 58–67.

39. Saidalavi K.. Natural language question generation using syntax and keywords / K. Saidalavi, E. Ajeesh, and D. Rajarshi // In Proceedings of QG2010: The Third Workshop on Question Generation. – 2010. – p. 1–10.

ДОДАТКИ

Додаток А. Програмний код

```
import nltk

from nltk.tokenize import TweetTokenizer

import pymorphy2

from nltk.corpus import stopwords


tknzr = TweetTokenizer()


class QuestionGenerator():

def __init__(self, sentence):

self.sentence = sentence

self.morph = pymorphy2.MorphAnalyzer(lang='uk')

self.words = tknzr.tokenize(sentence)

self.count = 0

self.animNoun = ""

self.stop_words = set(stopwords.words("ukrainian"))

self.prevWord = ""


def getWords(self):

self.count = 0

for word in self.words:

self.count += 1

self.definePoS(word)

self.prevWord = word
```

```
self.prevWord = ""


def definePoS(self, word):
p = self.morph.parse(word)
question = self.words.copy()
if p is not None:
taggedWord = p.__getitem__(0)
pos = taggedWord.tag.POS
case = taggedWord.tag.case
gender = taggedWord.tag.gender
tense = taggedWord.tag.tense
if pos is not None:
if str(pos) == "NOUN":
if self.prevWord != "":
check = self.morph.parse(self.prevWord)
if check.__getitem__(0).tag.POS != 'INTJ':
self.generateNounQuestion(taggedWord, word, question, case)
else:
self.generateNounQuestion(taggedWord, word, question, case)
elif str(pos) == "VERB":
if tense is not None:
self.generateVerbQuestion(tense)
elif str(pos) == "ADJF":
self.generateAdjQuestion(word, question, gender)
```

```
def listToString(self, list):
resultString = ""
for word in list:
#if word not in self.stop_words:
resultString += word
resultString += " "
resultString = resultString.capitalize()
return resultString.replace(" ?", "?")

def generateNounQuestion(self, taggedWord, word, question, case):
animacy = taggedWord.tag.animacy
if animacy is not None:
if animacy == "anim":
self.animNounQuestion(word, question, case)
self.animNoun = word
elif animacy == "inan":
self.inanNounQuestion(word, question, case)

def animNounQuestion(self, word, question, case):
verb = ""
wordCount = 0
verbPosition = 0
flag = False
question.remove(word)
who = self.morph.parse('хто')[1]
```

```
insertWho = who.inflect({case}).__getitem__(0)

question.insert(0, str(insertWho))

for q in question:

p = self.morph.parse(q)

taggedWord = p.__getitem__(0)

if taggedWord.tag.POS == "VERB":

if flag:

verb = ""

else:

if 'infn' not in taggedWord.tag:

verb = taggedWord.inflect({'masc'})

verbPosition = wordCount

flag = True

wordCount+=1

if verb != "" and verb is not None:

question[verbPosition] = verb.word

question.remove(".")

question.append("?")

question = self.checkAdjective(question)

print(self.listToString(question))


def inanNounQuestion(self, word, question, case):

question.remove(word)

question.insert(0, "що")

question.remove(".")
```

```
question.append("?")
question = self.checkAdjective(question)
print(self.listToString(question))

def checkAdjective(self, question):
resultQuestion = question.copy()
for word in question:
p = self.morph.parse(word)
if str(p.__getitem__(0).tag.POS) == "ADJF":
resultQuestion.remove(word)
return resultQuestion

def generateVerbQuestion(self, tense):
todo = self.morph.parse('робити')[0]
todoTense = todo.inflect({tense})
if self.animNoun is not None:
normFormNoun = self.morph.parse(self.animNoun)[0].normal_form
gender = self.morph.parse(self.animNoun)[0].tag.gender
if gender is not None:
genderAction = todoTense.inflect({gender})
if normFormNoun is not None:
question = ['що', genderAction.__getitem__(0), normFormNoun, '?']
print(self.listToString(question))
```

```
def generateAdjQuestion(self, word, question, gender):
question.remove(word)
which = self.morph.parse('який')[0]
genderWhich = which.inflect({gender})
nounList = self.findNoun()
question.remove(nounList[0])
question.insert(0, nounList[0])
if nounList[1] != None:
insertWhich = genderWhich.inflect({nounList[1]}).__getitem__(0)
question.insert(0, str(insertWhich))
else:
question.insert(0, str(genderWhich.__getitem__(0)))

question.remove(".")
question.append("?")
question = self.checkAdjective(question)
print(self.listToString(question))

def findNoun(self):
counter = 0
nounCase = ""
noun = ""
for word in self.words:
counter += 1
if counter - self.count == 1:
```

```
noun = self.morph.parse(word)[0]

nounCase = noun.tag.case

result = [noun.__getitem__(0), nounCase]

return result
```

Додаток Б. Програмний код для використання моделі BERT

```
import bert

import tensorflow as tf


from tensorflow import keras as k


from config import BERT_DIR, BERT_WEIGHTS_PATH, MAX_LEN, CLASSES




def create_bert_layer() -> bert.BertModelLayer:

    bert_params = bert.params_from_pretrained_ckpt(BERT_DIR)

    bert_params.mask_zero = True


    bert_layer = bert.BertModelLayer.from_params(bert_params, name='bert')


    bert_layer.apply_adapter_freeze()


    return bert_layer




def create_model() -> k.Sequential:

    bert_layer = create_bert_layer()


    model = k.Sequential([

        k.layers.Input(shape=(MAX_LEN,), dtype='int32', name='input_ids'),
```

```
    bert_layer,
    k.layers.TimeDistributed(k.layers.Dense(768 * 3, activation=tf.nn.relu)),
    k.layers.TimeDistributed(k.layers.Dense(len(CLASSES), activation=tf.nn.softmax))
])

model.build()

bert_layer.apply_adapter_freeze()
bert.load_stock_weights(bert_layer, BERT_WEIGHTS_PATH)

model.compile(loss='categorical_crossentropy',
              optimizer=tf.optimizers.Adam(learning_rate=1e-4),
              metrics=['categorical_accuracy'])

return model
```

Додаток В. Програмний код для навчання NER-моделі

```
import os
from typing import List, NoReturn

import tensorflow as tf
from sklearn.model_selection import train_test_split

from config import CHECKPOINT_FILE_NAME, BATCH_SIZE
from models import create_model
from preprocessing import encode_inputs, encode_labels, tokenize



def prepare_training_data() -> (List[List[str]], List[List[str]]):
    import pandas as pd

    inputsList = []
    labelsList = []

    if not os.path.isdir('ner-uk'):
        import subprocess

        subprocess.run(['git',
                        'clone',
                        'https://github.com/lang-uk/ner-uk'])
```

```
for root, _, files in os.walk('ner-uk/data/'):
    for file in files:
        file_path = os.path.join(root, file)
        try:
            path_without_extension, extension = os.path.splitext(file_path)
            ann_path = path_without_extension + '.ann'

            if os.path.isfile(file_path) \
                and extension == '.txt' and not file_path.endswith('.tok.txt') \
                and os.path.isfile(ann_path):
                with open(file_path, 'r') as f:
                    file_content = f.read()

                file_row = pd.read_csv(ann_path,
                                       sep='\t',
                                       header=None,
                                       names=['Index', 'Type', 'Snippet'])
                file_row[['Type', 'Start', 'End']] = file_row['Type'].str.split(' ', expand=True)
                file_row[['Start', 'End']] = file_row[['Start', 'End']].astype(int)

                sub_inputs, sub_labels = tokenize(file_content, file_row)

                inputsList.extend(sub_inputs)
                labelsList.extend(sub_labels)
        except Exception as ex:
```

```
            print(f'{file_path} -> {ex}')

    return inputsList, labelsList



def load_training_data() -> (List[List[str]], List[List[str]]):
    import pickle

    inputs_path = 'train-inputs.pickle'
    labels_path = 'train-labels.pickle'

    if not os.path.isfile(inputs_path) or not os.path.isfile(labels_path):
        inputsList, labelsList = prepare_training_data()

        with open(inputs_path, 'wb') as file:
            pickle.dump(inputsList, file)
        with open(labels_path, 'wb') as file:
            pickle.dump(labelsList, file)
    else:
        with open(inputs_path, 'rb') as file:
            inputsList = pickle.load(file)
        with open(labels_path, 'rb') as file:
            labelsList = pickle.load(file)

    return inputsList, labelsList
```

```
def main() -> NoReturn:
    checkpoint_path = os.path.join(os.environ.get('CHECKPOINTS_DIR', '.'), CHECKPOINT_FILE_NAME)
    print(f'Checkpoint file: {checkpoint_path}')

    inputsList, labelsList = load_training_data()

    for i in reversed(range(len(labelsList))):
        any_with_class = False
        for j in range(len(labelsList[i])):
            if labelsList[i][j] != 'NONE':
                any_with_class = True
                break
        if not any_with_class:
            del inputsList[i]
            del labelsList[i]

    encoded_inputs_list = encode_inputs(inputsList)
    encoded_labels_list = encode_labels(labelsList)

    x_train, x_test, y_train, y_test = train_test_split(encoded_inputs_list,
                                                        encoded_labels_list,
                                                        test_size=0.15,
                                                        random_state=17)
```

```
print(f'Loaded {len(encoded_inputs_list)} samples')


model = create_model()


if os.path.isfile(checkpoint_path):
    model.load_weights(checkpoint_path)
    print(f'Weights is loaded from: {checkpoint_path}')


print(model.summary())


checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
                                                verbose=1,
                                                save_best_only=False,
                                                save_weights_only=True,
                                                save_freq=4*BATCH_SIZE)


def learning_scheduler(epoch):
    lr = 5e-5 * round(0.1 ** epoch, 10)
    return lr


learning_scheduler = tf.keras.callbacks.LearningRateScheduler(learning_scheduler, verbose=1)


callbacks = [checkpoint, learning_scheduler]


model.fit(x_train,
```

```
y_train,
batch_size=BATCH_SIZE,
validation_data=(x_test, y_test),
callbacks=callbacks,
epochs=3,
initial_epoch=int(os.environ.get('INITIAL_EPOCH', '0')),
verbose=1)
```