ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти магістр

на тему: **«Моделювання та програмна реалізація генеративних мовних алгоритмів»**

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи та
технології спеціальності
126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_21
Пилипенко В.О.
Керівник: Флегантов Л. О.
Рецензент: Петраш Р. В.

Полтава – 2023 року

ВСТУП

Актуальність теми роботи зумовлена тим, що генеративні мовні алгоритми є передовим напрямком в області штучного інтелекту та машинного навчання, що відіграє ключову роль у розвитку технологій обробки природної мови, які заходять застосування у багатьох сферах, включаючи автоматичний переклад, створення чат-ботів, виробництво контенту, освіту, у медичній сфері тощо. Розвиток генеративних мовних моделей сприяє створенню ефективних, автоматизованих та інтерактивних систем, здатних вести природні діалоги з користувачами, автоматизувати створення унікального контенту, підвищує ефективність роботи з великими обсягами даних, включаючи їх обробку, аналіз та генерацію змістовного виводу. Однак, незважаючи на значний прогрес, існують численні виклики та проблеми в області генеративних мовних алгоритмів, які потребують подальших досліджень для їх вирішення.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0117U003099.

Мета дослідження: розробка та впровадження алгоритмів штучного інтелекту для генерування природної мови.

Завдання дослідження:

- аналіз існуючих методів генерації природної мови;
- розробка алгоритму для моделювання мовних структур;
- програмна реалізація розробленого алгоритму;
- тестування та оцінка ефективності алгоритму.

Об'єкт дослідження: генеративні мовні алгоритми у сфері штучного інтелекту, що використовуються для створення природної мови.

Предмет дослідження: методи моделювання та програмна реалізація генеративних мовних алгоритмів, аспекти їх розробки, функціонування, оптимізації та застосування в різних контекстах.

Методи наукових досліджень: вивчення наукових публікацій, статей, та інших джерел для збору інформації про існуючі методи та підходи в галузі генеративних мовних алгоритмів. Аналіз існуючих алгоритмів для визначення їхніх переваг, недоліків, та потенційних полів для вдосконалення. Використання програмного забезпечення для створення моделей мовних алгоритмів та їх тестування в контрольованих умовах. Розробка та тестування нових алгоритмів або модифікацій існуючих, з метою оцінки їх ефективності та придатності. Використання статистичних методів для обробки та інтерпретації отриманих даних, щоб визначити ефективність та надійність алгоритмів.

Інформаційна база – книги, навчальні посібники, наукові публікації та статті в наукових журналах з питань AI, ML та обробки природної мови; дослідження, опубліковані в рамках конференцій NeurIPS, ICML, ACL та інших, у т.ч., які розглядають алгоритми та методи генеративних моделей; технічні звіти та документація провідних компаній, що займаються розробкою мовних моделей, зокрема, OpenAI (GPT серії), Google (BERT, T5), та інших; аналітичні статті та блоги експертів у галузі AI, ML та обробки природної мови; онлайн-курси та відео-лекції на тему AI, ML та ГММ; дослідницькі бази даних та архіви, як PubMed, IEEE Xplore, arXiv для пошуку релевантних досліджень, дискусії на професійних форумах і спільнотах, як Stack Overflow, GitHub, Reddit у секціях, присвячених штучному інтелекту і обробці мови.

Елементи наукової новизни: розробка алгоритмів для специфічних застосувань; розробка способів включення генеративних мовних алгоритмів у ширший контекст технологічних рішень; експериментальне дослідження впливу різних факторів на ефективність алгоритмів.

Практична значущість. Результати роботи можуть бути використані для: створення більш ефективних та природніх систем штучного інтелекту, наприклад,

для обслуговування клієнтів у різних галузях; автоматизації процесів створення текстового контенту для медіа, реклами, освіти тощо.

Апробація результатів роботи здійснена в рамках Науково-практичної конференції за підсумками проходження виробничих практик здобувачів вищої освіти спеціальності 126 Інформаційні системи та технології, кафедра інформаційних систем та технологій Полтавського державного аграрного університету, 17 вересня 2023 р. Вип. VII (частина І). Полтава: ПДАУ, 57 с., Щорічної студентської наукової конференції Полтавського державного аграрного університету, 10 листопада 2022 р. Полтава: ПДАУ, 2022. 263 с. С. 176-177.

Структура випускної кваліфікаційної роботи логічно пов'язана із завданнями дослідження і містить вступ, три розділи основної частини, висновки, список використаних джерел, додаток. Загальний обсяг текстової частини роботи складає 99 сторінок формату А4. Робота містить 27 рисунків і 7 таблиць. Список використаних джерел налічує 68 найменувань.

РОЗДІЛ 1
РОЗВИТОК І ЗАСТОСУВАННЯ ГЕНЕРАТИВНИХ МОВНИХ МОДЕЛЕЙ

1.1 Генеративні мовні моделі: поняття та класифікація

Генеративна мовна модель (ГММ) (Generative Language Model, GML) – це тип комп'ютерної моделі, яка призначена для генерації тексту чи мовленнєвого вмісту. Основна ідея ГММ полягає в тому, щоб модель навчалася розуміти структуру мови та вміла створювати новий текст, схожий на той, на якому вона навчалась під час тренування.

ГММ є елементом в галузі глибокого навчання у контексті обробки природної мови (Natural Language Processing, NLP). Головною функцією ГММ є здатність автоматично створювати новий текст, який має схожу структуру та семантику з вихідними навчальними даними. Це може включати генерацію речень, параграфів, або навіть цілих текстів з урахуванням контексту. Їхнє використання охоплює широкий спектр завдань, від автоматичного завершення тексту до створення текстових описів та навіть генерації творчих та креативних висловлювань. Тобто ГММ здатні навчатися не лише створювати послідовності слів, але й «розуміти» їхній семантичний зміст та взаємозв’язок між ними, що вимагає врахування контексту, історії та відносин між словами.

ГММ застосовуються в різноманітних завданнях обробки природної мови, таких як машинний переклад, автоматичне завершення тексту, створення синтетичних даних для тренування інших моделей, генерація описів зображень тощо. Окремі ГММ, такі як GPT (Generative Pre-trained Transformer), демонструють здатність до вивчення широкого спектру знань з великих корпусів тексту та використання цих знань для генерації нового контенту. Це може бути використано для розв’язання завдань, що вимагають розуміння та генерації тексту в різних областях.

ГММ також можуть бути використані для дослідження лінгвістичної та культурної варіативності. Тобто, вони можуть адаптуватися до різних стилів мовлення, діалектів та елементів культури, що робить їх корисними для різноманітних мовленнєвих задач.

ГММ відкривають можливості для розвитку та вдосконалення систем обробки природної мови та взаємодії між людьми та машинами. Вони не лише допомагають у завданнях автоматичної генерації тексту, але й сприяють розвитку розуміння мови та контексту, що є важливим в аспекті створення інтелектуальних систем.

Основними характеристиками ГММ є: стратегія генерації, вивчення контексту, використання глибокого навчання.

ГММ використовують різні стратегії для генерації тексту. Деякі можуть використовувати ймовірнісний підхід, де ймовірність кожного слова залежить від його контексту та попередніх слів.

Вивчення контексту: ефективні ГМ здатні розуміти широкий контекст введеного тексту. Це включає аналіз контексту попередніх слів, речень чи навіть параграфів для правильної генерації нового тексту.

Використання глибокого навчання: сучасні ГММ базуються на глибокому навчанні. Нейронні мережі, зокрема рекурентна нейронна мережа (RNN), довга короткочасна пам'ять (LSTM), чи Transformer, є популярними архітектурами для генерації тексту.

Проблема навчання: під час навчання ГММ використовує навчальні дані, щоб оптимізувати свої параметри та вивчити відповідність між вхідними та вихідними даними. Це включає великі корпуси тексту для різноманітності та кращої узагальнюючої здатності моделі.

ГММ знайшли застосування в різних галузях, включаючи машинний переклад, автоматичне завершення тексту, створення синтетичних даних для навчання інших моделей, генерацію описів для зображень та інше.

У таблиці 1.1 представлені ключові аспекти ГММ, що є важливими для розуміння їх можливостей та застосувань.

Таблиця 1.1 – Ключові аспекти ГММ

Характеристика	Опис
Використання статистичного навчання	Моделі базуються на статистичних методах для аналізу та генерації мовних даних.
Способність до генерації тексту	Здатність створювати тексти, що звучать природно, включаючи відтворення відповідей у діалогах, створення абзаців тощо.
Контекстна залежність	Моделі враховують контекст, в якому використовуються слова, змінюючи свої відповіді залежно від нього.
Глибоке навчання та нейронні мережі	Використання глибоких нейронних мереж, зокрема трансформерів, для ефективної обробки даних.
Самонавчання	Техніки самонавчання дозволяють моделям постійно вдосконалюватися, аналізуючи нові тексти.
Мультилінгвальність	Здатність обробляти і генерувати текст на кількох мовах.
Застосування в різних областях	Широкий спектр застосувань, включаючи автоматичний переклад, чат-боти, системи відповідей на запитання тощо.

Класифікація ГММ включає:

1. Марківські моделі:

N-грамні моделі: генерують текст, виходячи з ймовірностей появи N-грам у навчальних даних. Зазвичай використовуються для простих завдань генерації тексту.

2. Моделі, основані на правилах:

Граматичні моделі: використовують граматичні правила мови для генерації тексту, забезпечуючи лінгвістичну коректність.

3. Моделі на основі машинного навчання:

Ймовірнісні моделі: орієнтовані на прогнозування ймовірностей слів чи токенів в залежності від контексту. Наприклад, модель мови на основі рекурентних нейронних мереж (RNN) або довготривалих короткочасних пам'ятей (LSTM).

Згорткові моделі: використовують згорткові шари для виявлення локальних залежностей у тексті. Можуть бути використані для генерації тексту на основі контексту.

Transformer: забезпечують механізми уваги для взаємодії між різними частинами вхідного тексту. Моделі, такі як GPT (Generative Pre-trained Transformer), є прикладами цього підходу.

4. Моделі на основі глибокого навчання:

Автокодувальні моделі: спробують вивчити представлення вхідних даних і використовувати це представлення для генерації нового тексту. Наприклад, варіаційні автокодувальні мережі (VAE).

5. Поглиблене навчання:

Моделі на основі підсиленого навчання: використовують механізми підсиленого навчання для покращення якості генерованого тексту, нагороджуючи модель за правильні, зрозумілі або корисні вирази.

У таблиці 1.2. представлені різні стратегії генерації тексту в ГММ та їхні особливості, які впливають на вибір певної стратегії для конкретного застосування.

Таблиця 1.2 – Стратегії генерації тексту в ГММ

Стратегія	Опис	Особливості	Приклад використання
Генерація на основі правил	Тексти генеруються на основі заздалегідь визначених мовних правил.	Висока структурованість і послідовність, обмежена гнучкість.	Юридичні документи, формальні звіти.
Статистичні моделі	Генерація тексту на основі статистичних моделей, враховуючи частоту слова або фрази.	Залежність від величини та якості навчальних даних, базове розуміння мови.	Традиційні системи обробки мови, базові чат-боти.
Моделі на основі нейронних мереж	Використовують глибоке навчання та нейронні мережі для генерації тексту.	Здатність вловлювати складні контексти та нюанси мови, потребують значних обчислювальних ресурсів.	Покращені чат-боти, системи автоматизованої відповіді.
Трансформерні моделі	Використовують архітектуру трансформерів для ефективної генерації тексту.	Висока точність, здатність обробляти довгі послідовності тексту.	Передові чат-боти, автоматичний переклад.

Продовження таблиці 1.2.			
Моделі послідовного передбачення	Генерують текст, передбачаючи наступне слово або символ на основі попередніх.	Підходять для інтерактивних сценаріїв, можуть виникати проблеми з довгостроковим контекстом.	Інтерактивні системи, такі як чат-боти.
Генерація з використанням умовиводу	Використовують логічні умовиводи для створення тексту, що відповідає певним критеріям або умовам.	Потребують складних алгоритмів для визначення та дотримання умов.	Специфічні задачі, де потрібно генерувати текст за умовами.
Генерація з використанням машинного перекладу	Генерують текст шляхом перекладу з однієї мови на іншу.	Забезпечує мультимовну підтримку, може виникати проблема з точністю перекладу.	Створення мультимовного контенту.

Прикладами ГММ є: GPT, BART, N – gram, ELMo, BERT, LSTM, T5, XLNet.

У таблиці 1.3 представлені приклади ГММ для різних стратегій генерації тексту.

Таблиця 1.3 – Приклади ГММ для різних стратегій генерації тексту

Стратегія генерації	Приклади моделей	Описання моделі
Генерація на основі правил	ELIZA	Одна з перших програм, яка імітувала розмову з людиною. Використовує прості правила для генерування відповідей.
Статистичні моделі	n-gram моделі	Прості моделі, які генерують текст на основі статистичної ймовірності послідовності слів.
Моделі на основі нейронних мереж	LSTM мережі	Використовуються для задач, де важливий контекст і послідовність, наприклад у перекладі тексту та чат-ботах.
Трансформерні моделі	GPT-3 від OpenAI	Передова трансформерна модель, здатна генерувати змістовний та високоякісний текст.
Моделі послідовного передбачення	Seq2Seq моделі	Часто використовуються в машинному перекладі та чат-ботах, генерують текст на основі заданого запиту.
Генерація з використанням умовиводу	Системи експертних оцінок	Використовують логічні правила та умовиводи для генерації специфічних відповідей в областях, де потрібні точність та специфічні знання.

Генерація з використанням машинного перекладу	Google Translate	Використовує комплексні алгоритми машинного перекладу для генерації тексту на різних мовах.

Таблиця 1.3 демонструє різноманітність підходів до генерації тексту у ГММ, а також дає приклади конкретних моделей, які використовують ці стратегії.

1.2 Попередні дослідження в галузі генеративних мовних алгоритмів

Ранні дослідження в галузі генеративних мовних алгоритмів (ГМА) визначалися ключовими аспектами, що стали фундаментом для подальшого розвитку та вдосконалення.

Початкові дослідження використовували засновані на правилах та статистичні підходи для обробки природної мови. Метою було розробити алгоритми, які здатні адекватно розпізнавати та генерувати текст на основі лінгвістичних правил та частоти вживання слів [1].

Однією з основних цілей ранніх досліджень було розвиток алгоритмів, які можуть розуміти граматичні структури мови. Моделі спрямовувалися на розпізнавання та генерацію речень, зокрема, з урахуванням синтаксичних та семантичних аспектів [2].

В контексті ранніх досліджень велика увага була приділена завданням машинного перекладу. Дослідження спрямовувалися на створення алгоритмів, які забезпечували точний та зрозумілий переклад між мовами [3].

Певна кількість досліджень була спрямована на використання ГМА для генерації творчого контенту, такого як поезія, літературні твори, чи музичні тексти. Це відкривало нові можливості для застосування генеративних алгоритмів в творчому процесі.

Розуміння семантичних зв'язків у тексті було важливим напрямком ранніх досліджень. Розробка алгоритмів, які можуть адекватно обробляти та розуміти смислові аспекти тексту, виявилася ключовою для розвитку ГМА.

З появою нейронних мереж, особливо рекурентних та згорткових, ранні дослідження почали експериментувати з цими архітектурами для покращення здатності моделей в розумінні та генерації тексту.

Початкові дослідження визначали завдання з розуміння та генерації природної мови. Розробка алгоритмів, що можуть адекватно розуміти та виробляти текст, була ключовою метою. З розвитком ГМА виникала потреба в покращенні систем машинного перекладу. Вчені вдосконалювали алгоритми для ефективного перекладу текстів між мовами. Ці дослідження зосереджувалися на створенні творчих текстів, включаючи поезію, літературні твори та музичні тексти [4]. Це відкривало можливості для використання ГМА в творчому процесі. Дослідження спрямовувалися на вирішення проблем взаєморозуміння між комп'ютерами та людьми, особливо у випадках, коли необхідна контекстуальна та семантична обробка тексту. За появою глибокого навчання дослідники активно вдосконалювали генеративні моделі, використовуючи трансформаційні архітектури та інші передові техніки.

Приховані моделі Маркова (HMM) і моделі суміші Гауса (GMM) були розроблені ще в 1950-х роках [5]. Ці моделі генерували послідовні дані, такі як мова та часові ряди. Однак ГМ отримали значні покращення продуктивності лише після появи глибокого навчання.

Прихована модель Маркова – статистична модель, вперше запропонована Baum L.E. (Baum and Petrie, 1966) [6], яка використовує процес Маркова, який містить приховані та невідомі параметри. У цій моделі спостережувані параметри використовуються для ідентифікації прихованих параметрів. Потім ці параметри використовуються для подальшого аналізу.

Модель суміші Гауса – це ймовірнісна модель, яка передбачає, що всі точки даних створено із суміші кінцевої кількості розподілів Гауса з невідомими параметрами.

З розвитком обробки природної мови (NLP) дослідники розробили підходи до обробки мови, засновані на правилах. Ці методи спиралися на створені вручну правила, призначені для аналізу та розуміння структури мови. Хоча системи на

основі правил мали успіхи, вони також зіткнулися зі значними обмеженнями з точки зору масштабованості та надійності.

Приклад системи, заснованої на правилах, був експеримент Georgetown-IBM, який проводився в 1954 році. В експерименті брала участь машина, запрограмована на переклад речень з російської на англійську. Машина використовувала набір ручних правил для аналізу структури речень і перекладу їх англійською мовою. Незважаючи на те, що експеримент став важливою віхою в галузі, машина була обмежена в здатності виконувати складні завдання обробки мови.

У 1960-х роках дослідники продовжували розробляти засновані на правилах системи для NLP. Однією з перших систем обробки природної мови була ELIZA, розроблена Joseph Weisenbaum, комп'ютерним науковцем з MIT. ELIZA – це чат-бот на основі правил, розроблений для імітації розмови з лікарем-терапевтом. Програма використовувала зіставлення шаблонів і розпізнавання ключових слів для створення відповідей на введення користувача. ELIZA була обмежена у своїй здатності розуміти мову, але це був прорив у цій галузі та викликав інтерес серед дослідників та широкої громадськості [7].

Ранньою системою, заснованою на правилах, була SHRDLU, яку розробив Terry Vinograd, комп'ютерний вчений з MIT. SHRDLU була програмою, яка могла розуміти та реагувати на команди природної мови у віртуальному світі. Програма використовувала набір власноруч створених правил для аналізу структури речень і виконання дій у віртуальному світі. Хоча SHRDLU був обмежений у своїй здатності обробляти складні завдання обробки мови, він продемонстрував потенціал систем на основі правил для NLP.

Системи на основі правил зіткнулися зі значними обмеженнями щодо масштабованості та надійності. Створені вручну правила було складним і трудомістким для створення, і вони часто не вдавалися до складності та мінливості природної мови. У результаті дослідники почали досліджувати нові підходи до NLP, включаючи статистичні моделі та методи машинного навчання.

Незважаючи на свої обмеження, системи, засновані на правилах, зробили значний внесок у сферу NLP. Вони продемонстрували потенціал комп'ютерів для

розуміння та обробки природної мови та проклали шлях для розробки складніших мовних моделей.

У 1990-х роках дослідники почали досліджувати статистичні моделі обробки природної мови (NLP). Статистичні моделі покладаються на великі обсяги даних, щоб вивчати шаблони в мові та робити прогнози щодо нових даних. Ці моделі являли собою значний відхід від систем, заснованих на правилах, які спиралися на створені вручну правила для аналізу та розуміння мови. Поява статистичних моделей революціонізувала сферу NLP і проклала шлях для розвитку сучасних мовних моделей [8].

Статистична модель для NLP прихована модель Маркова (HMM), розроблена Leonard E. Baum, Peter H. Petrie, and George E.P. Box у 1960-х роках. HMM використовуються для моделювання послідовностей подій, які приховані або невідомі, наприклад, частини мови в реченні. Пізніше HMM були застосовані для моделювання мови, де вони використовуються для прогнозування ймовірності слова з урахуванням попередніх слів у реченні.

N-gram – це послідовність із n слів, а мовна модель N-gram передбачає ймовірність слова за попередніми n-1 словами. Модель N-gram є простою, але ефективною статистичною моделлю, яка використовувалася в багатьох завданнях NLP, включаючи розпізнавання мовлення та машинний переклад.

У 1980-х і 90-х роках стали з'являтися статистичні моделі як новий перспективний підхід до NLP. Одним із ключових розробників у цій галузі була Karen Spark Jones, яка представила концепцію зворотної частоти документа (IDF) для вимірювання релевантності термінів у документі. Ця ідея лягла в основу сучасних пошукових систем, які значною мірою покладаються на статистичні моделі для надання відповідних результатів.

Наприкінці 1990-х років дослідники почали досліджувати методи машинного навчання для NLP, включаючи машини підтримки векторів (SVM) і нейронні мережі. SVM (Support Vector Machines) – це тип алгоритму, який можна використовувати для завдань класифікації, наприклад для позначення частини мови або розпізнавання іменованих об’єктів. Нейронні мережі, з іншого боку, є

типом штучної нейронної мережі, яка може вивчати складні закономірності в даних. Нейронні мережі використовувалися в багатьох завданнях NLP, включаючи машинний переклад і класифікацію тексту.

Більшість ранніх мобільних телефонів GSM не підтримували можливість надсилання текстових повідомлень. Першими SMS-шлюзами для мобільних телефонів були мережеві сповіщення, зазвичай для інформування про повідомлення голосової пошти та сповіщення про рахунки. Nokia була першим виробником телефонів, чия загальна телефонна лінія GSM у 1993 році підтримувала надсилання користувачами текстових повідомлень. У 1997 році компанія стала першим виробником, який випустив мобільний телефон із повною клавіатурою: Nokia 9000i Communicator (рис.1.1).

У 1995 році середній американський користувач надсилав 0,4 текстових повідомлення на місяць. Поступово телефони та мережі адаптувалися до кращого сприйняття SMS. У 1999 році нарешті можна було обмінюватися текстами між різними мережами, що підвищило його корисність. До 2000 року середня кількість текстових повідомлень, надісланих у США, зросла до 35 на місяць на людину.

Поширений спосіб комерційного надсилання текстових повідомлень називається «мультитап», кожна цифра в телефоні пов'язана з трьома або чотирма буквами. Наприклад, клавіша «3» відображає «D», «E» і «F». Мультитап простий, але не дуже ефективний. У 1990-х роках співзасновник Tegic Cliff Kushler винайшов T9, скорочення від «Текст на 9 клавішах». Замість багаторазового натискання технологія інтелектуального введення тексту відображає слова з одного натискання клавіші. Коли T9 знайомиться зі словами та фразами, які зазвичай використовуються текстовим текстом, вони стають відповідними порядку частоти. У 2011 році Cliff Kushler винайшов Swype, текстову функцію для сенсорних екранів, яка дозволяє користувачам перетягувати пальці, щоб з'єднати точки між літерами в слові.

У 2013 році з розробкою Word2vec, мовної моделі на основі нейронної мережі, яка вивчає розподілене представлення слів. Word2vec був розроблений Thomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean у Google [10]. Word2vec

являє собою значний відхід від традиційних методів NLP, які спираються на особливості та правила ручної роботи. Натомість Word2vec вивчає векторне представлення слів, яке фіксує їх значення та зв'язки з іншими словами. Такий підхід призвів до значних покращень у багатьох завданнях NLP, включаючи мовне моделювання та машинний переклад.

Рисунок 1.1 – Зовнішній вигляд Nokia 9000i Communicator [9]

Впровадження механізмів уваги, які дозволяють нейронним мережам зосереджуватися на певних частинах речення чи документа. Механізми уваги були представлені в 2014 році Dmytro Bagdanov, Kyunhyun Cho and Joshua Bengio [11]. Механізми уваги призвели до значних покращень у машинному перекладі, резюмуванні тексту та інших завданнях NLP.

Поява статистичних моделей і методів машинного навчання покращила сферу NLP, дозволивши дослідникам створювати більш точні та масштабовані мовні моделі. Ці моделі призвели до значного вдосконалення багатьох завдань NLP, включаючи машинний переклад, класифікацію тексту та розпізнавання мовлення.

Поява нейронних мереж у 2000-х роках ознаменувала прорив для NLP, уможлививши більш складне моделювання структури мови та більш точні

прогнози. Такі дослідники, як Joshua Bengio, Jeffrey Hinton, and Yann LeCun, зіграли ключову роль у розробці архітектур глибокого навчання, які можна застосувати до завдань NLP [12]. Ця робота проклала шлях для появи великомасштабних попередньо навчених мовних моделей, таких як GPT [13] і BERT [14], які з тих пір стали найсучаснішими в цій галузі.

Наступна таблиця 1.4 узагальнює ключові моменти та напрямки ранніх досліджень у галузі ГММ.

Таблиця 1.4 – Ключові аспекти ранніх досліджень у галузі ГММ

Аспект дослідження	Опис
Ранні підходи до ГМА	Використання правил та статистичних методів для розпізнавання та генерації тексту на основі лінгвістичних правил та частоти вживання слів.
Розвиток граматичних структур	Розробка алгоритмів для розуміння граматичних структур мови, зокрема синтаксичних та семантичних аспектів.
Машинний переклад	Створення алгоритмів для точного та зрозумілого машинного перекладу між мовами.
Генерація творчого контенту	Використання ГМА для створення поезії, літературних творів та музичних текстів.
Початкові моделі	Розробка прихованих моделей Маркова (HMM) та моделей суміші Гауса (GMM) у 1950-х, які генерували послідовні дані.
Прихована модель Маркова	Статистична модель, вперше запропонована Baum L.E. у 1966, яка використовує приховані та невідомі параметри для аналізу.
Системи на основі правил	Розробка систем, таких як ELIZA та SHRDLU, які використовували правила для аналізу структури мови.
Статистичні моделі в NLP	Поява статистичних моделей, які використовують великі обсяги даних для вивчення мовних шаблонів.
Розвиток SMS	Розвиток технологій для надсилання SMS у мобільних телефонах, починаючи з 1990-х.
Word2vec	Розробка мовної моделі на основі нейронної мережі для вивчення розподіленого представлення слів.
Механізми уваги в NLP	Впровадження механізмів уваги для зосередження на певних частинах тексту, значні покращення у машинному перекладі.
Нейронні мережі в NLP	Використання архітектур глибокого навчання для складного моделювання структури мови.
GPT	Поява великомасштабних попередньо навчених мовних моделей, які використовують глибоке навчання для вдосконалення завдань NLP.
BERT	Розвиток передових мовних моделей на основі глибокого навчання, що забезпечують значне вдосконалення багатьох завдань NLP, включаючи машинний переклад, класифікацію тексту та розпізнавання мовлення.

1.3 Сучасні методи моделювання текстів та мови

Сучасні методи моделювання текстів та мови включають в себе різноманітні підходи та алгоритми, які базуються на глибокому навчанні, трансформаційних моделях та інших передових техніках.

Нижче представлені кілька ключових методів, які на сьогоднішній день є популярними в галузі моделювання текстів та мови (рис.2.1).

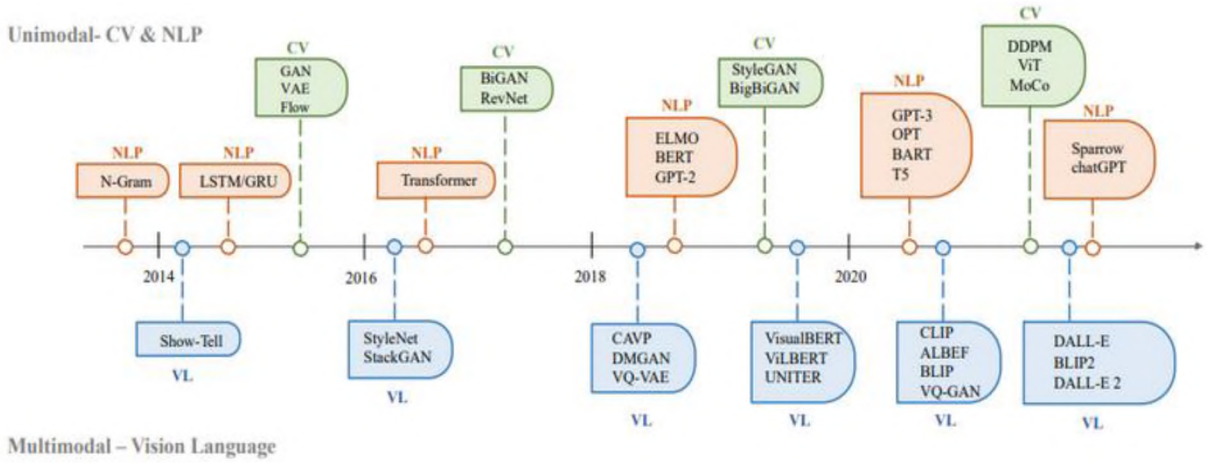

Рисунок 2.1 – Діаграма еволюції ГММ [15]

Модель Transformer стала архітектурою для моделей обробки природної мови. Вони використовують механізм уваги для ефективної обробки довгих послідовностей. Моделі, такі як GPT (Generative Pre-trained Transformer) від OpenAI та BERT (Bidirectional Encoder Representations from Transformers) від Google, є прикладами трансформаторів, які досягають вражаючих результатів у різних завданнях.

Механізм уваги − це компонент архітектур штучних нейронних мереж, який дозволяє моделі приділяти вагу різним частинам вхідного сигналу в залежності від їхнього внеску у вирішення конкретної задачі.

На відміну від останніх моделей представлення мови, BERT розроблено для попереднього навчання глибоких двонаправлених представлень із тексту без міток шляхом спільного обумовлення лівого та правого контексту на всіх рівнях. У

результаті попередньо підготовлену модель BERT можна налаштувати лише за допомогою одного додаткового рівня виводу для створення найсучасніших моделей для широкого кола завдань, таких як відповіді на запитання та мовні висновки, без суттєвих завдань – конкретні модифікації архітектури.

BERT є концептуально простим і емпірично потужним. Він отримує нові найсучасніші результати для одинадцяти завдань обробки природної мови, включаючи підвищення оцінки GLUE до 80,5% (абсолютне покращення на 7,7%), точність MultiNLI до 86,7% (абсолютне покращення на 4,6%), SQuAD v1.1 запитання, що відповідає на тест F1 до 93.2 (абсолютне покращення на 1,5 бала) і тест F1 SQuAD v2.0 на 83.1 (абсолютне покращення на 5,1 бала) [16].

Методи з рекурентними нейронними мережами (RNN). У деяких випадках RNN ще використовуються для моделювання послідовностей тексту, хоча вони не є такими ефективними для довгих послідовностей порівняно з трансформаторами. Використовуються архітектури, такі як LSTM (Long Short-Term Memory) і GRU (Gated Recurrent Unit), для управління проблемами з градієнтом під час тренування [17].

Рекурентна нейронна мережа (RNN) – це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів. Ці алгоритми глибокого навчання зазвичай використовуються для порядкових або тимчасових проблем, таких як мовний переклад, обробка природної мови (NLP), розпізнавання мовлення та підписання зображень; вони включені в популярні програми, такі як Siri, голосовий пошук і Google Translate [18].

LSTM – (Long short-term memory, LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довгострокових залежностей [19].

Gated Recurrent Unit (GRU) − це тип рекурентної нейронної мережі (RNN), який був представлений Cho та ін. у 2014 році як простіша альтернатива мережам довгострокової короткочасної пам’яті (LSTM) [20]. Як і LSTM, GRU може обробляти послідовні дані, такі як текст, мова та дані часових рядів. Основна ідея GRU полягає у використанні механізмів стробування для вибіркового оновлення

прихованого стану мережі на кожному кроці часу. Шлюзові механізми використовуються для керування потоком інформації в мережу та з неї. GRU має два механізми стробування, які називаються воротами скидання та воротами оновлення.

ELMo (Embeddings from Language Models). ELMo використовує контекстні вектори слів, які вивчаються з урахуванням контексту, що оточує слово в реченні. Ця модель здатна враховувати семантичні залежності слів в різних контекстах. Вкладення з мовних моделей, або ELMo, – це тип глибокого контекстуального подання слів, який моделює як складні характеристики використання слів (наприклад, синтаксис та семантика), так і те, як ці варіанти використання різняться залежно від лінгвістичного контексту (тобто модельна полісемія). Вектори слів – це вивчені функції внутрішніх станів глибокої двонаправленої мовної моделі (biLM), яка попередньо навчається на великому текстовому корпусі [21].

ULMFiT (Universal Language Model Fine-tuning). ULMFiT – це метод, який дозволяє доналаштовувати попередньо треновані мовні моделі для специфічних завдань. Це дозволяє використовувати попередньо навчені моделі та адаптувати їх до конкретного корпусу тексту чи [22].

XLNet – це модель, яка поєднує ідеї GPT та BERT, яка використовує трансформену архітектуру та враховує контекст як злітні, так і спадні залежності [23]. XLNet – це новітня і найбільша модель, що з'явилася в сфері обробки природної мови й активно розвивається (Natural Language Processing, NLP). XLNet поєднує сучасні досягнення в NLP та інноваційний підхід до вирішення задачі мовного моделювання. Навчена на величезному корпусі, ця модель досягає визначних результатів у NLP-завданнях бенчмарку GLUE. XLNet є авторегресійною мовною моделлю, яка видає на виході ймовірність одночасної появи послідовності токенів на основі архітектури рекурентного трансформера. Завданням навчання моделі є підрахунок ймовірності для заданого слова (токена), за наявності всіх інших слів у реченні (а не тільки слів зліва або праворуч від заданого) [24].

Transformer з увагою до контексту (C-Transformer). Цей підхід використовує механізм уваги Transformer для моделювання взаємодій між різними частинами тексту та враховує контекстуальні залежності.

GPT-3 та більші мовні моделі. Моделі рівня GPT-3 від OpenAI є одними з найбільших та найпотужніших ГММ, які здатні генерувати текст високої якості та виконувати різноманітні завдання.

GPT-3, або Generative Pre-trained Transformer третього покоління, – це модель машинного навчання нейронної мережі, навчена використовувати дані Інтернету для створення тексту будь-якого типу. GPT-3 розроблений OpenAI, вимагає невеликої кількості вхідного тексту для створення великих обсягів релевантного та складного тексту, створеного машиною [25].

Transformer для задач машинного перекладу. Модель Transformer також здебільшого використовуються в сучасних системах машинного перекладу, таких як T5 (Text-To-Text Transfer Transformer). Кожне завдання, включно з перекладом, відповідями на запитання та класифікацією, передбачає подачу тексту, як вхідних даних моделі, і навчання моделі створенню цільового тексту. Це дозволяє використовувати ту саму модель, функцію втрат, гіперпараметри тощо для нашого різноманітного набору завдань.

Наступна таблиця 1.5 узагальнює основні сучасні методи моделювання текстів та мови, включаючи опис моделей та їхні ключові характеристики.

Таблиця 1.5 – Сучасні методи моделювання текстів та мови

Метод моделювання	Опис
Модель Transformer	Використання механізму уваги для ефективної обробки довгих послідовностей. GPT та BERT є прикладами моделей, що досягають високих результатів у різних завданнях.
Рекурентні нейронні мережі (RNN)	Використовуються для моделювання послідовностей тексту, зокрема за допомогою LSTM та GRU, для управління проблемами з градієнтом під час тренування.
Застосування RNN	RNN використовуються в мовному перекладі, NLP, розпізнаванні мовлення, підписанні зображень; включені в популярні програми, такі як Siri та Google Translate.
LSTM	Рекурентні нейронні мережі, здатні до навчання довгострокових залежностей.

Gated Recurrent Unit (GRU)	Простіша альтернатива LSTM, використовує механізми стробування для оновлення прихованого стану мережі.
Продовження таблиці 1.5	
ELMo	Використовує контекстні вектори слів, вивчені з урахуванням контексту, що оточує слово в реченні.
ULMFiT	Метод для доналаштування попередньо тренованих мовних моделей для специфічних завдань.
XLNet	Поєднує ідеї GPT та BERT, використовує трансформену архітектуру, враховує контекст як зліва, так і справа.
Авторегресійна мовна модель	XLNet – авторегресійна мовна модель, яка видає на виході ймовірність одночасної появи послідовності токенів.
GPT-3	Одна з найбільших та найпотужніших ГММ, здатна генерувати текст високої якості та виконувати різноманітні завдання.
Transformer для машинного перекладу	Використовуються в сучасних системах машинного перекладу, таких як T5. Зміни порівняно з BERT включають додавання причинного декодера до двонапрямленої архітектури, заміну завдання «заповнити порожні місця» сумішшю альтернативних завдань перед навчанням.

Висновки до розділу 1

Проведено детальний аналіз та огляд теоретичних основ ГМА, які визначають важливий фундамент для подальших досліджень. Розділ охоплює поняття, класифікації, огляд попередніх досліджень ГМА та сучасні методи моделювання.

Розглянуто сутність ГММ як важливого напрямку в глибокому навчанні. Розкрито їхню основну функцію – генерацію нового тексту, що має подібну структуру та семантику до навчальних даних. Особливий акцент був зроблений класифікації моделей.

Проведений огляд попередніх досліджень у галузі ГМА. Визначено хронологічну послідовність розвитку та ключових проривів в області ГМА.

Розглянуто сучасні методи моделювання текстів та мови. Окреслений їх основний функціонал.

РОЗДІЛ 2
ТЕОРЕТИЧНІ ОСНОВИ МОДЕЛЮВАННЯ ГЕНЕРАТИВНИХ МОВНИХ АЛГОРИТМІВ

2.1 Огляд теоретичних підходів до моделювання мови

Модель Маркова − це стохастичний метод для випадково мінливих систем, які мають властивість Маркова. Це означає, що в будь-який момент часу наступний стан залежить лише від поточного стану та не залежить від минулого. Два типи моделі Маркова, які зазвичай застосовуються, використовуються, коли представлена система є автономною, тобто коли на систему не впливає зовнішній агент [27].

Ланцюги Маркова. Це найпростіший тип моделі Маркова, який використовується для представлення систем, у яких можна спостерігати всі стани. Ланцюги Маркова показують усі можливі стани, а між станами вони показують швидкість переходу, яка є ймовірністю переходу з одного стану в інший за одиницю часу. Застосування цього типу моделі включає передбачення ринкових крахів, розпізнавання мови та алгоритми пошукових систем.

Приховані марковські моделі. Вони використовуються для представлення систем з деякими неспостережуваними станами. На додаток, до показу станів і швидкості переходу, приховані моделі Маркова також представляють спостереження та ймовірність спостереження для кожного стану. Приховані моделі Маркова використовуються для цілого ряду програм, включаючи термодинаміку, фінанси та розпізнавання образів.

Простою марковською моделлю є марковський ланцюг, який можна виразити рівняннями, матрицею переходу або графіком. Матриця переходу використовується для вказівки ймовірності переходу з кожного стану в інший стан. Поточні стани перераховуються в рядках, а наступні стани представлені у вигляді стовпців. Тоді кожна комірка містить імовірність переходу з поточного стану до

наступного. Для будь-якого заданого рядка всі значення комірок мають у сумі дорівнювати одиниці.

Графік складається з кіл, кожне з яких позначає стан, і стрілок, що вказують на можливі переходи між станами. Стрілки напрямків позначені ймовірністю переходу. Сума ймовірностей переходу стрілок напрямків, що виходять із будь-якого кола, повинна дорівнювати одиниці.

Інші моделі Маркова базуються на ланцюжкових представленнях, але з доданою інформацією, такою як спостереження та ймовірності спостережень.

Матриця переходів нижче представляє перемикання передач в автомобілі з механічною коробкою передач (рис. 2.1). Можливо шість станів, і перехід із будь-якого заданого стану, в будь-який інший залежить лише від поточного стану, тобто на те, куди автомобіль перейде з другої передачі, не впливає те, де вона була до другої передачі. Така матриця переходів може бути побудована на основі емпіричних спостережень, які показують, наприклад, що найбільш ймовірні переходи з першої передачі на другу або нейтральну.

Shifting gears

	First gear	Second gear	Third gear	Fourth gear	Neutral	Reverse
First gear	.005	.5	.05	.04	.4	.005
Second gear	.4	.005	.5	.05	.04	.005
Third gear	.05	.4	.005	.5	.04	.005
Fourth gear	.04	.05	.8	.005	.1	.005
Neutral	.6	.005	.005	.005	.005	.38
Reverse	.1	.005	.005	.005	.88	.005

Рисунок 2.1 – Зовнішній вигляд матриці переходів [28]

Вектори слів (також звані вбудовуваннями слів) – це тип представлення слів, який дозволяє словам зі схожими значеннями мати однакове представлення

(рис.2.2). Це техніка, яка використовується в обробці природної мови (NLP) для представлення слів для аналізу тексту, як правило, як дійсний вектор, який кодує значення слова таким чином, що слова, близькі у векторному просторі, ймовірно, матимуть подібні значення(рис. 2.3).

Алгоритм word2vec використовує модель нейронної мережі для вивчення асоціацій слів із великого корпусу тексту (великого та структурованого набору текстів). Після навчання така модель може виявляти слова-синоніми або пропонувати додаткові слова для часткового речення.

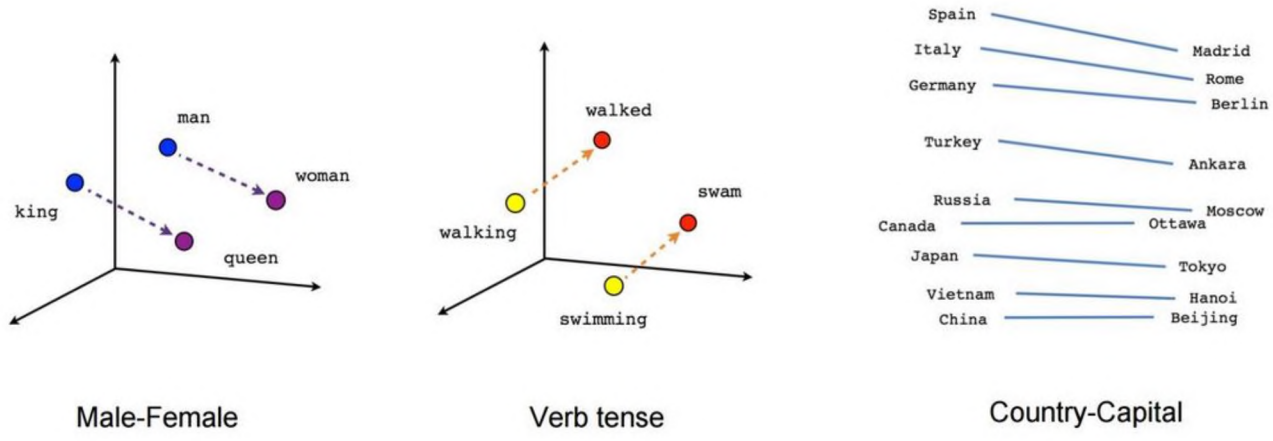

Рисунок 2.2 – Зовнішній вигляд векторів слів [29]

Кожне слово закодовано у вектор (як число, представлене в кількох вимірах), щоб відповідати векторам слів, які з'являються в подібному контексті. Таким чином формується щільний вектор для тексту. Ці векторні моделі відображають семантично схожі фрази на найближчі точки на основі еквівалентності, подібності або спорідненості ідей і мови

Теорія інформації – це галузь прикладної математики, яка займається кількісним визначенням того, скільки інформації міститься в сигналі. Вперше він був винайдений для вивчення надсилання повідомлень із дискретних алфавітів по шумному каналу, такому як радіопередача.Теорія інформації розповідає, як розробити оптимальні коди та обчислити очікувану довжину повідомлень, відібраних із певних розподілів ймовірностей, використовуючи різні схеми кодування ($I(x)=-P(x)$).

У ML можна застосувати цю теорію до безперервних змінних, де деякі інтерпретації довжини повідомлення не застосовуються [30].

Структуровані ймовірнісні моделі [32]. Алгоритми машинного навчання часто включають розподіли ймовірностей за дуже великою кількістю випадкових змінних. Розподіли ймовірностей включають прямі взаємодії між відносно невеликою кількістю змінних.

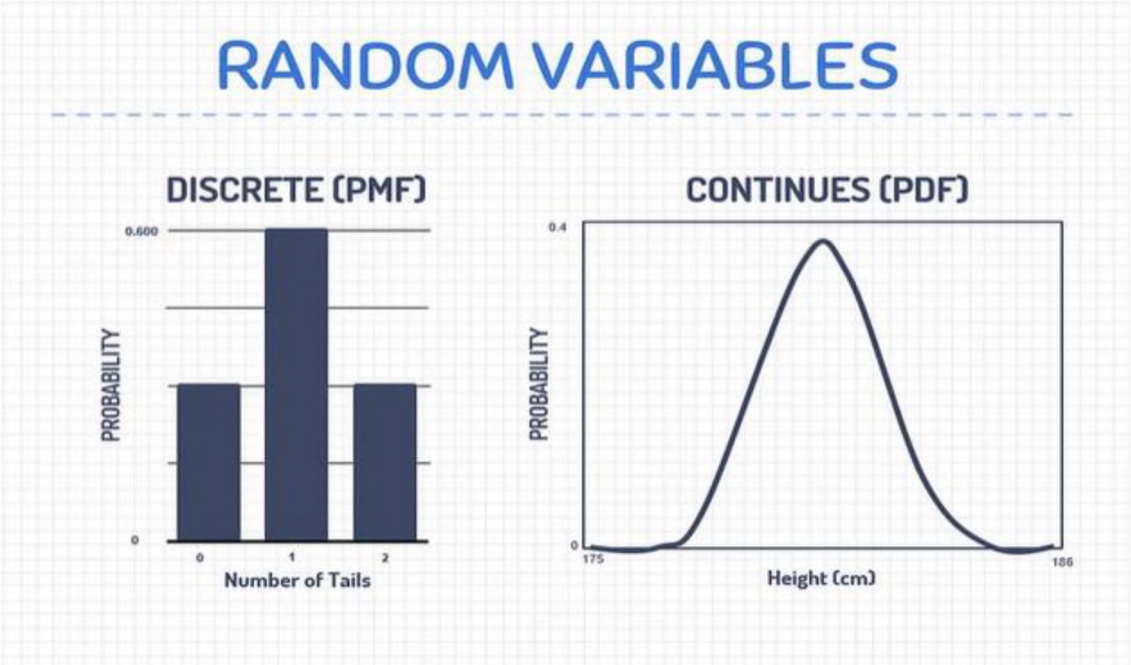

Рисунок 2.3 – Зовнішній вигляд випадкових величин [31]

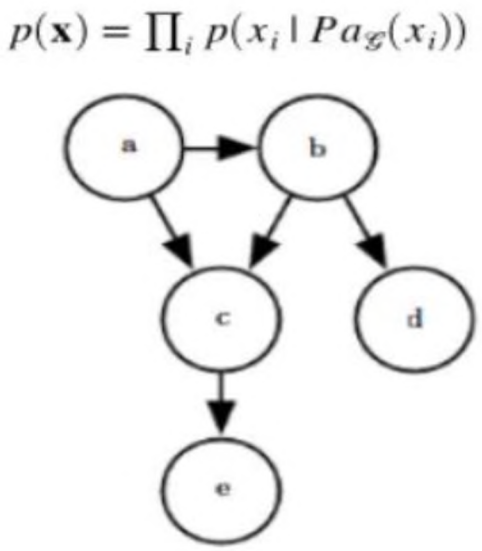

Рисунок 2.4 – Зовнішній вигляд спрямованих моделей [33]

Спрямовані моделі представляють факторизацію в розподілі ймовірностей, де існує потік впливу між вузлами(рис. 2.4).

$$p(\mathbf{x}) = \frac{1}{Z}\prod_i \phi^{(i)}(C^{(i)})$$

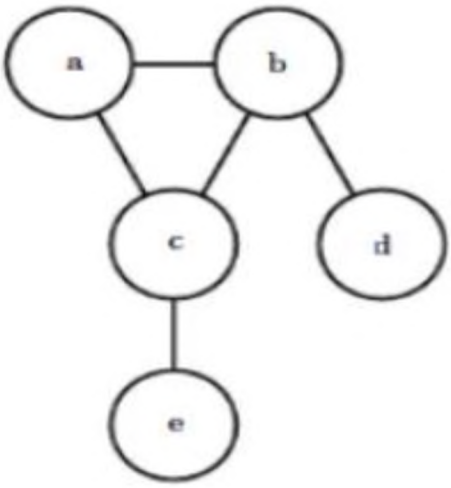

Рисунок 2.5 – Зовнішній вигляд неорієнтованих моделей [34]

Неорієнтовані моделі представляють собою розкладання на множники в набір функцій, які традиційно не розподіляють ймовірності будь-якого типу, тобто не обмежені, що сума фактора повинна дорівнювати 1, але має бути невід'ємною (рис. 2.5).

2.2 Загальна структура генеративних мовних алгоритмів

Загальна структура ГММ включає кілька ключових компонентів (рис.2.6):

– Введення (Input). Текстові дані, зазвичай вхідними даними є текстові послідовності, такі як пропозиції або документи, на яких модель навчатиметься або генеруватиме нові тексти;

– Токенізація (Tokenization). Розбиття тексту на токени, текст розбивається на дрібніші одиниці, такі як слова або підслів (subwords). Це дозволяє моделі працювати з дискретними елементами тексту;

– Вбудовування токенів (Token Embedding). Перетворення токенів на вектори, де кожен токен представляється у вигляді вектора, який зазвичай має набагато меншу розмірність, ніж вихідний простір токенів;

– Багатоголовий Transformer (Multi-Head Transformer). Архітектура Transformer, багатоголовий Transformer складається з кількох шарів Transformer. Кожен шар включає механізми уваги, нормалізацію та повнозв'язкові шари;

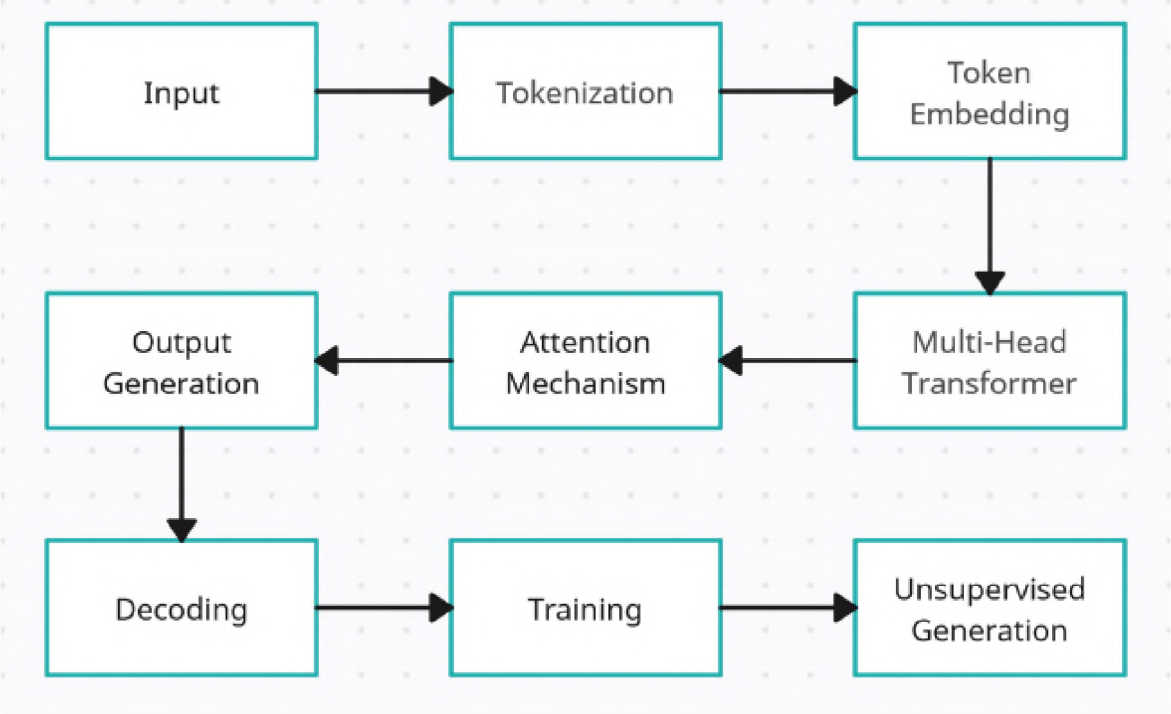

Рисунок 2.6 – Зовнішній вигляд загальної структури ГММ [35]

– Увага (Attention Mechanism). Механізм уваги дозволяє моделі зосереджуватись на різних частинах вхідних даних при генерації виходу. Це особливо важливо для обробки довгих залежностей у тексті;

– Генерація (Output Generation). Пророцтво наступного токена. Модель генерує послідовність токенів покроково. Кожен наступний токен передбачається на основі попередніх токенів та внутрішнього стану моделі.

– Декодування (Decoding). Перетворення на текст, отримані вихідні вектори токенів декодуються назад у текстову форму, щоб отримати фінальний результат генерації;

– Навчання (Training). Навчання з учителем, у процесі навчання модель порівнює свої прогнози з фактичними даними та коригує свої ваги для покращення якості генерації;

– Генерація без вчителя (Unsupervised Generation). Семплювання та декодування. Після навчання модель може бути використана для створення тексту без залучення фактичних відповідей. Семплювання із розподілу ймовірностей допомагає створювати різноманітні тексти.

2.3 Опис популярних алгоритмів та методів генерації текстів

Для завдань генерації тексту найбільш часто використовують RNN. Вони мають повторювану архітектуру, яка дозволяє їм фіксувати послідовні залежності у вхідних даних. Кожен вхідний маркер обробляється прихованим шаром, а вихідні дані повертаються в мережу як вхідні дані для наступного маркера. RNN можуть генерувати текст символ за символом або слово за словом. Вони відомі своєю здатністю обробляти вхідні дані змінної довжини та здатністю отримувати контекстну інформацію. LSTM – це широко використовуваний варіант RNN, який допомагає вирішити проблему зникнення градієнта. RNN показали вражаючі результати у створенні реалістичного та зв'язного тексту [36].

Рекурентна нейронна мережа – це тип штучної нейронної мережі, розробленої для обробки послідовних даних, що робить її придатною для таких завдань, як генерація тексту. На відміну від традиційних нейронних мереж, RNN мають цикл зворотного зв'язку, який дозволяє інформації зберігатися та впливати на майбутні результати. Цей цикл фіксує часову природу послідовних даних, повертаючи прихований стан мережі на кожному кроці. Це дозволяє RNN запам'ятовувати минулу інформацію та генерувати результати на основі контексту [37]. Завдяки своїй здатності моделювати залежності та шаблони в послідовних даних, RNN довели свою ефективність в алгоритмах генерації тексту та таких завданнях, як переклад мови та розпізнавання мовлення. RNN можна застосовувати

для створення тексту, використовуючи попередні слова як контекст для передбачення наступного слова. Цей послідовний підхід робить RNN ефективними для захоплення залежностей і шаблонів у мові, що призводить до цілісного та контекстуально релевантного генерування тексту. Навчаючи модель на великому корпусі тексту, RNN можуть вивчати правила граматики, структуру речень і навіть імітувати стиль вхідних даних. Однак RNN можуть страждати від генерування повторюваного або безглуздого тексту через обмеження довгострокових залежностей. Щоб подолати це, такі методи, як LSTM або GRU, можна використовувати для збереження важливої інформації в довгих послідовностях і покращення якості генерації тексту(рис.2.7).

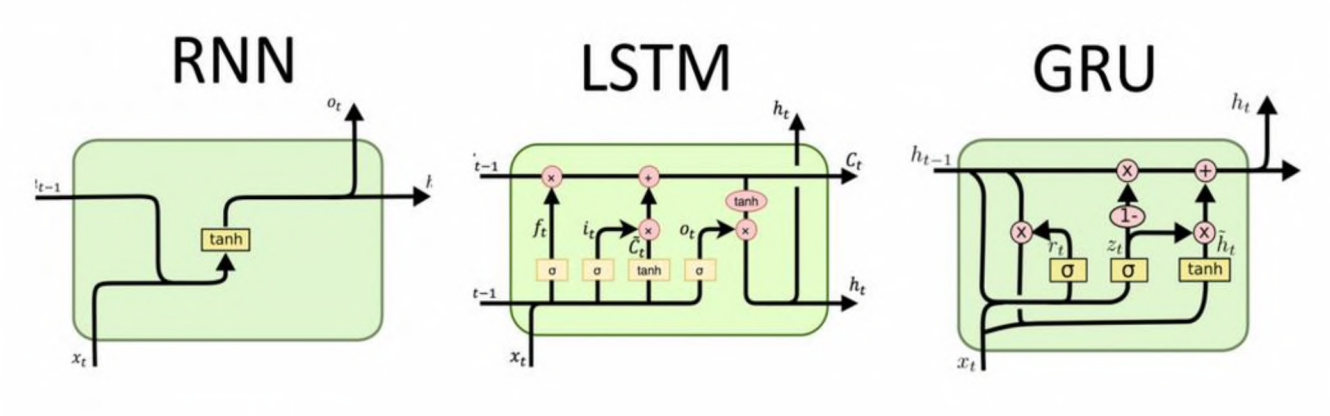

Рисунок 2.7 – Зовнішній вигляд рекурентних нейронних мереж [38]

Генераційні змагальні мережі (GAN; Generative Adversarial Networks) − це потужний тип алгоритму, який використовується для створення тексту. GAN складаються з двох частин: генератора та дискримінатора (рис. 2.8). Генератор генерує нові зразки тексту, тоді як дискримінатор намагається відрізнити реальний текст від згенерованого. Оскільки вони конкурують один з одним, генератор стає кращим у створенні реалістичного тексту, тоді як дискримінатор стає кращим у ідентифікації згенерованого тексту. GAN успішно використовуються для генерування тексту в різних програмах, таких як мовний переклад, поезія та створення діалогів. Вони показали великий потенціал у створенні високоякісних і різноманітних текстових результатів [39].

GAN – це клас алгоритмів, які використовуються для створення тексту. Вони складаються з двох нейронних мереж: генератора та дискримінатора. Генератор навчається створювати нові зразки тексту, тоді як дискримінатор прагне класифікувати, чи є згенерований текст підробленим чи справжнім. Ці дві мережі працюють у тандемі, при цьому генератор покращує свій вихід, обманюючи дискримінатор, а дискримінатор стає кращим у відрізненні справжнього від підробки. GAN показали перспективу в різних завданнях створення тексту, включаючи створення реалістичних діалогів, створення віршів і навіть написання новинних статей. Вони пропонують новий підхід до створення тексту, який дуже нагадує людське письмо.

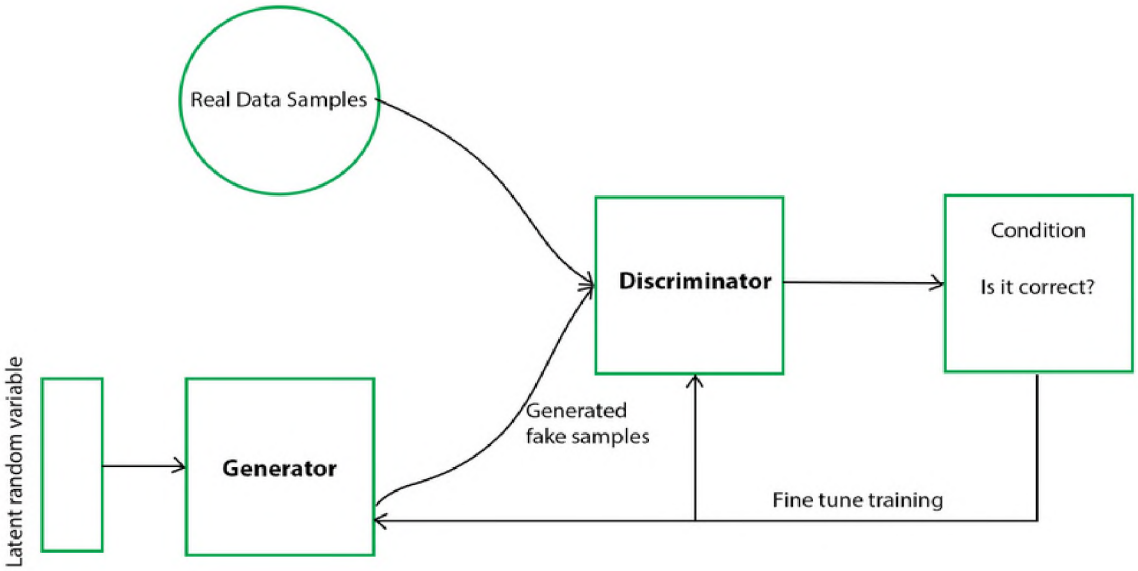

Рисунок 2.8 – Зовнішній вигляд моделі GAN [42]

GAN знайшли застосування у сфері генерації тексту. Навчаючи генераторну мережу створювати зразки тексту та мережу дискримінатора, щоб відрізняти реальний текст від згенерованого тексту, GAN показали перспективу у створенні реалістичного та зв'язного текстового вмісту. Цей підхід успішно використовується в різних областях, включаючи розробку чат-ботів, мовний переклад і навіть творче написання. GAN мають унікальну перевагу в тому, що вони можуть вивчати та імітувати базові шаблони та структури певного текстового

корпусу, що дозволяє генерувати контекстуально точні та значущі результати. Однак такі проблеми, як створення різноманітного та неупередженого вмісту, все ще залишаються, що вимагає подальших досліджень і вдосконалення методів створення тексту на основі GAN [40].

GAN має генератор і дискримінатор. Генератор створює підроблені зразки даних (зображення, аудіо тощо) і намагається обдурити дискримінатор. Дискримінатор, з іншого боку, намагається відрізнити справжні зразки від підроблених. Генератор і дискримінатор є нейронними мережами, і обидва вони конкурують один з одним на етапі навчання. Ці кроки повторюються кілька разів, і після кожного повторення генератор і дискримінатор стають все кращими і кращими [41].

Алгоритм Transformer представив концепцію самоуважності. Він використовує мережу механізмів уваги, щоб зрозуміти зв'язки між словами та створити більш зв'язний текст. На відміну від попередніх алгоритмів, Transformer не покладається на повторювані підключення, що робить його швидшим і ефективнішим. Дозволяючи кожному слову звертати увагу на всі інші слова в реченні, Transformer фіксує довгострокові залежності та створює контекстно насичений текст. Він також відмінно справляється з різними модальностями, що робить його придатним для різноманітних додатків, таких як переклад мови та підписи до зображень. Його надзвичайна продуктивність зробила його наріжним каменем у сучасних завданнях обробки природної мови [43].

Transformer − це потужний алгоритм генерації тексту. Він відомий своїм механізмом привернення уваги та здатністю обробляти довготривалі залежності в реченні. На відміну від традиційних моделей, таких як рекурентні нейронні мережі, Transformer працює за принципом самоуважності, що дозволяє йому зосереджуватися на різних частинах вхідної послідовності. Ця унікальна архітектура широко використовується в різних програмах, включаючи машинний переклад, чат-боти та узагальнення тексту. Завдяки своїй здатності генерувати зв'язний текст, відповідний контексту, Transformer значно просунув сферу алгоритмів генерації тексту.

Модель Transformer стала потужним інструментом для вирішення завдань генерації тексту. У порівнянні з традиційними рекурентними нейронними мережами, механізм уваги Transformer дозволяє моделювати довгострокові залежності. Це забезпечує більш плавний і узгоджений вихід. Transformers відмінно справляються з такими завданнями, як машинний переклад, моделювання мови та створення діалогів(рис. 2.9).

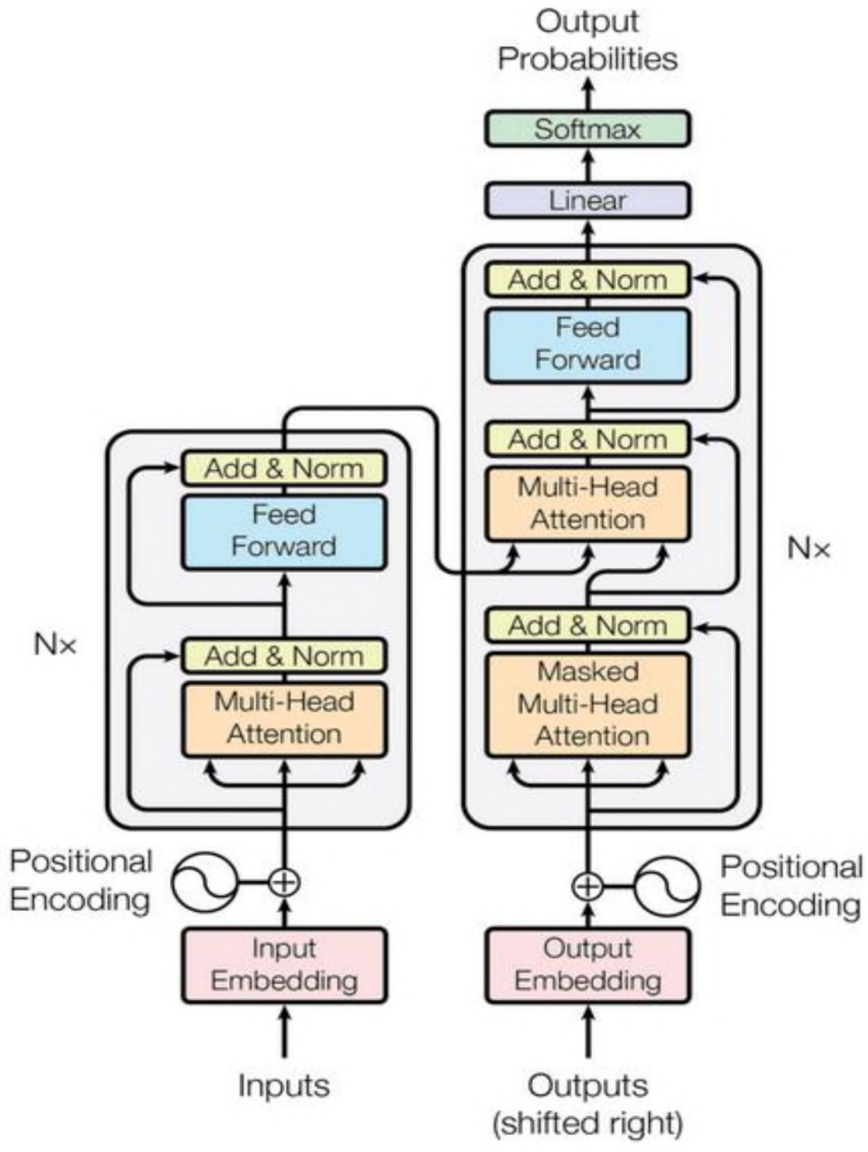

Рисунок 2.9 – Зовнішній вигляд моделі Transformer [44]

Механізм самоконтролю в Transformers допомагає вловлювати кореляції між різними словами у послідовності введення, забезпечуючи ефективне й точне генерування тексту. Використовуючи можливості Transformer, дослідники

продовжують розвивати сферу генерації тексту, розсуваючи межі того, що можливо у генерації природної мови.

Завдання кодера в лівій половині архітектури Transformer полягає в тому, щоб зіставити вхідну послідовність з послідовністю безперервних представлень, які потім подаються в декодер. Декодер у правій половині архітектури отримує вихідні дані кодувальника разом із вихідними даними декодера на попередньому кроці часу для генерації вихідної послідовності (рис. 2.9).

Мережі глибокої віри (DBN) − це тип нейронних мереж, які демонструють потужні можливості у створенні тексту. DBN складаються з кількох шарів прихованих блоків, які кодують складні шаблони в даних. У цих мережах використовується фаза попереднього навчання, коли нижні рівні вчаться реконструювати вхідні дані, фіксуючи важливі характеристики. Потім на етапі тонкого налаштування налаштовуються параметри, щоб максимізувати вірогідність навчальних даних [45]. DBN моделюють залежності в тексті, що робить їх ефективними для таких завдань, як створення правдоподібних речень або створення діалогу. Використовуючи можливості глибокого навчання, DBN роблять значний внесок у вдосконалення алгоритмів генерації тексту.

DBN, або Deep Belief Networks, є типом імовірнісної графічної моделі, яка зазвичай використовується в алгоритмах генерації тексту. Вони складаються з кількох складених шарів прихованих одиниць із зв'язками між сусідніми шарами. DBN довели свою ефективність у захопленні складних шаблонів і залежностей у текстових даних. Під час навчання мережа вчиться виявляти приховані особливості у вхідних даних, що дозволяє генерувати високоякісні зразки тексту [46]. Використовуючи свою глибоку архітектуру, DBN мають здатність виявляти довготривалі залежності та генерують узгоджений і контекстуально релевантний текст. Ці мережі знайшли широке застосування в різних задачах обробки природної мови, включаючи моделювання мови, машинний переклад і синтез тексту.

DBN також знайшли широке застосування в алгоритмах генерації тексту. Моделюючи залежності між словами в тексті, DBN можуть генерувати зв'язні та контекстуально відповідні речення. Це робить їх придатними для таких завдань, як

діалогові системи, машинний переклад і створення мови для чат-ботів. Здатність DBN фіксувати довготривалі залежності в тексті допомагає створювати більш природні та людські речення [47].

DBN можна навчати на великих масивах текстових даних, що дозволяє їм вивчати складні шаблони та створювати високоякісний текст (рис. 2.10). Перший етап полягає в навчанні рівня властивостей, який може безпосередньо отримувати вхідні сигнали від пікселів. В альтернативній субкасті, що вийшла на пенсію, вивчіть риси попередньо досягнутих рис, розглядаючи значення цієї субкасти як пікселі. Нижня межа журнальної відповідальності набору даних для навчання покращується кожного разу, коли до мережі додається нова підкаста посилок або функцій.

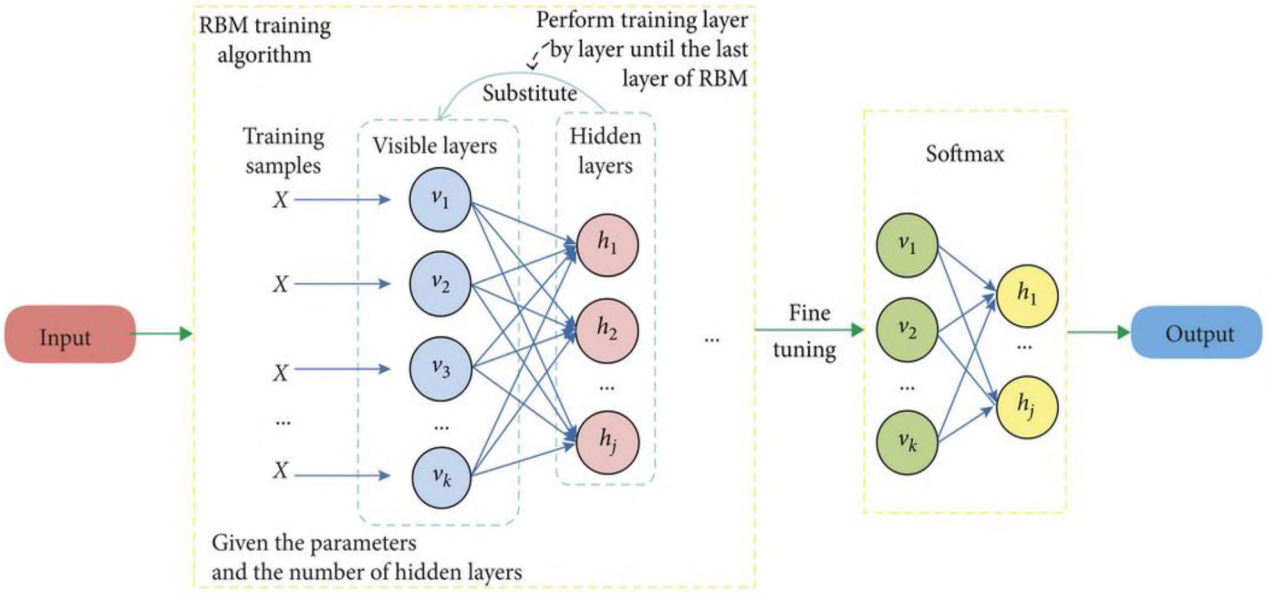

Рисунок 2.10 – Зовнішній вигляд побудови мережі DBN [49]

Операційний конвеєр Deep Belief Network виглядає наступним чином:

– Використовується алгоритм навчання Greedy [48] для попереднього навчання DBN. Для вивчення генеративних ваг зверху вниз – жадібний метод навчання, який використовує пошаровий підхід. Ці генеративні ваги визначають зв'язок між змінними в одному шарі та змінними в шарі вище;

– На двох верхніх прихованих шарах ми виконуємо численні кроки вибірки Gibbs в DBN. Два верхніх прихованих шару визначають RBM, таким чином, цей етап фактично витягує з нього семпл;

– Потім генерується вибірка з видимих одиниць, використовуючи один прохід вихідної вибірки через решту моделі;

– Використовується єдиний перехід знизу вгору, щоб вивести значення латентних змінних у кожному шарі. На нижньому рівні жадібне попереднє навчання починається з спостережуваного вектору даних. Потім він навпаки точно налаштовує генеративні ваги.

Побудова мережі Deep Belief Network потребує навчання кожного рівня RBM. Спочатку ініціюються одиниці та параметри для цієї мети. В алгоритмі Contrastive Divergence є дві фази: позитивна та негативна. Розраховуються двійкові стани прихованих шарів у позитивній фазі шляхом обчислення ймовірностей ваг і видимих одиниць. Вона відома як позитивна фаза, оскільки підвищує ймовірність набору навчальних даних. Негативна фаза зменшує ймовірність створення зразків моделлю. Щоб навчити повну мережу Deep Belief Network, використовується техніку жадібного навчання. Алгоритм жадібного навчання навчає один RBM за раз, доки не будуть навчені всі RBM [50].

2.4 Вибір моделі та обґрунтування методології моделювання

Мовні моделі можна приблизно розділити на кодери (BERT) – вони мають лише блоки кодувальників, декодери (GPT-x, PaLM, OPT) та блоки декодера та кодувальник-декодер (BART, T0/T5) [51].

Архітектури BERT вирішують завдання передбачення слова у реченні, і може побачити всі слова до та після цього замаскованого слова. Це має термін «моделювання замаскованої мови». Архітектура кодувальника зазвичай використовується для завдань моделювання мови, які включають кодування послідовності вхідних токенів і створення представлення фіксованої довжини,

також відомого як контекстний вектор або вбудовування, яке підсумовує вхідні дані. Цей вектор контексту потім може бути використаний наступним завданням, таким як машинний переклад або резюмування тексту [52]. Прикладом є модель BERT.

Для архітектур декодера та кодера-декодера завдання полягає в тому, щоб передбачити наступний маркер або набір маркерів, тобто, якщо задані маркери [0, ..., n-1], то передбачити n. Ці архітектури використовуються для завдань моделювання мови, які включають генерацію послідовності вихідних токенів на основі вектора вхідного контексту (рис. 2.11).

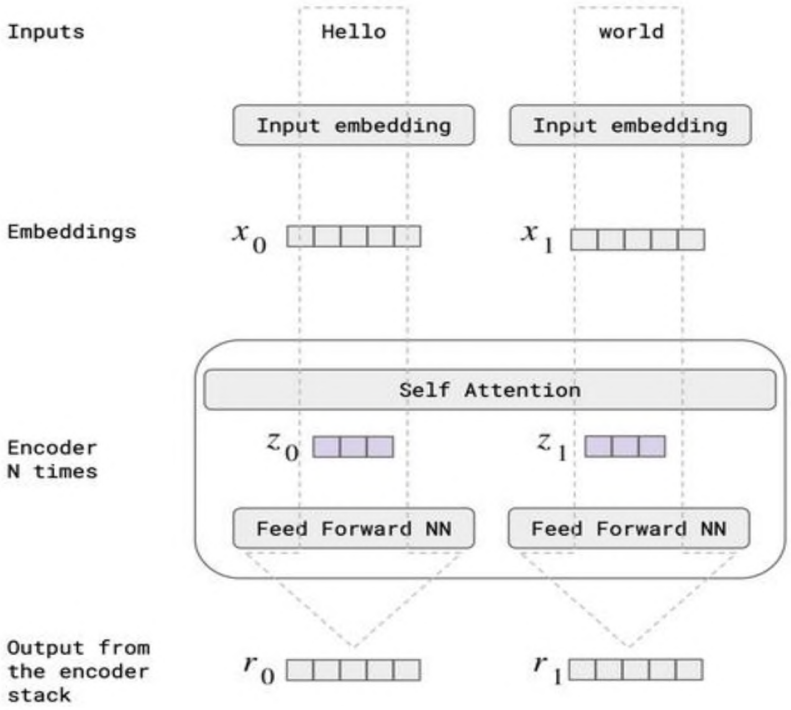

Рисунок 2.11 – Архітектури декодера [54]

Кожен кодер складається з двох рівнів: Self-Attention і Feed Forward Neural Network. Вхідні дані кодера спочатку проходять через рівень Self-Attention. Цей рівень допомагає кодеру переглядати інші слова у вхідному реченні, коли він кодує певне слово. Результати цього рівня передаються на повністю підключений рівень (нейронна мережа прямої подачі) [53]. Коли модель обробляє кожну лексему (кожне слово у вхідній послідовності), рівень Self-Attention дозволяє шукати

підказки в інших токенах у вхідній послідовності, які можуть допомогти покращити кодування цього слова (рис. 2.12).

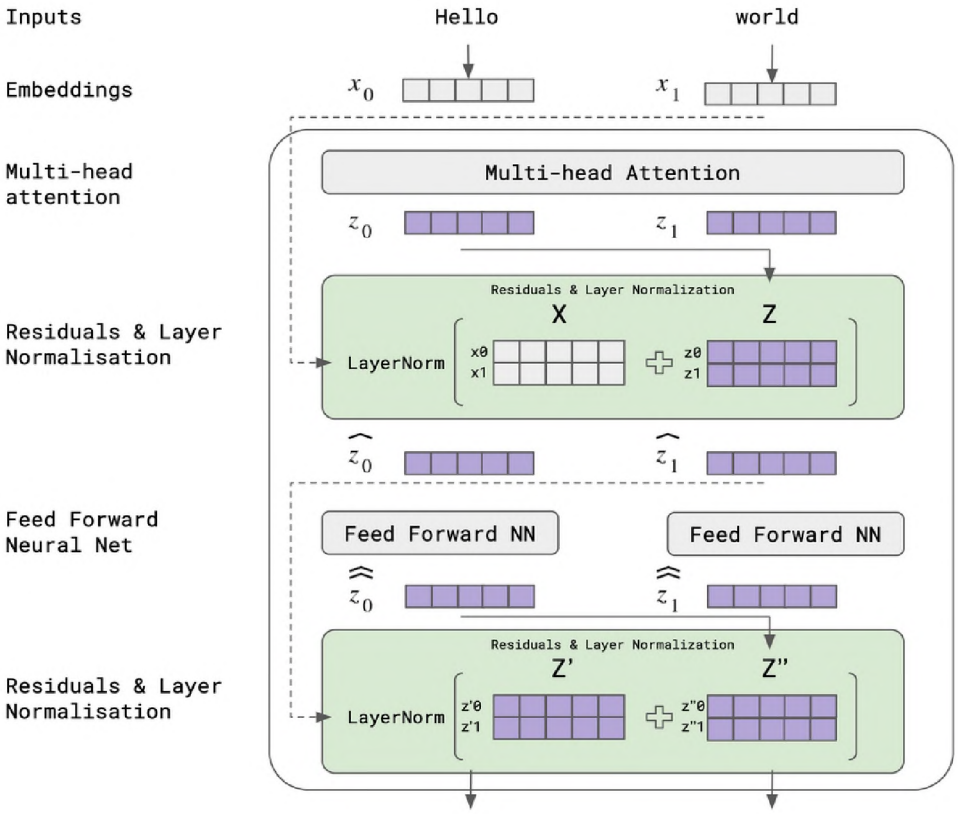

Рисунок 2.12 – Архітектура кодера [55]

На повністю підключеному рівні нейронна мережа прямого зв'язку не взаємодіє з іншими словами, і тому різні ланцюжки можуть виконуватися паралельно, коли вони проходять через цей рівень. Це одна з головних характеристик Transformers, яка дозволяє їм обробляти всі слова у вхідному тексті паралельно.

В архітектурі Transformer залишкові з'єднання є типом скороченого з'єднання, яке дозволяє інформації обходити один або кілька рівнів мережі. Залишкові зв'язки (-->) використовуються навколо кожного з двох підрівнів, після чого відбувається нормалізація шару [56]. Залишкові з'єднання використовуються для вирішення проблеми зникнення градієнтів, яка може статися, коли градієнти мережі стають дуже малими, і мережі стає важко ефективно навчатися. Дозволяючи інформації обходити деякі з шарів, залишкові зв'язки можуть

допомогти підтримувати величину градієнтів, коли інформація проходить через мережу, полегшуючи навчання мережі.

2.5 Розробка теоретичної моделі генеративного алгоритму

Щоб проілюструвати, як працює Transformer, використано, як приклад, завдання перекладу тексту з англійської на іспанську. Модель отримує англійське речення як вхід (після застосування вбудовування) і виводить ту саму послідовність, зважену механізмом уваги. Математично Encoder виконує перетворення у просторі вбудовування англійських токенів, зважуючи вектори відповідно до важливості значення їх у реченні (рис. 2.13).

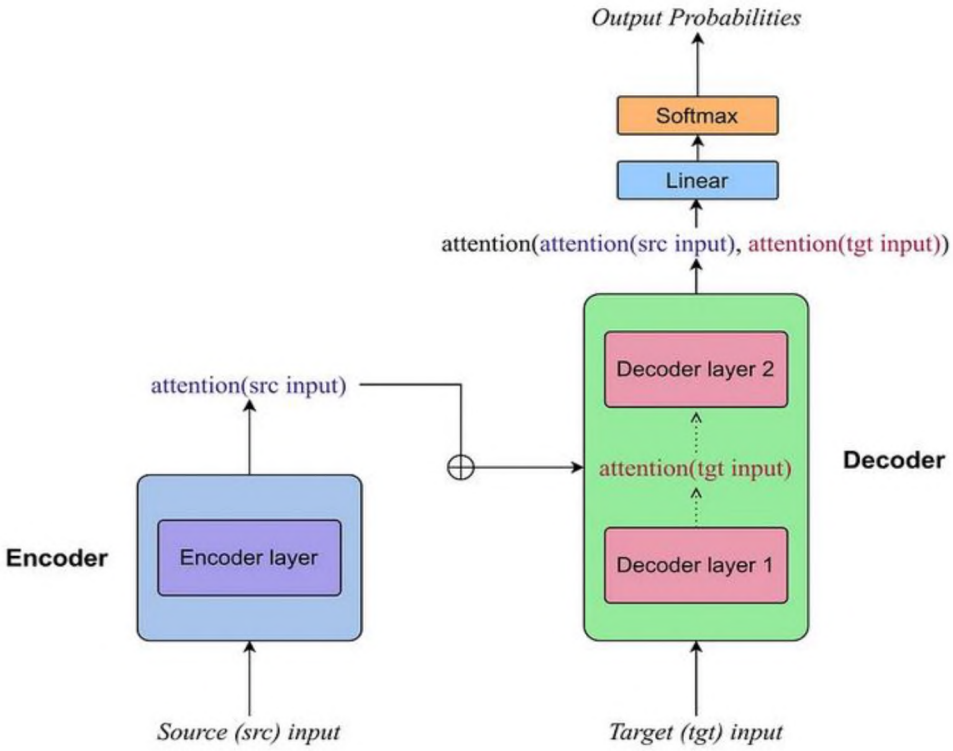

Рисунок 2.13 – Зовнішній вигляд Transformer [58]

Декодер приймає як перший вхід перекладене речення іспанською мовою (виводить на вихідну діаграму), звертає увагу, а потім поєднує результат із виходом

кодера до іншого механізму уваги. Декодер вчиться пов'язувати цільовий простір вбудовування (іспанською мовою) щодо вхідного простору вбудовування (англійською), щоб знаходити базове перетворення між обома векторними [57].

Вхідний текст Transformer вбудовано у багатовимірний векторний простір, тому замість речення вводиться послідовність векторів. Однак існує краща математична структура для представлення послідовностей векторів, матриць! І навіть більше того, навчаючи нейронну мережу, ми не навчаємо її вибірку за зразком, ми скоріше використовуємо пакети, в які упаковано кілька зразків. Отримані вхідні дані є тензором форми [N, L, E], де N – розмір партії, L – довжина послідовності, а E – розмір вбудовування.

Ліворуч двовимірний тензор представляє першу вбудовану послідовність партії. Праворуч повний пакетний тензор, що містить усі послідовності одного пакету, який подається в Transformer Encoder. Подібним чином цільовий тензор має таку саму форму, але містить справжній результат (рис 2.14).

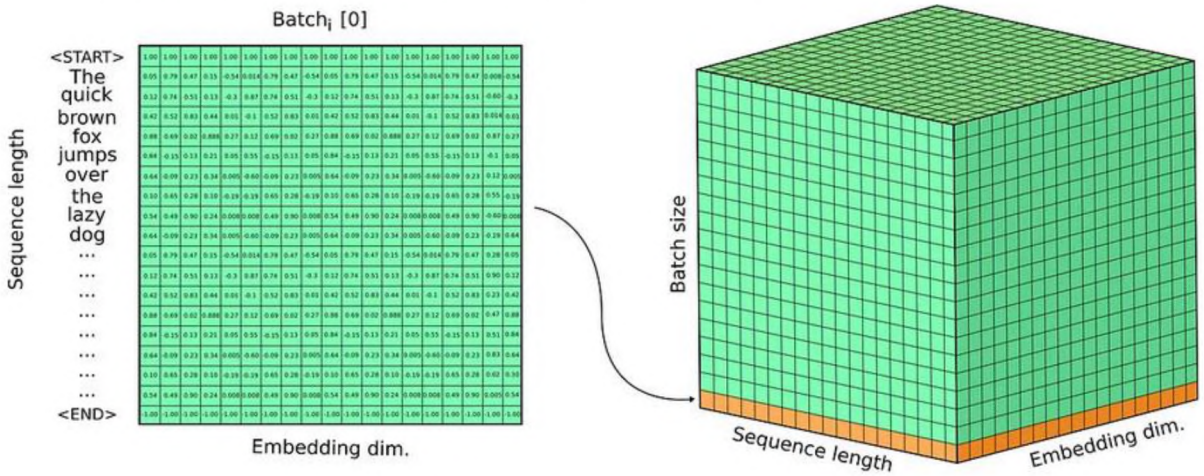

Рисунок 2.14 – Зовнішній вигляд вхідного тензора Source [59]

У вихідних даних Transformer, застосовується рівень Linear + Softmax, який створює певні ймовірності виходу (рівень Softmax виводить розподіл ймовірностей за визначеними класами). Результатом роботи Transformer є не перекладене речення, а розподіл ймовірностей цільового словника, який визначає слова з найвищою ймовірністю. Для кожної позиції в довжині послідовності генерується розподіл ймовірностей для вибору наступного маркера з вищою ймовірністю. Оскільки під час навчання Transformer обробляє всі речення одночасно, ми

отримуємо на виході тривимірний тензор, який представляє розподіли ймовірностей за словниковими лексемами форми [N, L, V], де N – розмір пакету, L — довжина послідовності, а V обсяг словникового запасу.

Ліворуч – прогноз розподілу ймовірностей у словнику для однієї послідовності (рис 2.15).

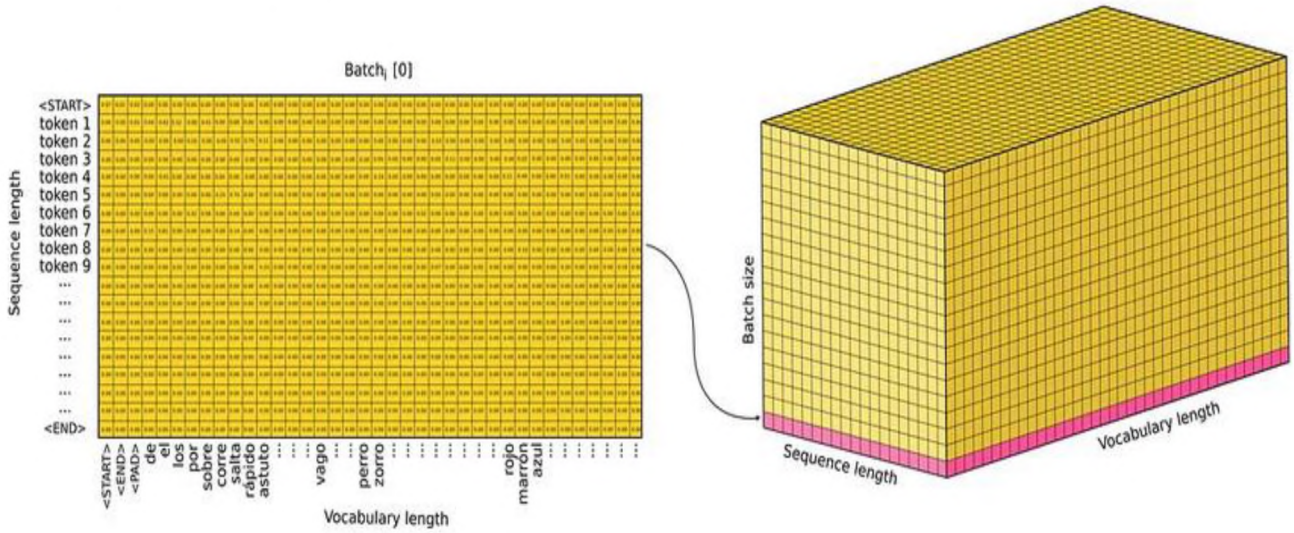

Рисунок 2.15 – Ілюстрація тензора ймовірностей виходу [60]

Кожен стовпець представляє слово в просторі цільового словника, а рядки відповідають маркерам послідовності. Праворуч повний прогнозований тензор для всієї партії (рис 2.15).

2.6 Вибір мови програмування та інструментів для реалізації моделі

Для реалізації ГММ можна використовувати різні мови програмування, залежно від вимог проекту. Python вважається найкращим вибором для роботи з ГММ через широкий вибір бібліотек для обробки природної мови (Natural Language Processing, NLP), таких як NLTK, SpaCy, і TensorFlow для роботи з глибоким навчанням. JavaScript може бути використаний для реалізації мовних алгоритмів в браузері. Застосування бібліотек, таких як TensorFlow.js або ml5.js, дозволяє використовувати глибоке навчання непрямо в браузері. Java використовується для

створення потужних мовних алгоритмів, особливо якщо потрібна висока продуктивність [61].

Якщо потрібна висока продуктивність то краще обрати C++, з їхнім широким діапазоном бібліотек. R часто використовується в аналізі даних і машинному навчанні, і його можна використовувати для створення генеративних мовних алгоритмів. В таблиці 2.1 було проаналізовано ефективність мов програмування за такими критеріями: зручність, наявність бібліотек, мобільність та ефективність.

Таблиця 2.1 – Оцінка ефективності мов програмування

Мова програмування	Зручність	Наявність бібліотек	Мобільність	Продуктивність
Python	+	+	+	+
Java	–	+	–	+
C++	–	+	–	+
R	+	–	–	+
JavaScript	+	–	+	–

Для даної роботи була обрана мова програмування Python. З урахуванням її зручності, ефективності та широким переліком бібліотек.

Python - високорівнева, інтерпретована, об'єктно-орієнтована мова програмування, розроблена Гвідо ван Россумом у 1991 році. Її синтаксис вирізняється простотою та читабельністю, сприяючи швидкому вивченню і розробці. Python використовується у різних галузях, включаючи веб-розробку, аналіз даних, штучний інтелект і машинне навчання, завдяки обширній екосистемі стандартних і сторонніх бібліотек [62].

Для розробки програмного коду використовувалося інтегроване середовище розробки PyCharm. PyCharm – це інтегроване середовище розробки (IDE), спеціально розроблене для розробки Python. Він розроблений компанією JetBrains, відомою створенням потужних інструментів розробки. PyCharm надає повний набір функцій, щоб допомогти розробникам Python ефективно писати, тестувати та налагоджувати код [63].

Основні функції PyCharm:

Редактор коду. PyCharm пропонує редактор коду з підсвічуванням синтаксису, автозавершенням і аналізом коду. Це допомагає розробникам писати чистий код без помилок.

Навігація кодом. IDE (Integrated Development Environment) надає інструменти для легкої навігації кодом, дозволяючи розробникам швидко переходити між файлами, класами, методами та іншими елементами коду.

Налагодження. PyCharm підтримує потужні інструменти налагодження, включаючи візуальний налагоджувач, який дозволяє розробникам встановлювати контрольні точки, перевіряти змінні та поетапно виконувати код.

Тестування. IDE підтримує різні інфраструктури тестування для Python, такі як unittest, pytest і doctest. Він надає інструменти для виконання тестів, перегляду результатів і керування конфігураціями тестів.Інтегрований термінал: PyCharm містить інтегрований термінал, який дозволяє розробникам запускати сценарії Python і керувати своїм середовищем розробки, не виходячи з IDE.

Контроль версій, він підтримує такі популярні системи контролю версій, як Git, Mercurial і Subversion, що дозволяє розробникам керувати своїми сховищами вихідного коду безпосередньо з IDE.

Веб-розробка. Хоча PyCharm розроблений в основному для Python, він також містить функції для веб-розробки, що робить його придатним для проектів, які включають як Python, так і веб-технології.

Інструменти бази даних. PyCharm надає інструменти для роботи з базами даних, дозволяючи розробникам взаємодіяти з базами даних безпосередньо з IDE.

Професійна версія. PyCharm має дві версії: Community та Professional. Версія Professional включає додаткові функції, такі як наукові інструменти, інструменти баз даних і підтримку таких веб-фреймворків, як Django та Flask.

Висновки до розділу 2

Виконано огляд існуючих підходів до моделювання мови, які використовують різні моделі.

Описано та представлено загальну структуру генеративних мовних алгоритмів, популярні алгоритми та методи генерації текстів, такі як RNN, GAN, Transformer та DNB.

Для побудови ГМА вибрано модель Transformer, обґрунтувано методологію моделювання, побудовано графічну модель для її подальшої реалізації.

У якості мови реалізації обрано Python та інструмент для реалізації моделі PyCharm.

РОЗДІЛ 3
ПРАКТИЧНІ АСПЕКТИ РЕАЛІЗАЦІЇ ГЕНЕРАТИВНИХ АЛГОРИТМІВ

3.1 Проєктування структури алгоритму

Алгоритм проєктування структури ГМА представлений на рисунку 3.1.

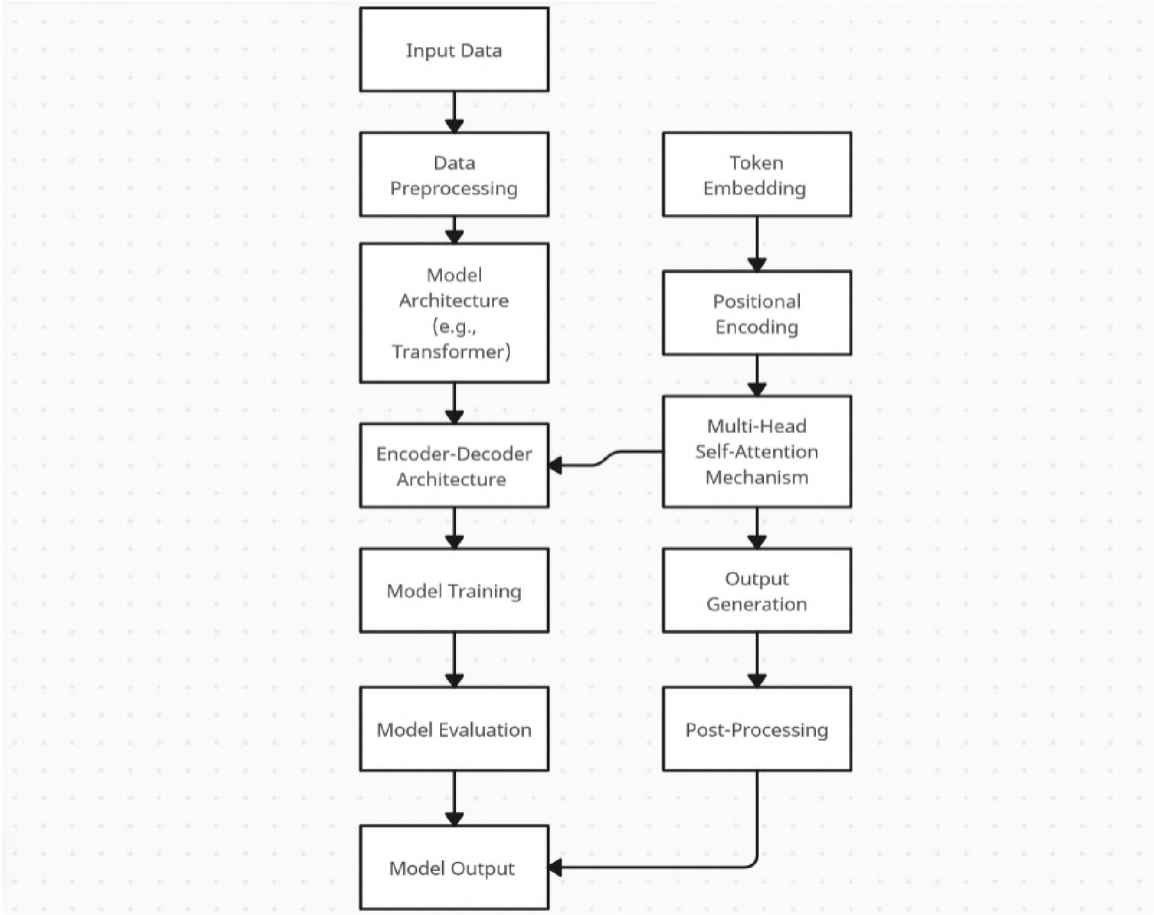

Рисунок 3.1 – Зовнішній вигляд структури алгоритму [64]

Вхідні дані (input data) – необроблені дані, які використовуються для навчання та висновків. Для багатьох завдань обробки природної мови (ОПМ), вхідними даними є текстова інформація. Це набір документів, великий корпус тексту чи навіть одне речення, залежно від завдання.

Попередня обробка даних (data preprocessing) – токенізація, обробка спеціальних символів та інші етапи попередньої обробки. Першим кроком при

роботі з текстовими даними є їх токенізація. Текст розбивається на окремі частини, такі як слова або підслова (токени). Це полегшує роботу моделі з текстом, конвертуючи його у послідовність числових представлень.

Вбудовування маркерів (token embedding) – перетворює вхідні маркери на безперервні векторні представлення. Токени перетворюються на вектори вбудовування слів. Ці вектори представляють собою числові вектори фіксованої довжини, які виражають семантику слів у просторі вбудовування. Цей процес допомагає моделі розуміти схожість між словами.

Позиційне кодування (positional encoding) – додає позиційну інформацію до вбудованих маркерів для розуміння послідовності.

Архітектура Transformer: архітектура кодера-декодера з механізмами самоконтролю.

Багатоголовий механізм самоуважності: дозволяє моделі зосереджуватися на різних частинах вхідної послідовності.

Тренування моделі: включає оптимізацію за допомогою функції втрат, зворотного поширення та градієнтного спуску.

Генерація виходу: генерує послідовності токенів як під час навчання, так і під час висновку.

Подальша обробка: додаткові кроки для підвищення якості згенерованого результату.

Оцінка моделі: метрики та процедури оцінки для оцінки ефективності моделі.

Вихід моделі: остаточний згенерований результат мовної моделі.

3.2 Реалізація ключових функцій та модулів

Реалізація моделі генеративної мови передбачає створення ключових функцій і модулів за допомогою інфраструктури глибокого навчання, такої як TensorFlow [65] або PyTorch [66].

TensorFlow – це відкрите програмне забезпечення для машинного навчання, розроблене командою Google Brain, що використовує концепцію графів обчислень для формалізації та оптимізації процесів навчання та використовується для розробки та тренування моделей штучного інтелекту.

Однією з ключових особливостей TensorFlow є здатність працювати з різними рівнями складності завдань, від простих моделей машинного навчання до глибоких нейронних мереж. Граф обчислень дозволяє ефективно виражати взаємозв'язки між операціями та використовується для оптимізації розподілених обчислень, що сприяє масштабованості тренування на великих обсягах даних.

Велика та активна спільнота користувачів сприяє поширенню знань та взаємодії у галузі машинного навчання. TensorFlow підтримує кілька мов програмування, таких як Python, C++, Java, що забезпечує його доступність для розробників із різних фахових напрямків.

Основною структурою даних в TensorFlow є тензори, які визначаються як многовимірні матриці даних. Ця абстракція спрощує операції над даними та взаємодію з різноманітними моделями. TensorFlow включає багатий набір вбудованих функцій та шарів, що допомагає розробникам у побудові та тренуванні різноманітних моделей для різних завдань машинного навчання.

TensorFlow є неодноразово випробуваним та високопродуктивним інструментом для реалізації завдань машинного навчання та глибокого навчання, і продовжує впливати на розвиток сучасних методів інтелектуального аналізу даних.

PyTorch представляє собою бібліотеку відкритого програмного забезпечення для машинного навчання, розроблену Facebook, яка набула широкої популярності у вчених та інженерів завдяки своєму динамічному підходу до обчислень та інтуїтивному інтерфейсу. Однією з ключових особливостей PyTorch є його підхід до динамічного обчислення, що дозволяє використовувати стандартні конструкції Python під час визначення та тренування моделей.

Тензори, які є основною структурою даних в PyTorch, взаємодіють із системою автоматичного диференціювання (Autograd), що дозволяє ефективно

обчислювати градієнти для оптимізації параметрів моделей. Це сприяє зручності та ефективності в розробці та налагодженні складних моделей машинного навчання.

Зручний інтерфейс PyTorch робить його популярним серед дослідників, оскільки він дозволяє легко реалізовувати та експериментувати з новими ідеями. Крім того, спільнота розробників PyTorch активно сприяє обміну знаннями та внесенню внесків у розвиток бібліотеки.

PyTorch також визначається своєю ролью у впровадженні та тренуванні моделей глибокого навчання, зокрема трансформерів, які здобули широке визнання у галузі обробки природної мови та інших областях.

Таким чином, PyTorch виконує ключову роль у сприянні розвитку та розширенні можливостей машинного навчання, забезпечуючи інструментарій, що дозволяє здійснювати розробку та налагодження моделей ефективно та інноваційно.

Даний код реалізує клас TokenEmbedding (рис. 3.2), який є частиною нейронної мережі для вбудовування токенів. Вбудовування токенів – це техніка, яка дозволяє нам представити слова або токени у вигляді векторів чисел фіксованої довжини.

```
import torch
import torch.nn as nn

class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)

    def forward(self, input_tokens):
        return self.embedding(input_tokens)
```

Рисунок 3.2 – Код реалізації класу TokenEmbedding

Описання класу TokenEmbedding.

Імпорт бібліотек:

– import torch – основна бібліотека для роботи з тензорами та машинним навчанням у PyTorch;

– import torch.nn as nn – одуль nn у PyTorch містить класи для визначення та навчання нейронних мереж.

Оголошення класу TokenEmbedding. Цей клас є підкласом nn.Module, що дозволяє використовувати PyTorch для автоматичного ведення журналу обчислень та автоматичної диференціації.

Конструктор класу.

```
__init__(self, vocab_size, embed_size)
```

Конструктор приймає два аргументи – vocab_size (розмір словника, кількість унікальних токенів) і embed_size (розмірність вектора вбудовування для кожного токена).

Створення об'єкта *Embedding.*

```
self.embedding = nn.Embedding(vocab_size, embed_size)
```

У конструкторі створюється об'єкт nn.Embedding, який буде використовуватися для вбудовування токенів. Цей об'єкт автоматично ініціалізує матрицю вбудовувань та навчатиме її під час тренування моделі.

Метод forward. *def forward (self, input_tokens)*. Цей метод визначає, як вхідні токени будуть передаватися через об'єкт вбудовування під час прямого проходження мережі.

```
return self.embedding(input_tokens)
```

Використовуючи об'єкт вбудовування, метод повертає вбудовані вектори для переданих токенів.

Клас PositionalEncoding (рис 3.3) реалізує позиційне кодування, яке використовується в моделях на базі трансформаторів, таких як GPT (Генеративний попередньо навчений трансформер). Позиційне кодування є ключовим, останні ці моделі не мають вбудованого розуміння порядку або позиції токенів у системі.

Описання класу PositionalEncoding.

Ініціалізація.

`__init__(self, embed_size, max_seq_len=512)`

Конструктор ініціалізує PosiitionalEncoding із вказаним розміром модуля вкладення (embed_size) та максимально довгою компанією (max_seq_len). Обчислюється позиційне кодування для залишків токенів до максимальної тривалості.

`self.encoding = torch.zeros(max_seq_len, embed_size)`

Цей рядок створює матрицю для зберігання позиційного кодування для кожної позиції в системі.

Позиція = *torch.arange(0, max_seq_len).unsqueeze(1).float()*. Цей рядок створює тензор, який представляє позицію від 0 до *max_seq_len – 1* і розширює його, щоб мати форму (*max_seq_len, 1*).

```
import torch
import torch.nn as nn

class PositionwiseFeedforward(nn.Module):
    def __init__(self, embed_size, ff_hid_dim):
        super(PositionwiseFeedforward, self).__init__()
        self.fc1 = nn.Linear(embed_size, ff_hid_dim)
        self.fc2 = nn.Linear(ff_hid_dim, embed_size)

    def forward(self, x):
        x = torch.nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Рисунок 3.3 – Код реалізації класу PositionalEncoding

`div_term = torch.exp(torch.arange(0, embed_size, 2).float() * -(math.log(10000.0) / embed_size))`

Цей рядок обчислює div_term. Він використовується у формулі позиційного кодування для введення різних частот для різних позицій.

```
self.encoding[:, 0::2] = torch.sin(позиція * div_term)
self.encoding[:, 1::2] = torch.cos(позиція * div_term)
```

Ці рядки заповнюють парні стовпці синуса позиції, помноженої на div_term, та непарні стовпці косинуса. Це загальний підхід для створення унікального позиційного кодування для кожної позиції.

```
self.encoding = self.encoding.unsqueeze(0)
```

Цей рядок надає розмірність партії до кодування. Прохід вперед: forward (self, x). Метод forward після вхідного тензора x і повертає вхідний тензор із доданим позиційним кодуванням.

```
return x + self.encoding[:, :x.size(1)].detach()
```

Цей рядок додає позиційне кодування до вхідного тензору. Частина [:, :x.size(1)] забезпечує відсутність позиційного кодування для відповідних позицій (відповідно до тривалості вхідної кампанії). Метод detach() використовується для того, щоб запобігти потік градієнтів через позиційне кодування, після чого це не є параметром, який потрібно оновити під час навчання.

Клас MultiHeadAttention (рис. 3.4) реалізує механізм багатоголового механізму самоуваги, який є ключовим елементом трансформерів, таких як GPT. Цей механізм дозволяє моделі фокусуватися на різних частинах вхідного контексту під час обробки.

Описання класу MultiHeadAttention.

Ініціалізація:

```
__init__(self, embed_size, heads)
```

Конструктор ініціалізує клас MultiHeadAttention з розміром вкладення (embed_size) та кількістю головок (heads). Розмір кожної головки (head_dim) обчислюється. Перевірка гарантує, що розмір вкладення є кратним розміру кожної головки.

```
self.values = nn.Linear(self.head_dim, self.head_dim, bias=False)
self.keys = nn.Linear(self.head_dim, self.head_dim, bias=False)
```

`self.queries = nn.Linear(self.head_dim, self.head_dim, bias=False)`

`self.fc_out = nn.Linear(heads * self.head_dim, embed_size)`

Ці рядки визначають лінійні шари для обчислення значень, ключів, запитів та фінального виводу. Важливо, що використовуються лінійні шари без зміщення (bias=False), оскільки це буде враховано в подальших обчисленнях.

```python
import torch
import torch.nn as nn
1 usage
class MultiHeadAttention(nn.Module):
    def __init__(self, embed_size, heads):
        super(MultiHeadAttention, self).__init__()
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        assert (
            self.head_dim * heads == embed_size
        ), "Embedding size needs to be divisible by heads"
        self.values = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.keys = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.queries = nn.Linear(self.head_dim, self.head_dim, bias=False)
        self.fc_out = nn.Linear(heads * self.head_dim, embed_size)

    def forward(self, values, keys, query, mask):
        # Get number of training examples
        N = query.shape[0]
        value_len, key_len, query_len = values.shape[1], keys.shape[1], query.shape[1]

        # Split the embedding into self.heads different pieces
        values = values.reshape(N, value_len, self.heads, self.head_dim)
        keys = keys.reshape(N, key_len, self.heads, self.head_dim)
        queries = query.reshape(N, query_len, self.heads, self.head_dim)
        values = self.values(values)
        keys = self.keys(keys)
        queries = self.queries(queries)
        energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])

        if mask is not None:
            energy = energy.masked_fill(mask == 0, float("-1e20"))
        attention = torch.nn.functional.softmax(energy / (self.embed_size ** (1 / 2)), dim=3)
        out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(
            N, query_len, self.heads * self.head_dim
        )
        out = self.fc_out(out)
        return out
```

Рисунок 3.4 – Код реалізації класу MultiHeadAttention

Прохід вперед:

`forward (self, values, keys, query, mask)`

Метод forward приймає значення, ключі, запит і маску та повертає вихід багатоголового шар самоуваги.

`N = query.shape[0]`

```
value_len, key_len, query_len = values.shape[1], keys.shape[1], query.shape[1]
```

Визначається кількість прикладів у навчальній вибірці та розміри значень, ключів та запитів.

```
values = values.reshape(N, value_len, self.heads, self.head_dim)
```

```
keys = keys.reshape(N, key_len, self.heads, self.head_dim)
```

```
queries = query.reshape(N, query_len, self.heads, self.head_dim)
```

Значення, ключі та запити розбиваються на окремі головки.

```
values = self.values(values)
```

```
keys = self.keys(keys)
```

```
queries = self.queries(queries)
```

Застосовуються лінійні перетворення до значень, ключів та запитів для кожної головки.

```
energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])
```

Обчислюється матриця уваги шляхом використання векторного добутку між запитами та ключами.

```
energy = energy.masked_fill(mask == 0, float("-1e20"))
```

Якщо задано маску, то вона використовується для маскування елементів матриці уваги.

```
attention = torch.nn.functional.softmax(energy / (self.embed_size ** (1 / 2)), dim=3).
```

Застосовується функція softmax для отримання нормалізованих ваг уваги.

```
out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(N, query_len, self.heads * self.head_dim)
```

Обчислюється вихід, використовуючи зважену суму значень для кожної головки.

```
out = self.fc_out(out)
```

Фінальний вихід формується за допомогою лінійного шару.

```
return out
```

Результат повертається.

Клас PositionwiseFeedforward (рис 3.5) реалізує модель позиційно-залежного напереднього шару (Position-wise Feedforward Layer), який використовується в трансформер-подібних моделях для обробки та трансформації векторів з кожної позиції входу.

```python
import torch
import torch.nn as nn

class PositionwiseFeedforward(nn.Module):
    def __init__(self, embed_size, ff_hid_dim):
        super(PositionwiseFeedforward, self).__init__()
        self.fc1 = nn.Linear(embed_size, ff_hid_dim)
        self.fc2 = nn.Linear(ff_hid_dim, embed_size)

    def forward(self, x):
        x = torch.nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Рисунок 3.5 – Код реалізації класу PositionwiseFeedforward

Описання класу PositionwiseFeedforward.

Ініціалізація.

```
__init__(self, embed_size, ff_hid_dim)
```

Конструктор ініціалізує клас PositionwiseFeedforward з розміром вкладення (embed_size) та розміром прихованого шару (hidden layer) (ff_hid_dim).

```
self.fc1 = nn.Linear(embed_size, ff_hid_dim)
```

```
self.fc2 = nn.Linear(ff_hid_dim, embed_size)
```

Ці рядки створюють два лінійних шари (fc1 та fc2). Перший шар отримує вхід розміру embed_size та виводить прихований шар розміром ff_hid_dim. Другий шар отримує вхід розміром ff_hid_dim та повертає вихід розміру embed_size.

```
forward(self, x)
```

Метод forward виконує переднє поширення через цей позиційно-залежний напередній шар.

```
x = torch.nn.functional.relu(self.fc1(x))
```

Вхідний тензор x проходить через перший лінійний шар та функцію активації ReLU. Функція ReLU активує лише позитивні значення, допомагаючи у вираженні нелінійних залежностей в даних.

```
x = self.fc2(x)
```

Результат з першого шару подається на другий лінійний шар без функції активації. Це дозволяє моделі виучувати складні залежності між входом та вихідним сигналом.

```
return x
```

Результат операцій повертається.

3.3 Тестування та налаштування алгоритму

Для взаємодії з моделлю ГММ використовується бібліотека gpt-3.5-turbo [67]. Цей алгоритм є частиною системи для чат-ботів або автоматизованих систем NLP (рис. 3.6).

Обробка природної мови (NLP) відноситься до галузі інформатики, а точніше до галузі штучного інтелекту або штучного інтелекту, яка займається наданням комп'ютерам здатності розуміти текст і вимовлені слова майже так само, як це можуть зробити люди.

Імпорт бібліотеки чи модуля g4f. Імпорт бібліотеки g4f, який містить функціонал для взаємодії з моделлю GPT-3.5 Turbo.

Створення запиту до GPT-3.5 Turbo.

```
response = g4f.ChatCompletion.create(...)
```

Створюється запит до моделі GPT-3.5 Turbo для завершення чат-повідомлення.

```
import g4f

response = g4f.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "Хто кращий баскетболіст в світі?"}],
    stream=True,
)

for message in response:
    print(message, flush=True, end='')

```

Рисунок 3.6 – Зовнішній вигляд автоматизованої системи NLP

Визначення ролей та повідомлень. messages=[{"role": "user", "content": "Хто кращий баскетболіст в світі?"}]. Визначається список повідомлень для передачі моделі. Вказується одне повідомлення користувача ("user") з вмістом "Хто кращий баскетболіст в світі?".

Використання stream=True. Параметр вказує на використання стрімінгового підходу при обробці відповіді. Це корисно для отримання часткових результатів чи взаємодії з моделлю в реальному часі.

Виведення результатів. for message in response: print(message, flush=True, end=''). Проходиться по відповідям, отриманим від моделі, та виводить їх. Функція flush=True гарантує, що виведення буде негайно скинуте на екран.

Цей алгоритм взаємодіє з GPT-3.5 Turbo, задає запит на завершення чат-повідомлення та отримує результати в режимі стріму.

3.4 Експериментальна оцінка реалізованого алгоритму

Експериментальна оцінка реалізованого алгоритму включає в себе оцінку його ефективності та продуктивності.

У вищезгаданому фрагменті коду був виконаний експеримент, що має на меті оцінку логічного мислення ГММ. Процес експерименту включав в себе подачу спеціально сформульованих контекстних запитань генеративному мовному алгоритму з метою вивчення його здатності сприймати та адекватно реагувати на логічні аспекти інформації.

Експериментальний підхід включав в себе введення послідовності запитань, представлених у формі тексту, в якому зосереджено налагоджувалася логіка та контекстуальне розуміння ГМА. Зокрема, спостерігалося, наскільки алгоритм може виявляти логічні зв'язки в контексті та правильно відповідати на запитання, вимагаючи розуміння поданого контексту.

Приклад експерименту, який може бути використаний для оцінки логічного мислення ГММ.

Мета експерименту: оцінити здатність ГММ сприймати та адекватно реагувати на логічні аспекти інформації.

Процедура експерименту.

Підготовка даних: розробка набору контекстних запитань, які вимагають логічного аналізу. Ці запитання можуть включати логічні головоломки, задачі на умовивід, а також сценарії з множинними змінними. Запитання мають охоплювати різні рівні складності та типи логіки (індуктивна, дедуктивна, абдуктивна).

Проведення експерименту:

– подача запитань до генеративного мовного алгоритму;

– запис відповідей алгоритму на кожне запитання;

– повторення процесу з різними алгоритмами або версіями одного алгоритму для порівняння результатів.

Аналіз відповідей:

– оцінка точності та релевантності відповідей;

– аналіз здатності алгоритму до логічного мислення, умовиводу та вирішення проблем;

– порівняння відповідей різних алгоритмів для виявлення сильних та слабких сторін.

Фіксація результатів та їх інтерпретація:

– документування результатів та їх аналіз;

– формулювання висновків щодо рівня логічного мислення алгоритму.

Оцінка та висновки:

– підведення підсумків ефективності алгоритму у вирішенні логічних завдань;

– визначення обмежень та потенційних напрямків для покращення.

Цей експеримент дозволяє оцінити, наскільки добре ГММ може справлятися з логічними завданнями, і як він адаптується до різних типів логічних викликів.

Таблиця 3.1 – Запитання з різними типами логічного аналізу

Тип логіки	**Запитання**
Дедуктивна	Якщо всі люди є смертними, і Сократ є людиною, чи є Сократ смертним?
	У вазі є тільки червоні та зелені яблука. Якщо ви вийняли 10 червоних яблук, чи всі яблука, що залишилися у вазі, зелені?
Індуктивна	Ви бачите, що кожного ранку сонце встає на сході. Який висновок ви можете зробити про схід сонця завтрашнього ранку?
	Ви спостерігали, що кожен раз, коли ви купуєте морозиво в цьому магазині, воно буває або шоколадне, або ванільне. Які шанси, що наступного разу вони будуть мати інший смак?
Абдуктивна	Ви знаходите мокрий слід від черевиків у вашому будинку. Яке найбільш імовірне пояснення?
	Ваш друг завжди носить шапку, коли йде на роботу. Одного дня ви бачите його без шапки. Які можливі пояснення?
Логічні головоломки	У кімнаті двоє батьків та двоє дітей. Разом вони важать 140 кг. Батько важить 40 кг більше, ніж мати, а одна дитина важить половину від ваги матері. Скільки важить кожен з них?
	Якщо 5 машин роблять 5 деталей за 5 хвилин, скільки часу потрібно 100 машинам, щоб зробити 100 деталей?

Ця таблиця допомагає систематизувати і класифікувати запитання за типом логіки, що уможливлює більш зручний аналіз та оцінку відповідей

Практична реалізація експерименту.

Перший рядок коду імпортує бібліотеку g4f. Його встановлено за допомогою команди pip install g4f.

Визначення функції ask_gpt (рис 3.7):

Функція ask_gpt отримує рядок message в якості введення, передає його моделі GPT (здається, GPT-3.5 Turbo) та повертає отриману відповідь.

```
def ask_gpt(message: str) -> str:
    response = g4f.ChatCompletion.create(
        model=g4f.models.gpt_35_turbo,
        messages=[{"role": "user", "content": message}])
    return response
```

Рисунок 3.7 – Зовнішній вигляд функції ask_gpt

Виклик функції та виведення результату (рис. 3.8). print(ask_gpt ("Напиши зайве слово з представлених: стіл, сервант, чашка, собака")).

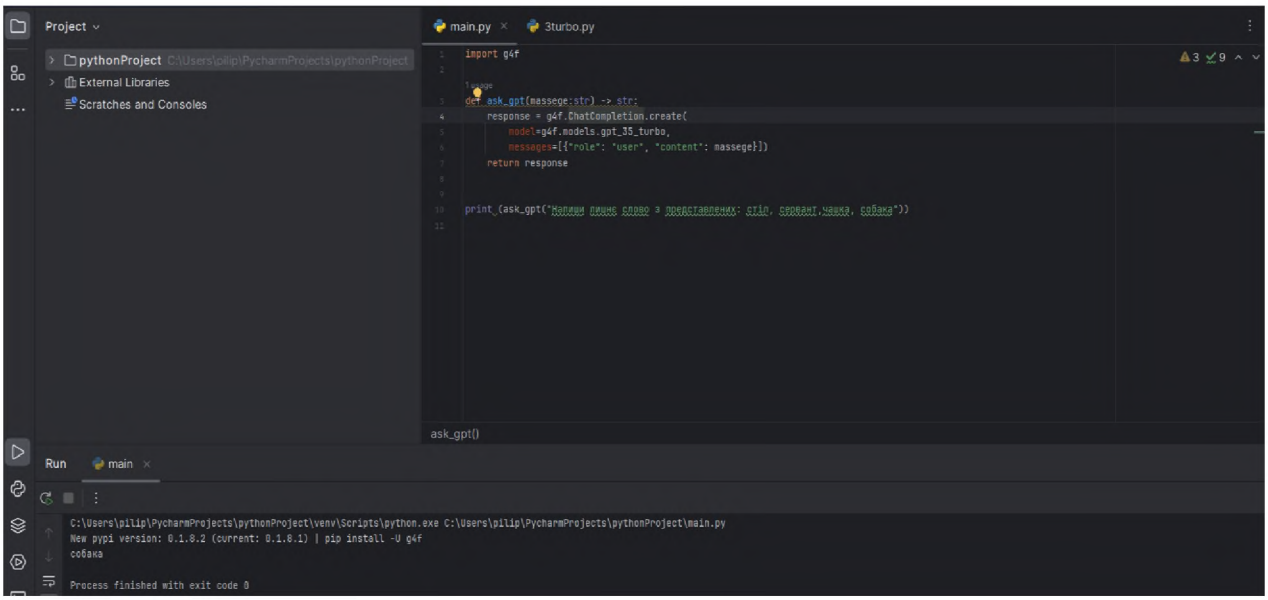

Рисунок 3.8 – Зовнішній вигляд виклику функції та виведення результату

Виконується функція ask_gpt з певним повідомленням та виводиться результат.

Далі була проведена робота з контекстом (рис 3.9). Відбулося редагування визначення функції ask_gpt для поставленої задачі.

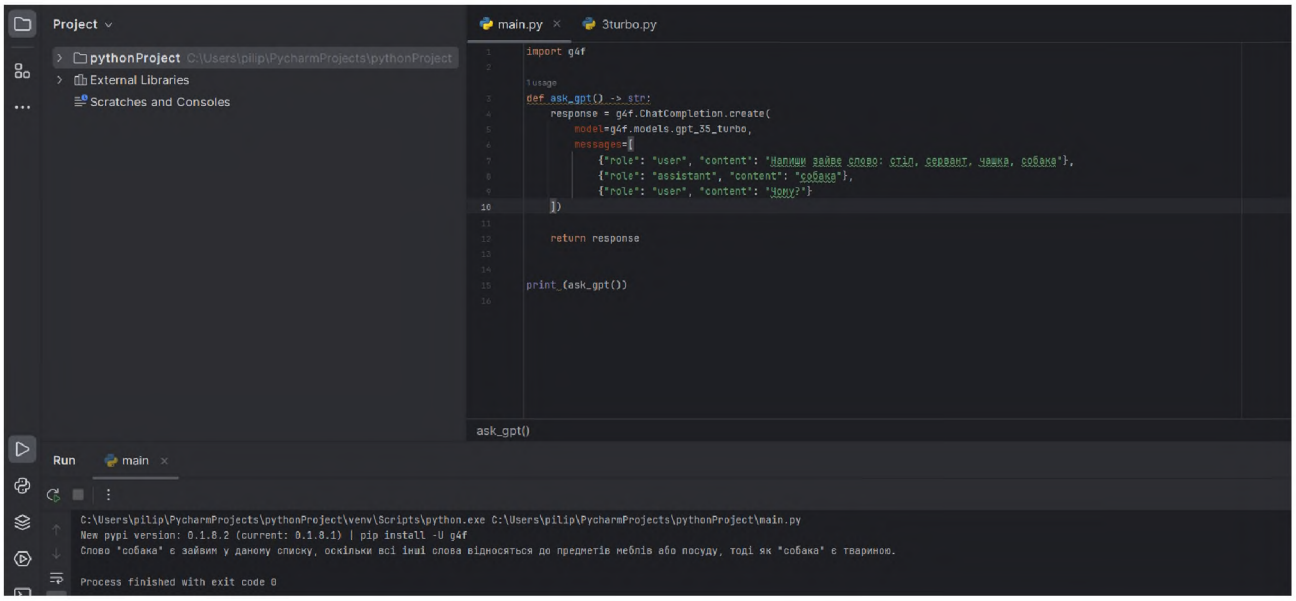

Рисунок 3.9 – Зовнішній вигляд робота з контекстом

В рамках цього експерименту визначалася здатність ГМА розуміти та обробляти запитання на основі поданого контексту. Результати експерименту слугують основою для подальшого розуміння та вдосконалення здатностей ГМА у логічному аналізі інформації та відповіді на контекстні питання.

В експерименті для оцінки ГММ можна використовувати наступні метрики:

1. Точність відповідей – оцінка, чи є відповіді алгоритму коректними та логічно обґрунтованими. Для цього можна порівняти відповіді алгоритму зі стандартними відповідями або відповідями експертів.

2. Релевантність та адекватність – оцінка, наскільки відповіді відповідають контексту заданого питання. Важливо, щоб відповіді не тільки були правильними, але й відповідали суті заданого запитання.

3. Консистентність – аналіз, чи відповіді алгоритму є послідовними та непротирічними при вирішенні різних запитань. Важливо, щоб алгоритм демонстрував стабільність у своїх рішеннях.

4. Час відповіді – вимірювання часу, який потрібен алгоритму для надання відповіді. Це важливо для оцінки ефективності алгоритму в реальних умовах використання.

5. Здатність до узагальнення – оцінка, наскільки добре алгоритм може застосовувати навчене на різноманітних прикладах та в різних сценаріях.

6. Об'єктивність та відсутність упереджень – аналіз, чи алгоритм відображає упередження або неправильно інтерпретує інформацію на основі стереотипів чи недостатньої інформації.

Ці метрики дозволяють оцінити здатність генеративного мовного алгоритму до логічного мислення, обробки інформації та прийняття рішень, а також виявити потенційні слабкі місця для подальшого вдосконалення.

Оцінка в діапазоні від 0 до 1 відображає рівень досягнутого рівня компетентності алгоритму в кількох ключових аспектах.

Точність відповідей (0.9). Ця метрика визначає, наскільки коректними та логічно обґрунтованими є відповіді алгоритму. Оцінка 0.9 свідчить про високу ступінь збігу зі стандартними або експертними відповідями.

Релевантність та адекватність (0.8). Ця метрика відображає, наскільки відповіді алгоритму вписуються у контекст запитань. Оцінка 0.8 свідчить про загалом добру адаптацію до контексту, іноді можливі невеликі відхилення.

Консистентність (0.85). Ця метрика оцінює послідовність та стабільність вирішення різних запитань. Оцінка 0.85 свідчить про високий рівень послідовності відповідей.

Час відповіді (0.7). Метрика визначає час, який алгоритм витрачає на формулювання відповідей. Оцінка 0.7 вказує на середню ефективність в реальному часі.

Здатність до узагальнення (0.8). Ця метрика відображає, наскільки добре алгоритм може застосовувати знання на різноманітних прикладах. Оцінка 0.8 свідчить про високу узагальнюючу здатність.

Об’єктивність та відсутність упереджень (0.9). Метрика аналізує, чи алгоритм виявляє упередження або неправильно інтерпретує інформацію. Оцінка 0.9 вказує на високий рівень об’єктивності.

Середня оцінка: 0.825.

Загальна оцінка підкреслює високий рівень ефективності алгоритму, проте можливість покращення в області часу відповіді.

3.5 Аналіз отриманих результатів

Загальна оцінка ефективності генеративного мовного алгоритму на основі аналізу вказаних метрик становить 0.825 з можливого максимуму 1.

Точність відповідей (0.9). Висока оцінка відображає, що алгоритм надає правильні та логічно обґрунтовані відповіді. Збіг з експертними даними свідчить про високу точність у відтворенні інформації.

Релевантність та адекватність (0.8). Задовільна оцінка вказує на те, що відповіді часто вписуються у контекст запитань, хоча можливі невеликі відхилення. Для поліпшення може бути важливо вдосконалити алгоритмічну стратегію адаптації до специфіки запитань.

Консистентність (0.85). Висока консистентність вказує на те, що алгоритм вирішує різні завдання стабільно та послідовно. Це є ключовим показником для надійності та неперервної продуктивності алгоритму.

Час відповіді (0.7). Знижена оцінка швидкості вказує на те, що алгоритм може вимагати додаткових оптимізацій у відповіді в реальному часі. Швидкість реакції є важливим аспектом для задач з великим обсягом обчислень.

Здатність до узагальнення (0.8). Задовільна здатність до узагальнення вказує на те, що алгоритм вміє застосовувати навчене на різноманітних прикладах, хоча можливі покращення у роботі з новими, непередбаченими сценаріями.

Об’єктивність та відсутність упереджень (0.9). Висока оцінка вказує на те, що алгоритм виявляє об’єктивність та не відтворює упередження. Це є критичним для забезпечення справедливості та рівноправ'я використання алгоритму.

Середній бал (0.825). Середній бал обчислюється як середнє арифметичне значень усіх метрик. Це загальний показник ефективності, який враховує всі аспекти, і вказує на загально високий рівень алгоритму, з деяким простором для покращень, зокрема в області часу відповіді та адаптації до конкретних запитань.

3.6 Економічна оцінка проєкту

Витрати на впровадження ГММ (рис. 3.10) можуть значно варіюватися в залежності від різних факторів, таких як складність завдання, обсяг даних, обчислювальні ресурси та використані технології.

Chat GPT Competitor	Cost	Billed	Free trial	Best for
Chat GPT Free	Free			Casual user
Chat GPT Plus	$20/month	Monthly	Y	Frequent use, more complex problems
Chatsonic	$19/month	Monthly / Annually	Y	Longer responses, up-to-date, image and text generation
Bing Chat	Free			Search assistant
Bard	Free			More complex problem solving, coding
Blenderbot	Free			Experimental AI chat testing
Jasper Chat	$49/month	Monthly / Annually	Y	Long-form responses and content creation

Рисунок 3.10 – Вартість послуг ChatGPT [68]

Ось декілька ключових пунктів, які можуть впливати на вартість:

Обсяг та складність даних. Залежно від кількості та якості даних, необхідних для навчання алгоритму, може змінюватися час та ресурси, які необхідно витратити.

Технічна складність. Складність алгоритму та використовувані технології можуть впливати на вартість. Наприклад, використання глибокого навчання може вимагати значних обчислювальних ресурсів.

Етапи впровадження. Якщо потрібно розробити та навчити алгоритм спочатку, це може збільшити вартість проєкту.

Відомість технологій та експертність. Якщо у вас вже є команда з досвідом у сфері машинного навчання та обробки природної мови (NLP), це може знизити вартість проєкту. З іншого боку, якщо потрібно наймати експертів, це може збільшити витрати.

Ліцензії та обслуговування. Деякі генеративні мовні алгоритми можуть використовувати комерційні бібліотеки або платформи, що може збільшити вартість через ліцензії та обслуговування.

Хоча ChatGPT все ще безкоштовний, ChatGPT Plus надає додаткові переваги підприємствам і окремим особам, які їх справді потребують. План підписки коштує 20 доларів США на місяць і допомагає фінансувати та розвивати ChatGPT, тобто передплатники підтримують платформу та забезпечують її доступність як безкоштовну послугу для всіх інших.

Ada є найшвидшою та найдешевшою моделлю і здатна виконувати дуже прості завдання. Він може обробляти такі речі, як розбір тексту, виправлення адреси та певні типи завдань класифікації, які не є надто складними. Вартість: $0,0004/1k токенів.

Babbage може виконувати прості завдання, як класифікація. Він також може виконувати завдання семантичного пошуку та ранжувати, наскільки документи відповідають пошуковим запитам. Вартість: $0,0005/1k токенів.

Curie забезпечує баланс між швидкістю та потужністю. Він задовільно працює для складніших завдань, як-от узагальнення та класифікація настроїв. Вартість: 0,002 $/1 тис. жетонів.

Davinci є потужною моделюю інструкцій GPT3. Він відповідає за контекстне розуміння та відповіді, які можна отримати від веб-інтерфейсу ChatGPT, ця модель забезпечить подібну продуктивність. Вартість: 0,02 $/1 тис. жетонів.

Як і ChatGPT Plus, Chatsonic працює на основі GPT-4. Його перевага полягає в тому, що він навчається на основі останніх даних в Інтернеті, тому він надаватиме відповіді щодо поточних подій і тенденцій. Він також мультимодальний, тому може розуміти зображення. Але, на відміну від ChatGPT Plus, він також має можливість генерувати зображення безпосередньо у вікні чату. Він пропонується як частина продукту для створення контенту Writesonic AI. Безкоштовна пробна версія – 10 000 слів на місяць – без GPT-4. Платно – від 19 доларів США на місяць за 100 000 слів і GPT-4.

Bing Chat – це потужний інтерфейс чату Microsoft AI. Він використовує GPT-4 і навчається на поточних даних Інтернету. Це безкоштовно, але дещо обмежено, оскільки користувач отримує 20 чатів за сеанс і 200 щоденних чатів. Він також надає коротші та менш глибокі відповіді, ніж ChatGPT Free або Plus. Відповіді більше ті, яких користувач очікує від чат-бота служби підтримки клієнтів, ніж від належного генератора контенту. Іншим обмеженням є те, що він доступний лише в браузері Edge від Mircosoft.

Google's Bard – це новий інтерфейс чату зі штучним інтелектом від Google на базі мовної моделі PaLM 2. Коли завантажується сторінка, отримується багато нагадувань про те, що інструмент знаходиться в бета-версії, та іноді дає неправильну відповідь.

Його все ще перевіряють, щоб дізнатися, чи дає він такі ж результати або кращі, ніж GPT4, але він навчений на більшому наборі даних і є більш легким рішенням. Наразі це безкоштовний інструмент, і користувачу потрібен лише обліковий запис Google, щоб ним користуватися.

Meta експериментує зі своїм інтерфейсом чату AI під назвою Blenderbot 3. Він був розроблений спільно з OpenAI і все ще є експериментальним. Його багато критикували за некоректні відповіді. Ймовірно, незабаром його замінить новий інструмент, що використовує останню модель мови Llama від Meta. Безкоштовно – доступно лише в США.

Jasper є одним із інструментів для копірайтингу зі штучним інтелектом, і його інтерфейс чату працює дуже схоже на ChatGPT. Він працює на базі GPT-3.5, але

його було налаштовано так, щоб він було більш придатним для бізнес-випадків використання, як-от маркетинг і продажі. Його вартість 49 доларів США на місяць.

Висновки до розділу 3

У розділі розглянуто практичні аспекти реалізації генеративних мовних алгоритмів.

Виконано проєктування структури алгоритму для реалізації в реальність плану дій. Спочатку вносяться вхідні дані, потім дані обробляються, вбудовуються маркери та позиційно кодуються. Будується архітектура кодера-декодера з механізмамом самоконтролю та модель зосереджується на різних частинах вхідної послідовності. Модель тренується, генерується послідовність токенів. Модель підлягає обробці, оцінці та тільки після цих кроків виходить остаточний згенерований результат мовної моделі.

Здійснена реалізація ключових функцій та модулів за допомогою інфраструктури глибокого навчання PyTorch. Клас TokenEmbedding вбудовує токени, класу PositionalEncoding реалізує позиційне кодування, клас MultiHeadAttention реалізує механізм багатоголового механізму самоуваги, який є ключовим елементом трансформерів та клас PositionwiseFeedforward реалізує модель позиційно-залежного напереднього шару.

Виконане тестування та налаштування програми за допомогою API Chat GPT. Імпортовано бібліотеку g4f та створено запит до моделі GPT-3.5 Turbo. Визначаються ролі та повідомлення, використовується стрімінговий підход при обробці відповіді та виводяться результати.

Проведено експерименти для оцінки ефективності та продуктивності реалізованого алгоритму. Експериментальний підхід включав в себе введення послідовності запитань, представлених у формі тексту, в якому зосереджено налагоджувалася логіка та контекстуальне розуміння ГМА.

Метою визначалась оцінка здатності ГММ сприймати та адекватно реагувати на логічні аспекти інформації.

В експерименті для оцінки ГММ можна використовувати наступні метрики: Точність відповідей, релевантність та адекватність, консистентність, час відповіді, здатність до узагальнення, об'єктивність та відсутність упереджень.

Оцінка в діапазоні від 0 до 1 відображає рівень досягнутого рівня компетентності алгоритму в кількох ключових аспектах. Середня оцінка: 0.825. Загальна оцінка підкреслює високий рівень ефективності алгоритму.

Проведено аналіз отриманих результатів. Загальна оцінка ефективності генеративного мовного алгоритму на основі аналізу вказаних метрик становить 0.825 з можливого максимуму 1. Точність відповідей (0.9), релевантність та адекватність (0.8), консистентність (0.85), час відповіді (0.7), здатність до узагальнення (0.8), об'єктивність та відсутність упереджень (0.9). Можливість покращення в області часу відповіді.

Економічно обґрунтовано дану модель, розглянуті всі можливі моделі Chat GPT з урахуванням їх ціни. ChatGPT все ще безкоштовний, ChatGPT Plus надає додаткові переваги за 20 доларів США на місяць. Ada є найшвидшою та найдешевшою моделлю і здатна виконувати дуже прості завдання її вартість складає $0,0005/1k токенів. Curie забезпечує хороший баланс між швидкістю та потужністю за 0,002 $/1 тис. жетонів. Davinci дає контекстне розуміння та відповіді, які можна отримати від веб-інтерфейсу ChatGPT, вартістю в 0,02 $/1 тис. жетонів.

Chatsonic перевага полягає в тому, що він навчається на основі останніх даних в Інтернеті, тому він надаватиме відповіді щодо поточних подій і тенденцій. Безкоштовна пробна версія – 10 000 слів на місяць – без GPT-4. Платно – від 19 доларів США на місяць за 100 000 слів і GPT-4. Bing Chat – це потужний інтерфейс чату Microsoft AI. Це безкоштовно, але дещо обмежено, оскільки користувач отримуєте 20 чатів за сеанс і 200 щоденних чатів.

Meta експериментує зі своїм інтерфейсом чату AI під назвою Blenderbot 3, безкоштовний в США. Jasper є одним із найкращих інструментів для копірайтингу

зі штучним інтелектом, і його інтерфейс чату працює дуже схоже на ChatGPT. Його вартість 49 доларів США на місяць.

Виконано проєктування структури алгоритму для реалізації в реальність плану дій. Здійснена реалізація ключових функцій та модулів за допомогою інфраструктури глибокого навчання PyTorch. Виконане тестування та налаштування програми за допомогою API Chat GPT. Проведено експерименти для оцінки ефективності та продуктивності реалізованого алгоритму. Проведено аналіз отриманих результатів. Економічно обґрунтовано дану модель, розглянуті всі можливі моделі Chat GPT з урахуванням їх ціни.

Загальна оцінка ефективності генеративного мовного алгоритму на основі аналізу вказаних метрик становить 0.825 з можливого максимуму 1. Точність відповідей (0.9), релевантність та адекватність (0.8), консистентність (0.85), час відповіді (0.7), здатність до узагальнення (0.8), об'єктивність та відсутність упереджень (0.9). Можливість покращення в області часу відповіді.

В рамках цієї роботи визначалася здатність ГМА розуміти та обробляти запитання на основі поданого контексту. Результати експерименту слугують основою для подальшого розуміння та вдосконалення здатностей ГМА у логічному аналізі інформації та відповіді на контекстні питання.

ВИСНОВКИ

Результати виконання кваліфікаційної роботи дозволяють зробити наступні висновки.

ГММ застосовуються в різноманітних завданнях обробки природної мови, таких як машинний переклад, автоматичне завершення тексту, створення синтетичних даних для тренування інших моделей, генерація описів зображень тощо. Окремі ГММ, такі як GPT (Generative Pre-trained Transformer), демонструють здатність до вивчення широкого спектру знань з великих корпусів тексту та використання цих знань для генерації нового контенту.

Оглянуто початкові дослідження, які використовували засновані на правилах та статистичні підходи для обробки природної мови. Метою яких, було розробити алгоритми, які здатні адекватно розпізнавати та генерувати текст на основі лінгвістичних правил та частоти вживання слів. Також, сучасні методи моделювання текстів та мови включають в себе різноманітні підходи та алгоритми, які базуються на глибокому навчанні, трансформаційних моделях та інших передових техніках.

Проведений аналіз показав, як існують теоритичні підходи до моделювання мови. Розглянуто загальну структуру ГММ. Описано популярні алгоритми та методи генерації текстів, таких як: RNN, GAN, Transformer та DBN. Відбувся вибір продуктивнішої моделі з поданих, нею виявилася Transformer. Проаналізована архітектура кодеру та декодеру Transformer (BERT). Створено та проілюстровано теоритичну модель генеративного алгоритму Transformer, як приклад, завдання перекладу тексту з англійської на іспанську.

Проведено аналіз ефективності мов програмування для реалізації ГММ. З урахуванням зручності, наявності бібліотек, мобільності та продуктивності вибрано мову програмування Python, як саму універсальну. Для розробки програмного коду використовувалося інтегроване середовище розробки PyCharm.

Виконано проєктування структури алгоритму для реалізації в реальність плану дій. Здійснена реалізація ключових функцій та модулів за допомогою

інфраструктури глибокого навчання PyTorch. Виконане тестування та налаштування програми за допомогою API Chat GPT.

Проведено експерименти для оцінки ефективності та продуктивності реалізованого алгоритму. Експериментальний підхід включав в себе введення послідовності запитань, представлених у формі тексту, в якому зосереджено налагоджувалася логіка та контекстуальне розуміння ГМА.

Метою визначалась оцінка здатності ГММ сприймати та адекватно реагувати на логічні аспекти інформації.

В експерименті для оцінки ГММ можна використовувати наступні метрики: Точність відповідей, релевантність та адекватність, консистентність, час відповіді, здатність до узагальнення, об'єктивність та відсутність упереджень.

Оцінка в діапазоні від 0 до 1 відображає рівень досягнутого рівня компетентності алгоритму в кількох ключових аспектах. Середня оцінка: 0.825. Загальна оцінка підкреслює високий рівень ефективності алгоритму.

Проведено аналіз отриманих результатів. Загальна оцінка ефективності генеративного мовного алгоритму на основі аналізу вказаних метрик становить 0.825 з можливого максимуму 1. Точність відповідей (0.9), релевантність та адекватність (0.8), консистентність (0.85), час відповіді (0.7), здатність до узагальнення (0.8), об'єктивність та відсутність упереджень (0.9). Можливість покращення в області часу відповіді.

Економічно обґрунтовано дану модель, розглянуті всі можливі моделі Chat GPT з урахуванням їх ціни. Вартість кожної моделі залежить від точності відповідей, релевантності та адекватності, консистентності, швидкості відповіді, здатності до узагальнення, об'єктивності та відсутності упереджень.