ДЕРЖАВНИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення та кібербезпеки

*Захищено на кафедрі інженерії програмного забезпечення та кібербезпеки*

*«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023р.*

*з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Підпис членів комісії:*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**КУРСОВА РОБОТА**

**з дисципліни**

**«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»**

**НА ТЕМУ:**

**«Розробка програмного забезпечення з генерації тексту за ключовим словом «Generatext» мовою програмування C#»**

**Виконала:** студентка факультету

інформаційних технологій

4 групи 2 курсу

Авєріної Наталії Ігорівни

**Науковий керівник:**

асистент кафедри інженерії програмного

забезпечення та кібербезпеки

**Хорольська Карина Вікторівна**

**Київ 2023**

**ЗМІСТ**

[**ВСТУП** 3](#_Toc133868104)

[**РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ПОШУК ЕФЕКТИВНОГО АЛГОРИТМУ З ГЕНЕРАЦІЇ ТЕКСТУ** 5](#_Toc133868105)

[**1.1.** **Історія розвитку розробок з генерації тексту** 5](#_Toc133868106)

[**1.2.** **Аналіз методів обробки природної мови** 6](#_Toc133868107)

[**1.3.** **Пошук ефективного алгоритму з генерації тексту** 8](#_Toc133868108)

[**Висновки до Розділу 1** 9](#_Toc133868109)

[**РОЗДІЛ 2 ПОРІВНЯЛЬНИЙ АНАЛІЗ НАЯВНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ** 11](#_Toc133868110)

[**2.1** **Порівняльний аналіз конкурентоспроможних на ринку продуктів з генерації тексту за ключовим словом** 11](#_Toc133868111)

[**2.2** **Вибір середовища та інструментів розробки** 15](#_Toc133868112)

[**Висновки до Розділу 2** 17](#_Toc133868113)

[**РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ «Generatext»** 19](#_Toc133868114)

[**3.1 Побудова структури коду за допомогою блок-схеми** 19](#_Toc133868115)

[**3.2 Технології, використані для розробки програмного продукту** 19](#_Toc133868116)

[**Висновок до розділу 3** 28](#_Toc133868117)

[**ВИСНОВОК** 29](#_Toc133868118)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 30](#_Toc133868119)

# **ВСТУП**

В сучасному світі інформаційних технологій стрімко набуває популярності генерація тексту на основі ключового слова. Це процес автоматичної генерації текстових даних з використанням алгоритмів машинного навчання, що дозволяє створювати великі обсяги тексту з мінімальною людською участю.

Цей підхід знайшов своє застосування в багатьох сферах, таких як соціальні мережі, засоби масової інформації, рекламні компанії, інформаційні портали, блоги та багато інших. Генерація тексту на основі ключового слова є важливим інструментом у веб-розробці та цифровому маркетингу, адже дозволяє автоматично створювати унікальний контент, що збільшує рейтинг сторінок у пошукових системах. Тож використання цієї технології дозволяє ефективно і швидко генерувати текстові дані на різноманітні теми, що значно підвищує продуктивність та ефективність роботи в різних сферах діяльності.

Оскільки генерація тексту на основі ключового слова є складним процесом, що вимагає обробки природної мови, у цій роботі будуть досліджені різні підходи до створення таких систем та їх ефективність. Також будуть проаналізовані можливості використання різних алгоритмів та моделей машинного навчання, що дозволять покращити якість генерації тексту на основі ключового слова.

**Завдання курсової роботи:**

* дослідити моделі та методи генерації тексту;
* проаналізувати наявні на ринку додатки-конкуренти
* розробити додаток, який дозволяв би генерувати текст за введеним ключовим словом від користувача.

**Метою** даної курсової роботи є дослідження можливостей застосування методів генерації тексту на основі ключового слова та розробка програмного додатку з генерації правдоподібного тексту з використанням цих методів.

**Об’єктом дослідження** є алгоритми та моделі, що дозволяють автоматично генерувати текст на основі заданого ключового слова.

**Предмет дослідження:** розробка програмного забезпечення з генерації тексту за ключовим словом.

Під час виконання проєкту буде створений додаток під назвою “Generatext” мовою програмування C# з використанням платформи .NET Framework.

# **РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ПОШУК ЕФЕКТИВНОГО АЛГОРИТМУ З ГЕНЕРАЦІЇ ТЕКСТУ**

## **Історія розвитку розробок з генерації тексту**

Історія генерації тексту сягає середини 20 століття, коли було створено перші системи, що використовували шаблони для автоматичного створення тексту.

У 1950-х роках була започаткована робота над програмою під назвою «Random Sentence Generator», що генерувала випадкові речення за допомогою комп’ютера. Методом генерації був випадковий вибір слів із заданого словника [3].

У 1970-х роках Террі Віноградом в рамках його докторської дисертації в Массачусетському технологічному інституті була створена комп’ютерна програма SHRDLU. Метою було дослідження можливостей комп’ютера в розумінні мовлення людини і його взаємодії з навколишнім світом через мову. Система мала можливість генерації речень на основі правил граматики [4].

У 1980-х роках з’явилася можливість використання статистичних методів для генерації тексту. Один з перших проектів, що використовував цей підхід, був розроблений групою дослідників у Каліфорнійському університеті в Берклі. Вони використали статистичні моделі для генерації заголовків новин, що були схожі на ті, що створюють журналісти [5]. Цей проєкт використовував статистичні методи для аналізу структури речень та використання цієї інформації для генерації тексту. Зокрема, вони використовували статистичні моделі для визначення того, які слова повинні бути використані в реченні, та які слова повинні йти разом. Наприклад, якщо модель бачила слово "поліція", то вона вважала, що ймовірно також з’являться слова "арешт", "злочинець" та "слідство" [6].

З появою штучного інтелекту та зростанням обчислювальної потужності розпочався новий етап розвитку генерації тексту. Сьогодні, методи машинного навчання та обробки природньої мови є ключовими інструментами для генерації тексту. Один з таких методів – n-грам модель, яка використовується для прогнозування наступного слова в тексті з урахуванням його попередніх слів.

Зараз активно розробляються нові методи генерації тексту, такі як глибокі нейронні мережі та генеративні моделі, що використовуються для створення більш точних та якісних текстів. Крім того, з'являються нові області застосування генерації тексту, такі як автоматичне створення коду програм та складання музики.

* 1. **Аналіз методів обробки природної мови**

Методи обробки природної мови (англ. Natural Language Processing) – це галузь комп’ютерної лінгвістики, яка досліджує можливості застосування комп’ютерних алгоритмів для аналізу та розуміння природної мови. Для розвитку генерації тексту за ключовим словом та інших супутніх технологій важливим є використання методів обробки природної мови. [7]

Основні методи NLP включають у себе:

* Токенізацію: процес розбиття тексту на окремі компоненти, називані токенами, які можуть бути словами, числами, символами пунктуації тощо. Цей процес є важливим етапом у багатьох проєктах з обробки природної мови, оскільки вона дозволяє перетворити текст на структурований формат, який можна подальше обробляти та аналізувати. Наприклад, використання токенів є важливим етапом для створення n-грам, що дозволяє аналізувати залежності між словами у тексті. [7]
* Частиномовний аналіз: процес визначення ролі кожного слова у реченні. Він допомагає розуміти граматичну структуру тексту та робити висновки про зміст. [7]
* Синтаксичний аналіз (також відомий як парсинг): процес аналізу тексту на основі граматики, що визначає правила правильної синтаксичної структури мови. Під час синтаксичного аналізу, вхідний текст розбивається на послідовність лексем (токенів), і ці лексеми потім перетворюються в дерево синтаксичного розбору згідно з правилами граматики. [7]
* Семантичний аналіз: процес визначення значення слова та його зв’язок з іншими словами у реченні. На етапі семантичного аналізу використовуються різні методи та техніки, такі як семантичні мережі, векторні представлення слів, методи машинного навчання та інші. Для покращення точності семантичного аналізу важливо мати достатньо великий корпус текстів, на яких можна навчати моделі та алгоритми. [7]
* Розпізнавання іменованих сутностей: процес визначення в тексті іменованих об’єктів, таких як люди, місця, організації та інші. Ця задача важлива для багатьох застосувань обробки мови, таких як автоматичний переклад, аналіз соціальних медіа, аналіз текстів новин, інформаційний пошук та багато інших. Одним з відомих наборів даних для розпізнавання іменованих сутностей є CoNLL-2003, що містить новинні тексти з позначеними іменованими сутностями. Також існують відкриті бібліотеки, такі як spaCy, NLTK, Stanford NER та інші. [7]
* Класифікацію текстів: процес визначення категорії, до якої належить текст. Це важливий етап у багатьох задачах обробки природної мови, таких як фільтрація спаму, аналіз настроїв, категоризація новин, розпізнавання мовленнєвих актів та багато інших. [7]
* Аналіз тональності: процес визначення настрою тексту: позитивний, негативний чи нейтральний. Для реалізації аналізу тональності можна використовувати різні бібліотеки та інструменти, такі як TextBlob, NLTK, Vader Sentiment та багато інших. Застосування аналізу тональності дуже широке, від аналізу відгуків користувачів в інтернет-магазинах до моніторингу соціальних мереж для виявлення настроїв аудиторії щодо певної події чи товару. [7]
* Генерацію мовлення: процес створення тексту з допомогою комп’ютерної програми.

Одним з методів генерації мовлення є шаблонний підхід, при якому використовуються заздалегідь написані шаблони речень, які заповнюються відповідними даними. Цей метод досить простий, але не дозволяє створювати складні та оригінальні текстові висловлювання.

Ще одним методом є статистичний підхід, при якому комп’ютерний алгоритм аналізує велику кількість текстів та статистичні дані щодо вживання певних слів та фраз в різних контекстах. Потім алгоритм використовує ці дані для створення нових речень, які відповідають заданому контексту. Цей метод може створювати більш складні та оригінальні текстові висловлювання, але його точність залежить від якості та кількості вихідних даних, на основі яких він працює.

Іншим методом є глибинне навчання, при якому комп’ютерні алгоритми навчаються на великій кількості даних та самостійно здатні створювати текстові висловлювання на основі отриманих знань. Цей метод може створювати найбільш складні та оригінальні текстові висловлювання, але його точність залежить від якості та кількості навчальних даних. [8]

* 1. **Пошук ефективного алгоритму з генерації тексту**

Пошук ефективного алгоритму для генерації тексту є актуальною проблемою в галузі обробки природних мов. Один з потенційних методів – використання n-грам. N-грами – це метод статистичної обробки природної мови, який використовується для аналізу та генерації тексту. Він базується на розбитті тексту на окремі фрагменти, зазвичай по словам, та аналізу частоти їх вживання поруч. Наприклад, для створення n-грам на основі трьох слів, розбивають текст на три слова та рахують, скільки разів кожна трійка слів зустрічається в тексті.

Один з ефективних алгоритмів генерації тексту на основі n-грам – це Markov Chain. Він полягає в тому, що збирається велика кількість текстів, які використовуються для тренування моделі, та створюється таблиця з частотами вживання трійок слів. Потім для генерації нового тексту вибирається випадкова трійка слів, знаходиться найбільш ймовірна наступна трійка слів, що може йти після неї, і використовується для створення наступного слова. Цей процес повторюється доти, доки не буде сформований весь текст. [6]

В даному проєкті буде використовуватися один з основних методів обробки природної мови – токенізація, та після неї створення n-грам. Програма буде приймати текст, який буде розділено на список слів. Після чого обчислюється кількість входження кожної біграми (пари сусідніх слів) в текст. Результатом є словник біграм, де ключі - перше слово в біграмі, а значення - словник, де ключі - друге слово в біграмі, а значення - кількість входжень біграми в текст. Так само буде зроблено і з триграмами. Результатом має бути словник, де ключі - біграми/триграми, а значення - наступне слово, що найчастіше зустрічається після цієї біграми/триграми.

## **Висновки до Розділу 1**

Генерація тексту є важливою задачею в галузі обробки природної мови. Історія розвитку цієї галузі досліджень показує, що з появою більш потужних обчислювальних систем і набору великої кількості текстових даних з’явилася можливість створювати більш складні моделі для генерації тексту.

Методи обробки природної мови, такі як аналіз тональності, класифікація текстів та генерація мовлення, розвиваються з метою покращення різних аспектів обробки текстів. Пошук ефективного алгоритму з генерації тексту є важливою задачею, яка може бути вирішена за допомогою різних підходів. У цьому контексті використання n-грам є одним з найпоширеніших методів генерації тексту. Використання n-грам дозволяє моделювати залежності між словами в тексті та генерувати більш природні тексти.

# **РОЗДІЛ 2 ПОРІВНЯЛЬНИЙ АНАЛІЗ НАЯВНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ**

* 1. **Порівняльний аналіз конкурентноспроможних на ринку продуктів з генерації тексту за ключовим словом**
  2. **Порівняльний аналіз конкурентоспроможних на ринку продуктів з генерації тексту за ключовим словом**

Для проведення порівняльного аналізу конкурентоспроможних на ринку продуктів з генерації тексту за ключовим словом необхідно спочатку визначитись з конкретними продуктами.

За запитом «text generation» в пошуковій системі Google можна виявити такі популярні програми на ринку:

* Chat GPT - це продукт компанії OpenAI, який базується на глибоких нейронних мережах та забезпечує високу якість генерації тексту. GPT може використовуватись для автоматичної генерації новин, текстів для соціальних мереж, статей, творів, віршів тощо. Переваги:
  + Адаптивність. Генератор адаптується до різних контекстів та задач, що забезпечує його універсальність в різних сферах застосування.
  + Багатомовність. Chat GPT розпізнає та генерує текст на багатьох мовах світу, що робить його достойним інструментом для використання в інтернаціональних проєктах та звичайним помічником для людей в кожній точці планети.
  + Зручний та зрозумілий інтерфейс зроблений у вигляді чату.
  + Висока якість генерації тексту. Генератор може вчитися на великих обсягах текстових даних та зберігати ці знання, що дозволяє покращувати якість генерації тексту.

Хоча Chat GPT має багато переваг, але він також має деякі недоліки:

* + Модель може генерувати відповіді, які здаються граматично правильними, але зовсім не відповідають запиту.
  + Іноді виникають повтори фраз або ідей, які вже були використані.
  + Модель може бути вимогливою до обчислювальних ресурсів
  + Питання безпеки використання. Chat GPT навчається на великій кількості тексту, включаючи переписки, соціальні мережі та інші джерела. Це означає, що при розмові з Chat GPT можуть виникати ситуації, коли система може отримувати конфіденційну інформацію, таку як персональні дані, фінансову інформацію, медичні записи та інше. Такі дані можуть стати доступними поганцям, якщо система не забезпечує належний рівень безпеки.
* TextGenrnn - це відкрите програмне забезпечення для генерації тексту, яке базується на рекурентних нейронних мережах. Його можна використовувати для генерації різних типів текстів, включаючи новини, вірші, історії та інше.

Переваги:

* + Добре підходить для генерації тексту з довільною темою і стилем.
  + Простий та зрозумілий інтерфейс.
  + Має можливість використовувати вагові коефіцієнти, що дає змогу змінювати важливість певних слів та фраз при генерації тексту.

Недоліки:

* Недостатньо ефективний при генерації довгих текстів з складною граматикою або технічних текстів.
* Зазвичай потребує досить велику кількість тренувальних даних для досягнення достатнього рівня точності.
* Може бути схильний до генерації нелогічних та неправдоподібних фраз або повторів.
* Обмежена функціональність.
* IBM Watson - це рішення зі штучного інтелекту, яке забезпечує генерацію тексту з використанням природної мови. IBM Watson може використовуватись для автоматичної генерації новин, статей, описів продуктів та іншого.

Переваги:

* Широкий спектр інструментів і можливостей для роботи з природною мовою, таких як генерація тексту, розпізнавання мовлення, переклад тексту та інші.
* Можливість використовувати IBM Watson на різних мовах, що робить його більш універсальним і доступним для різних користувачів.
* Інтеграція зі сторонніми системами та додатками, що дозволяє легко використовувати IBM Watson в існуючих проєктах.
* Великий обсяг документації, навчальних матеріалів та ресурсів, що допомагає швидко навчитися користуватися платформою.

Недоліки:

* Обмеження в налаштуванні та кастомізації платформи, що може обмежувати розвиток індивідуальних рішень.
* Питання безпеки та конфіденційності даних при роботі з платформою.
* Складність використання.
* Markovify: це простий генератор тексту, який використовує модель Маркова для генерації тексту.

Переваги:

* Легкий у використанні та налаштуванні.
* Може працювати з будь-яким типом текстів та мовами.
* Можливість налаштувати рівень складності моделі, щоб підібрати оптимальне співвідношення між точністю та швидкістю генерації тексту.
* Відкритий код та широкий спектр підтримки.

Недоліки:

* Не здатний згенерувати довгі, складні або логічно зв'язані тексти.
* Не завжди здатний зберегти стиль, тон або специфіку вихідного тексту.
* Не здатний генерувати тексти з відтворенням граматичної правильності.
* Не здатний згенерувати тексти з новими ідеями, що відрізняються від вихідного корпусу.

Створений нами продукт «Generatext», який працює на основі використання n-грамної моделі генерації тексту, буде мати наступні переваги порівняно з конкурентами:

* Висока швидкість обробки. Оскільки додаток «Generatext» має меншу кількість параметрів, використовує просту статистичну модель та не вимагає високопродуктивного обладнання, як наприклад GPT-3, працювати він буде ефективно та значно швидше.
* Низькі витрати. Розробка генератора тексту на основі n-грам не потребує великих витрат на обладнання та програмне забезпечення, що робить його доступним для широкого кола користувачів.
* Вбудована підтримка мовних моделей. Це дозволяє покращити якість генерації тексту, користуючись найновішими досягненнями у галузі обробки природної мови.
* Дуже простий інтерфейс. Користувачі без досвіду в програмуванні та користуванні найновішими технологіями зможуть швидко та легко згенерувати текст за ключовим словом, виконуючи прості дії, які будуть вказані на екрані.

Недоліки, які в майбутньому будуть виправлятися з впровадженням нових технологій:

* Не здатний згенерувати довгі, складні або логічно зв'язані тексти.
* Може бути схильний до генерації нелогічних та неправдоподібних фраз або повторів.
* Обмежена функціональність.
* Може працювати лише з однією мовою.
  1. **Вибір середовища та інструментів розробки**

Мова програмування є необхідним інструментом у написанні будь-якої програми. Аналогічно до людського спілкування – не знаючи мови складно пояснити щось так щоб тебе зрозуміли. Так само і в програмуванні – комп’ютер не зрозуміє яку задачу треба виконати і як, якщо написати код незрозумілою йому мовою, наприклад українською.

Мова програмування – це формальний мовний засіб, який використовується для написання програмного коду, який згодом буде виконуватись комп’ютером або іншою обчислювальною машиною.

Суворіше визначення: мова програмування — це система позначень для опису алгоритмів і структур даних, певна штучна формальна система, засобами якої можна виражати алгоритми. Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми та дії, які виконує виконавець (комп’ютер) під її управлінням.[1]

Історія мов програмування починається з 19 століття, коли було розроблено перші "машини на картах", які використовували спеціальні отвори в перфокартах для виконання обчислювальних операцій. У 20-му столітті з’явилися перші електронні комп’ютери, для яких були розроблені високорівневі і низькорівневі мови програмування. [1]

Перша високорівнева мова програмування – FORTRAN, була розроблена 1950-х роках для наукових обчислень. Пізніше з’явилися інші мови високого рівня, такі як COBOL, BASIC та C, які використовуються і до сьогодні. [1]

У 1980-х роках були розроблені об’єктно-орієнтовані мови програмування, такі як C++, які дозволяють програмістам створювати складні програми з більшим рівнем абстракції та повторного використання коду.

З появою Інтернету та розширенням індустрії програмного забезпечення з’явилися нові мови програмування, такі як JavaScript, Python та Ruby. Ці мови програмування використовуються для веб-розробки, машинного навчання, аналізу даних та багатьох інших задач.

Сьогодні існує безліч мов програмування, кожна з яких має свої особливості та призначення. Вибір мови програмування залежить від задачі, яку потрібно вирішити, та від особистих уподобань програміста.

Саме для реалізації системи генерації тексту за ключовим словом необхідні спеціальні інструменти, що дозволяють створювати та редагувати словник N-грамм. Один з найбільш ефективних інструментів для створення та редагування текстових даних є мова програмування Python, зокрема бібліотека NLTK, яка надає можливості для обробки природних мов та створення N-грамм.

Але для розробки такої системи було обрано середовище програмування Visual Studio та мову програмування C#. Visual Studio було обрано з огляду на те, що це середовище програмування дозволяє швидко створювати різноманітні застосунки та інтерфейси, має велику кількість вбудованих інструментів для автоматизації рутинних задач та відлагодження коду. Мова програмування C# була обрана з огляду на її об’єктно-орієнтовану структуру та високу продуктивність.

Об’єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

ООП дозволяє створювати більш складні програми з більшими обсягами коду, які можуть бути більш легко підтримувані, змінювані та розширювані. Програмування в ООП також дозволяє зменшити кількість дублюючого коду, що полегшує розробку та зберігання програмного забезпечення.

Основні принципи ООП:

* Інкапсуляція - властивість об'єктів, що дозволяє об'єднувати дані та методи, які працюють з цими даними, в єдиний об'єкт. Це дозволяє забезпечити приховування деталей реалізації та підвищити безпеку програми.
* Наслідування - процес, коли один клас наслідує властивості та методи іншого класу. Це дозволяє зменшити кількість коду та підвищити його читабельність.
* Поліморфізм - властивість об'єктів, що дозволяє їм виконувати різні функції, залежно від контексту виклику. Це дозволяє забезпечити більш гнучкий та складний функціонал.
* Абстракція - процес визначення загальних характеристик об'єктів та процесів, відокремлення їх від деталей реалізації. Це дозволяє забезпечити більш зрозумілу та легко змінювану архітектуру програмного продукту.

Використання принципів ООП дозволяє забезпечити більш гнучку та масштабовану архітектуру програмного продукту, що дозволяє легко додавати новий функціонал та збільшувати його ефективність та надійність.

Також додатково було використано такі інструменти:

* .NET Framework - це платформа для розробки та виконання програмного забезпечення, яка надає зручний інтерфейс для програмування мовами, такими як C#.
* LINQ (англ. Language Integrated Query - запити, інтегровані в мову) – компонент Microsoft .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET[2].
* Git - це розподілена система контролю версій, яка дозволяє зберігати та відстежувати зміни в коді проекту. Git надає можливість повернутися до будь-якої попередньої версії коду, яка була збережена, що дозволяє відновити роботу в разі помилки чи неправильної зміни коду.

Якщо програма розробляється декількома розробниками або навіть компанією, то Git дозволяє розробникам працювати з одним і тим же кодом, зберігаючи кожну зміну як версію. Кожен розробник може працювати зі своєю власною копією коду та об’єднувати зміни з іншими розробниками, що робить роботу над проектом більш організованою та ефективною.

А для того, щоб мати доступ до свого проєкту в будь-якому місці та з будь-якого комп’ютера, або передати код на розгляд або правки іншим розробникам, використовувався хмарний сервіс GitHub для зберігання та управління проектами програмного забезпечення. Для зручного керування версіями та виконання комітів використовувався графічний клієнт GitHub Desktop.

## **Висновки до Розділу 2**

Проведено порівняльний аналіз продуктів з генерації тексту за ключовим словом, таких як Chat GPT, TextGenrnn, IBM Watson та Markovify, на конкурентному ринку товарів. Підсумовуючи, можна сказати, що різні генератори тексту мають свої переваги та недоліки, і вибір продукту користувачем залежить від конкретної задачі та потреб.

Наш генератор тексту буде мати декілька переваг порівняно з конкурентами, такі як: відмінна швидкість генерації тексту, легкість у використанні та зрозумілість алгоритму. Однак, у порівнянні з іншими генераторами тексту, може бути менш точним та менш ефективним у генерації більш складних текстів, які мають велику кількість різноманітної інформації.

Вибір Visual Studio та мови програмування C# дозволить швидко та ефективно створити систему генерації тексту за ключовим словом з високою продуктивністю та широкими можливостями для налагодження та тестування. А користування .NET Framework, LINQ та Git сприяє покращенню продуктивності та якості розробки проєкту.

# **РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ «Generatext»**

## **3.2 Технології, використані для розробки програмного продукту**

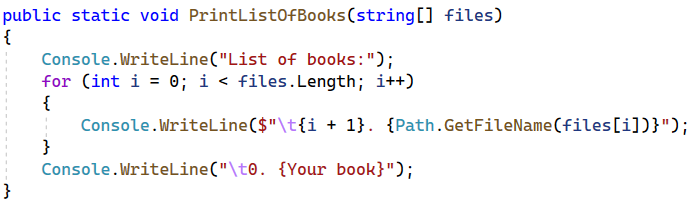
Даний програмний додаток, як вже було сказано, розроблено мовою програмування C# та побудовано із застосуванням шаблону «Console App (.NET Framework)». Шаблони надають користувачам деякий базовий код та структуру, які спрощують початок написання власного коду.

Мій програмний продукт містить 5 класів, серед яких:

* **BookSelector.cs**

Даний клас містить методи, що потрібні під час вибору користувачем книги та введення ним даних, необхідних для генерації тексту. Всі методи оголошені з модифікатором public, бо використовуються в інших класах та викликають інші методи всередині своєї реалізації.

Метод **PrintListOfBooks(string[] files)** друкує в консоль список книг, а якщо точніше – список файлів з вказаної директорії. Користувач з цього списку має вибрати номер файлу (книги, завдяки якій буде відбуватися генерація) або цифру ‘0’ – файл користувача (рис. 3.4).



**Рис. 3.4 Виведення списку файлів у консоль**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

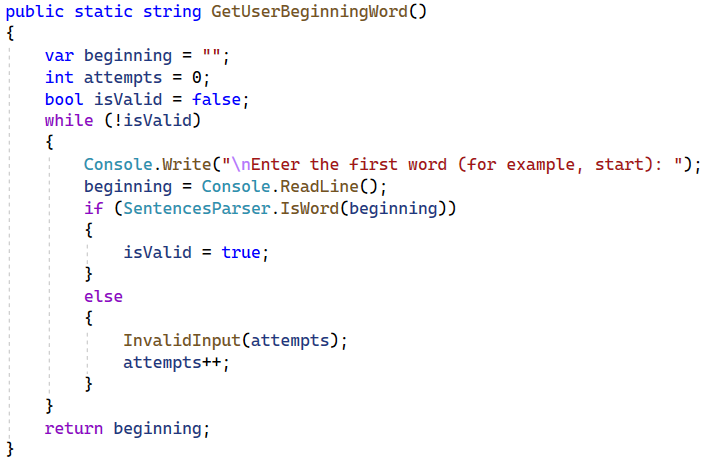
Після чого метод **BookChoice(string[] files*)*** вже безпосередньо отримує від користувача номер обраного файлу і передає текст з нього в змінну *text* (рис. 3.5).



**Рис. 3.5 Обробка вибору файлу користувачем**

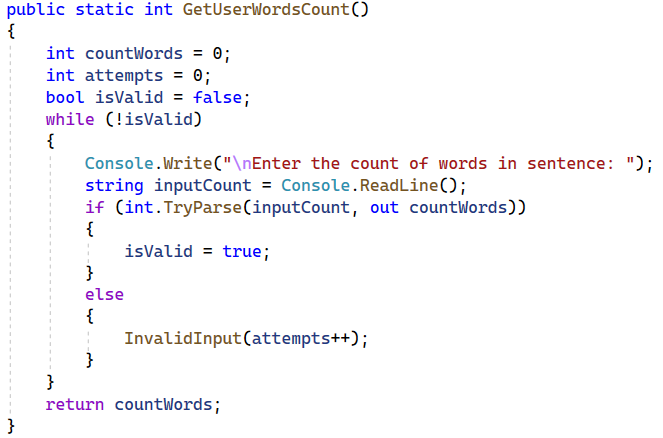
*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

Далі методи **GetUserBeginningWord()** (рис. 3.6) та **GetUserWordsCount()** (рис. 3.7) обробляють введені користувачем значення ключового (першого) слова та кількості слів у реченні відповідно, після чого вертають їх.



**Рис. 3.6 Обробка введеного ключового слова**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

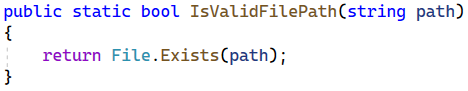


**Рис. 3.7 Обробка введеного значення кількості слів**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

Щоб уникнути помилок виконання програми, якщо користувач введе недопустиме значення (наприклад літеру замість цифри), в кожному методі, що приймає значення від користувача, виконуються перевірки.

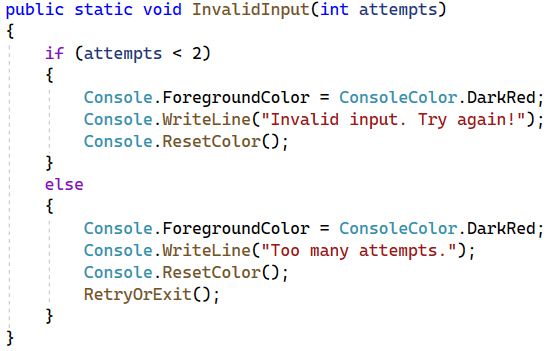
Наприклад для того, щоб перевірити чи правильно було введено шлях до користувацького файлу або чи існує він взагалі, використовується метод **IsValidFilePath(string path)** (рис. 3.8).



**Рис. 3.8 Перевірка існування файлу за введеним шляхом**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

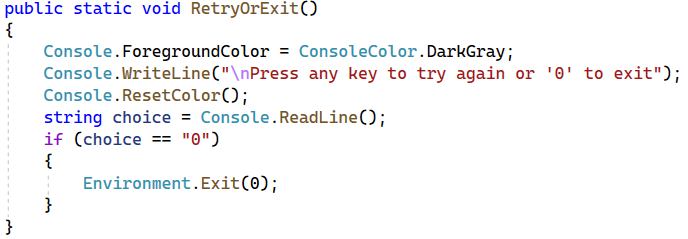
Для того, щоб повідомляти користувача, що той ввів недопустиме значення, виконується метод **InvalidInput(int attempts)** (рис. 3.9).



**Рис. 3.9 Виведення в консоль повідомлень про помилку**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

Після третьої спроби (лічильником є змінна *attempts*) програма пропонує користувачу вибір: продовжити чи завершити виконання програми (за допомогою метода **RetryOrExit()**) (рис. 3.10).

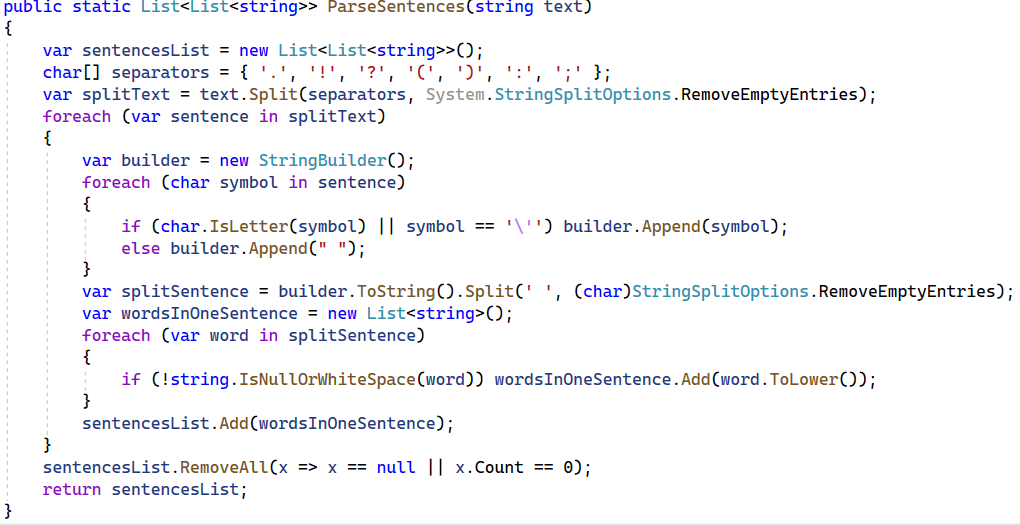


**Рис. 3.10 Вибір щодо продовження виконання програми**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

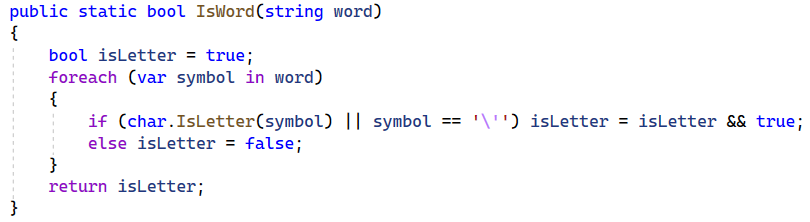
* **SentencesParser.cs**

Цей клас виконує розбиття тексту, на основі якого буде відбуватися генерація, на речення та слова. Для того, щоб розділити текст на речення, був створений метод **ParseSentences(string text)**, який приймає текст типу *string* з файлу та вертає *List<List<string>>()* (рис. 3.11)*.* В ньому використовується вбудований метод **string.Split** та створений власноруч метод **IsWord(string word)** типу *bool* (рис. 3.12), який перевіряє, чи є символи в реченні буквами. Після перевірки символів використовується рядковий клас ***StringBuilder***, визначений у просторі імен *System.Text*.



**Рис. 3.11 Виконання розбиття тексту**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*



**Рис. 3.12 Перевірка чи є частина розбитого тексту словом**

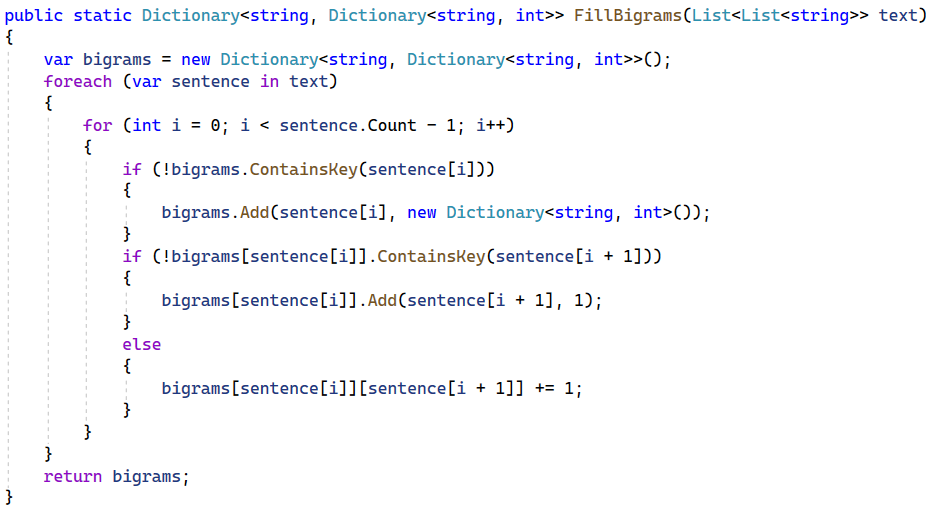
*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

* **FrequencyAnalysis.cs**

В даному класі створюється словник найчастіших продовжень біграм та триграм.

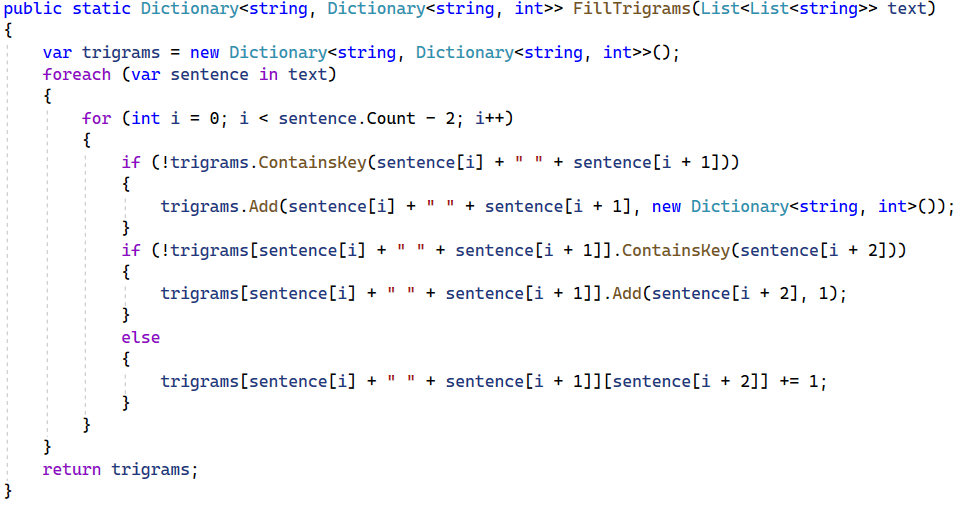
Наприклад, з тексту: *"She stood up. Then she left."* можна виділити такі біграми *"she stood"*, *"stood up"*, *"then she"* та *"she left"*, але не *"up then"*. І дві триграми *"she stood up"* та *"then she left"*, але не *"stood up then"*.

Отже, результатом використання цього класу є словник ***result***, ключами якого є всі можливі початки біграм і триграм, а значеннями їх найчастіші продовження. Якщо є кілька продовжень з однаковою частотою, використовуються ті, що лексикографічно менше. Для лексикографічного порівняння був використаний вбудований у .NET спосіб порівняння ***string.CompareOrdinal.***



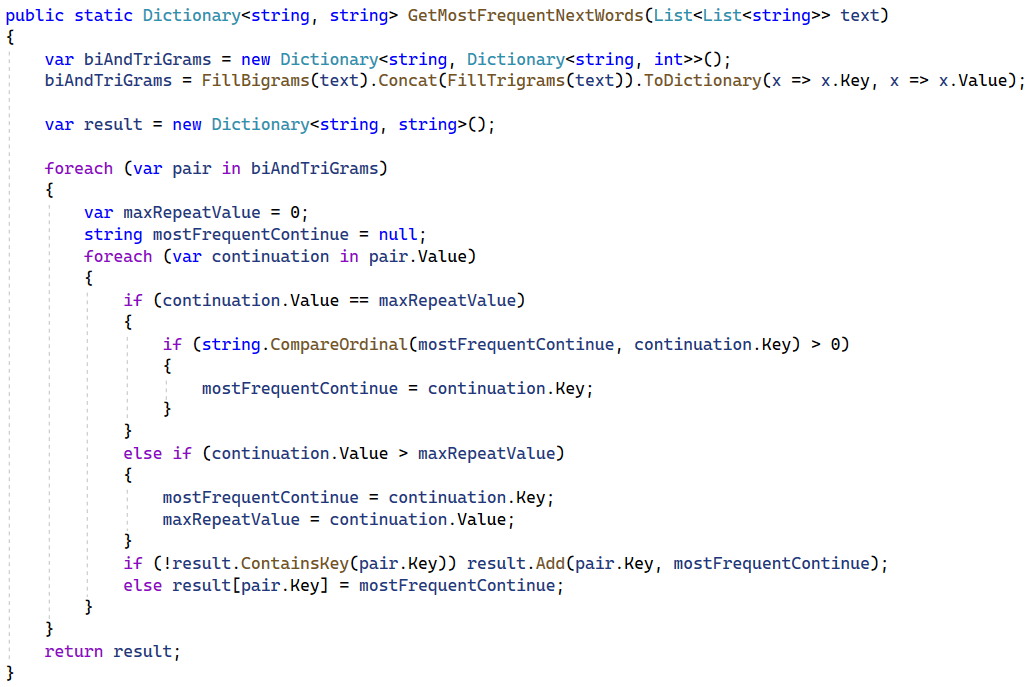
**Рис. 3.13 Створення словнику біграм**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*



**Рис. 3.14 Створення словнику триграм**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*



**Рис. 3.15 Отримання найбільш частого продовження**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

* **TextGenerator.cs**

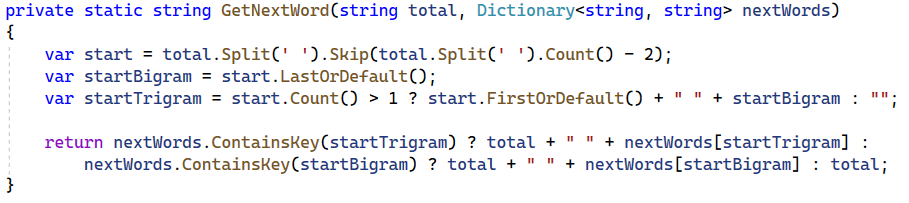
На вхід алгоритму передається словник **NextWords**, отриманий у класі **FrequencyAnalysis.cs**, одне або кілька перших слів фрази **phraseBeginning** і **wordsCount** — кількість слів, які потрібно дописати до початкової фрази.

Інакше, якщо у словнику є ключ, що складається з одного останнього слова фрази, то продовжувати потрібно словом, що зберігається у словнику цього ключа.

Інакше потрібно достроково закінчити генерування фрази і повернути згенерований на даний момент результат.

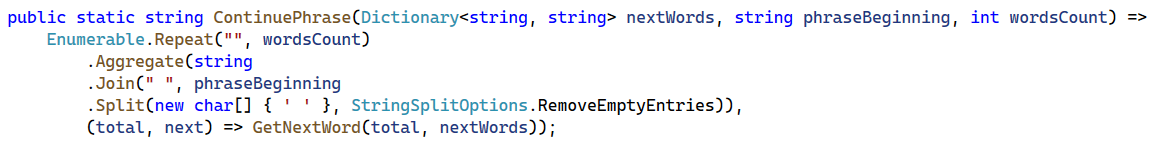
Метод **GetNextWord** розділяє рядок total на слова, використовуючи пробіл як роздільник, і відбирає останнє слово, яке є початком наступного слова. Якщо у total міститься більше одного слова, метод відбирає останні два слова, які є початком наступного слова.

Далі метод перевіряє, чи містить словник nextWords наступні слова після останнього слова total. Якщо так, метод додає наступне слово в рядок total і повертає його. Якщо словник не містить наступних слів після останнього слова total, метод перевіряє, чи є наступне слово, що може йти після передостаннього слова. Якщо так, метод додає наступне слово в рядок total і повертає його. Якщо і це не можливо, метод повертає рядок total без змін.



**Рис. 3.16 Отримання наступного слова**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*



**Рис. 3.17 Продовження фрази отриманим наступним словом**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

* **Program.cs**

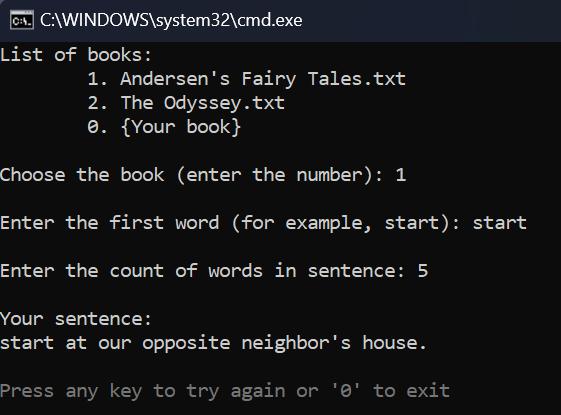
Цей клас містить метод **Main** (рис. 3.18), що є точкою входу до програми. В ньому виконуються початкові налаштування та викликаються інші методи, які виконують основну роботу додатку.



**Рис. 3.18 Виклик методів в основній програмі, генерація тексту**

*Джерело: розроблено автором у додатку Visual Studio (знімок з екрану)*

Результатом роботи програми є виведення в консоль списку книг, серед яких користувач обирає будь-яку або вводить шлях до своєї. Після чого він вводить необхідну для генерації інформацію та отримує речення. (рис. 3.19)



**Рис. 3.19 Результат роботи програми в інтерпретаторі командного рядка**

*Джерело: розроблено автором у додатку cmd.exe (знімок з екрану)*

## **Висновок до розділу 3**

Програмний продукт було розроблено мовою програмування C# та побудовано із застосуванням шаблону «Console App (.NET Framework)». Для роботи програми було створено 5 класів: BookSelector.cs (вибір книги користувачем), SentencesParser.cs (розділення тексту на слова), FrequencyAnalysis.cs (отримання найбільш частого продовження слова), TextGenerator.cs (генерація тексту) та Program.cs (виклик методів, виконання основної роботи додатку). Результатом роботи програми є виведення даних комп’ютером і введення даних користувачем в інтерпретаторі командного рядка.

# **ВИСНОВОК**

У результаті виконання роботи було досліджено алгоритми та моделі, що дозволяють автоматично генерувати текст на основі заданого ключового слова, та створено програмне забезпечення «Generatext», яке виконує власне генерацію.

Програмний продукт дозволяє користувачу швидко генерувати текстові дані на різноманітні теми, що значно підвищує продуктивність та ефективність роботи в різних сферах діяльності.

Головною функцією додатку «Generatext» є отримання від користувача інформації щодо файлу, що буде використаний для генерації, та ключового слова, і виведення комп’ютером згенерованого речення.

Мета і завдання курсової роботи були виконані.

Під час виконання роботи було використано мову програмування C# через її об’єктно-орієнтованість та високу продуктивність з використанням платформи .NET Framework. Вибір середовища розробки впав на Microsoft Visual Studio через наявність великого функціоналу та зручність використання.

Для роботи програми було вирішено обрати одну з найпопулярніших моделей генерації тексту: використання n-грам. Вони дозволяють моделювати залежності між словами в тексті та генерувати більш природні речення.

Використовуючи один з принципів об’єктно-орієнтованого програмування – інкапсуляцію, написання коду було поділено на 5 класів, кожен з яких ще на декілька методів. Це дозволило приховати реалізацію деталей внутрішнього функціонування об'єкту від користувача, зберігаючи при цьому його стан та можливість взаємодії з ним через визначені методи.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Gabbrielli, Maurizio. Programming languages principles and paradigms. London, New York: Springer, 2010.
2. Microsoft. Language-Integrated Query (LINQ) (документація). Microsoft Developer Network. 2021. URL: <https://docs.microsoft.com/uk-ua/dotnet/csharp/linq/>
3. Mellish, C.. The History of Natural Language Generation: An Introduction. In Natural Language Generation in Interactive Systems. Springer, Dordrecht. 2005.
4. SHRDLU: A Robot-Based Language-Game for a Child’s World, Terry Winograd (1972)
5. Hausser, H., Martin, J., & Pao, Y. H. A probabilistic approach to natural language generation. Artificial Intelligence. 1984.
6. Manning, C. D., & Schütze, H. Foundations of Statistical Natural Language Processing. Cambridge, MA: MIT Press. 1999.
7. Jurafsky, D., & Martin, J. H. Speech and Language Processing. Pearson. 2019.
8. Taylor, P. How to Build a Speech Recognition Application: A Guide to the Best Tools and Software. 2019. URL: <https://builtin.com/artificial-intelligence/speech-recognition>