**Лекція 6. Великі мовні моделі**

1. **Поняття та еволюція**

**Великі мовні моделі (Large Language Models, LLMs)** — це штучні нейронні мережі, які навчаються на великих обсягах текстових даних для розуміння, обробки та генерації природної мови.

Основні властивості LLMs:

* Розуміння контексту та значення слів у реченні
* Генерація зв’язного та змістовного тексту
* Вміння відповідати на запитання, перекладати мови, узагальнювати тексти
* Використання сотень мільярдів параметрів (GPT-4, LLaMA, Gemini)

**Еволюція мовних моделей**

1. **N-грамна модель** — це статистичний метод для прогнозування наступного елемента в послідовності на основі попередніх **n-1** елементів. Цей підхід широко використовується в обробці природної мови для таких задач, як **автозавершення, розпізнавання мови, переклад текстів** тощо.

**N-грам** — це послідовність з **n** елементів (наприклад, слів або символів), які використовуються для моделювання мови. Кожен елемент послідовності є частиною контексту, в якому інші елементи є можливими.

* **1-грам (уніграм)**: один елемент (слово або символ) без контексту попередніх елементів.
* **2-грам (біграм)**: послідовність з двох елементів, враховує тільки попереднє слово.
* **3-грам (триграм)**: послідовність з трьох елементів, враховує два попередніх слова.
* І так далі для **n-грам**.

Прогнозування в **n-грамній моделі** здійснюється на основі ймовірностей наступного елемента в послідовності, що залежить від **n-1** попередніх елементів. Наприклад:

Для **2-грамної моделі (біграм)**, ймовірність слова **w₂** після слова **w₁** визначається як:



**Недоліки n-грамних моделей:**

* Просторові обмеження: Чим більше значення n, тим більший простір для зберігання ймовірностей. Це може стати проблемою з огляду на пам'ять і обчислювальні ресурси.
* Обмеження контексту: Моделі враховують лише n-1 попередніх елементів, що може призвести до втрати важливої інформації, якщо контекст вищий за n-1.
* Рідкісні комбінації: Для рідко зустрічаються n-грам часто буде проблема, оскільки ймовірності таких комбінацій можуть бути дуже низькими або нульовими.

**2. Моделі на основі прихованих марковських процесів (HMMs)**

**Приховані марковські процеси (HMMs)** — це статистичні моделі, що використовуються для аналізу послідовностей спостережень, де система знаходиться в **скритому стані**, і ці стани змінюються за певними ймовірностями.

Розглянемо приклад мовної моделі, де ми маємо **приховані стани** (наприклад, частини мови: іменник, дієслово, прикметник), а **спостереженнями** є реальні слова в тексті.

* **Приховані стани**: (Іменник, дієслово, прикметник)
* **Спостереження**: (слова тексту: "котик", "біг", "швидкий")
* **Ймовірності переходів** між частинами мови:
	+ P(дієслово | іменник) = 0.6
	+ P(прикметник | дієслово) = 0.4

Якщо ми маємо спостереження: **"котик біг швидко"**, ми можемо за допомогою HMM обчислити ймовірність того, що ці слова відповідають певним частинам мови та побудувати ймовірнісну модель для тексту.

**Обмеження марковської гіпотези**:

* модель враховує лише поточний стан, що може не вистачати для складних залежностей у даних.
* не завжди можна точно визначити кількість станів, особливо в складних задачах.

**3. Рекурентні нейронні мережі (RNNs) та LSTMs**

RNNs мають унікальну архітектуру, яка дозволяє зберігати інформацію про попередні етапи послідовності завдяки наявності циклічних зв'язків між нейронами. В RNN кожен нейрон отримує на вході не тільки поточне значення, але й значення з попереднього етапу (через рекурентні зв'язки). Це дозволяє мережі "запам'ятовувати" інформацію про попередні стани.

LSTM (Long Short-Term Memory) — це вдосконалений тип RNN, розроблений для подолання проблеми затухаючого градієнта та слабкої здатності зберігати довготривалі залежності в послідовностях.

Проблема звичайних RNN полягає в тому, що їм важко запам'ятовувати інформацію на довгий проміжок часу. Уявіть довге речення або абзац. Інформація про слова, які були на початку, може "забуватися" до того моменту, як мережа дійде до кінця. Це називається проблемою "затухаючого градієнта".

LSTM - це як RNN з "покращеною пам'яттю". Вона має спеціальні механізми, які дозволяють їй:

Вирішувати, яку інформацію потрібно запам'ятати на довгий термін.

Вирішувати, коли потрібно "забути" стару інформацію, яка більше не є важливою.

Вирішувати, яку нову інформацію потрібно додати до "пам'яті".

Ці механізми реалізуються за допомогою спеціальних "вентилів" (gates) всередині LSTM-нейрона.

Простий приклад LSTM: Уявіть, що ми аналізуємо відгук про фільм: "Фільм був дуже цікавим, особливо гра акторів. Але кінець був трохи передбачуваним."

LSTM обробляє "Фільм був дуже цікавим". Вона запам'ятовує, що відгук загалом позитивний.

Потім вона обробляє "особливо гра акторів". Вона додає до своєї "пам'яті" інформацію про те, що гра акторів була відзначена.

Коли вона доходить до "Але кінець був трохи передбачуваним", слово "Але" сигналізує про зміну настрою. LSTM може зберегти інформацію про позитивні аспекти, але також запам'ятати і негативну оцінку кінця.

Завдяки своїм "вентилям", LSTM може довше зберігати важливу інформацію (наприклад, загальну позитивну оцінку фільму) і при цьому обробляти нову інформацію (негативну оцінку кінця), не "забуваючи" попередній контекст.

2.**Трансформери – революція в NLP**

**Трансформери** — це архітектура нейронних мереж, яка була запропонована в статті "Attention is All You Need" (2017) авторами **Ashish Vaswani** та іншими. Трансформери стали революцією в обробці природної мови (NLP) і отримали широке застосування завдяки своїм перевагам у порівнянні з рекурентними нейронними мережами (RNN) і LSTM.

Основною інновацією трансформерів є використання механізму **self-attention** (самоувага), який дозволяє моделі обробляти усі елементи вхідної послідовності одночасно, замість того щоб обробляти їх по черзі, як це робиться в RNN і LSTM. Це значно прискорює обчислення та дозволяє моделі краще справлятися з довготривалими залежностями в текстах.

Трансформери складаються з двох основних компонентів:

1. **Енкодер (Encoder)**
2. **Декодер (Decoder)**

Обидва компоненти включають в себе стеки однакових блоків, що містять кілька важливих етапів, таких як механізм самоуваги та лінійні шари.

Енкодер приймає вхідну послідовність (наприклад, текст) і перетворює її в набір числових векторів, що містять інформацію про кожне слово в контексті всього тексту. Він складається з таких основних частин:

* **Механізм самоуваги (Self-Attention)**: Самоувага дозволяє кожному елементу входу взаємодіяти з усіма іншими елементами в послідовності, оцінюючи їх значення і важливість для кожного елемента. Наприклад, у реченні "The cat sat on the mat" трансформер може визначити, що слово "sat" пов'язане з "cat", і використовувати цю інформацію для кращого розуміння контексту.
* **Нормалізація та Feed-Forward Neural Network (FFNN)**: Після механізму самоуваги проходить етап нормалізації і Feed-Forward шари які обробляють інформацію кожного слова незалежно, але вже з урахуванням контексту, отриманого від механізму самоуваги, і витягує більш складні ознаки.

Декодер працює аналогічно енкодеру, але з додатковим етапом взаємодії з виходами енкодера. Він генерує передбачення на основі вихідної послідовності, що вже була оброблена. Основні етапи декодера:

* **Механізм самоуваги**: Як і в енкодері, декодер використовує самоувагу, щоб враховувати зв'язки між елементами виходу.
* **Механізм уваги (Attention Mechanism)**: Декодер також використовує механізм уваги для взаємодії з виходами енкодера, щоб орієнтуватися на відповідні частини вхідної послідовності.

Самоувага є основою трансформера, дозволяючи кожному слову в послідовності "поважати" всі інші слова. Вона дозволяє моделі навчатися взаємозв'язку між елементами, незалежно від того, наскільки вони віддалені одне від одного в тексті.

**Приклад роботи механізму:**

**Уявіть, що кожне слово в реченні хоче "запитати" у інших слів, наскільки вони важливі для його розуміння. Для цього кожне слово перетворюється на три різні "представлення":**

**Запит (Query - Q): Це як "питання", яке слово ставить іншим словам.**

**Ключ (Key - K): Це як "ярлик" або "ключове слово", яке описує зміст кожного слова.**

**Значення (Value - V): Це сама "інформація", яку несе слово.**

**Давайте уявимо, що для кожного слова ми отримали такі спрощені вектори (насправді вони будуть набагато більшими):**

|  |  |  |
| --- | --- | --- |
| **Слово** | **Запит (Q)** | **Ключ (K)** |
| **Кіт** | **[0.8, 0.1]** | **[0.2, 0.9]** |
| **спить** | **[0.3, 0.7]** | **[0.8, 0.3]** |
| **на** | **[0.5, 0.5]** | **[0.4, 0.6]** |
| **килимку** | **[0.9, 0.2]** | **[0.1, 0.8]** |

**Ці вектори отримуються шляхом множення ембедингів (числових представлень) слів на різні матриці ваг під час навчання моделі.**

**2. Обчислення уваги:**

**Для кожного слова ми обчислюємо, наскільки воно "уважне" до всіх інших слів. Це робиться шляхом порівняння його запиту (Q) з ключами (K) всіх інших слів. Чим більше схожі запит одного слова і ключ іншого, тим більша їх "увага" один до одного.**

**Зазвичай використовується скалярний добуток між вектором запиту одного слова та вектором ключа іншого слова. Результат ділиться на корінь з розмірності векторів (щоб стабілізувати градієнти).**

**Розглянемо, як обчислюється увага слова "спить" до всіх інших слів:**

**Увага ("спить", "Кіт"): Скалярний добуток Q("спить") і K("Кіт") = (0.3 \* 0.2) + (0.7 \* 0.9) = 0.06 + 0.63 = 0.69.**

**Увага ("спить", "спить"): Скалярний добуток Q("спить") і K("спить") = (0.3 \* 0.8) + (0.7 \* 0.3) = 0.24 + 0.21 = 0.45.**

**Увага ("спить", "на"): Скалярний добуток Q("спить") і K("на") = (0.3 \* 0.4) + (0.7 \* 0.6) = 0.12 + 0.42 = 0.54.**

**Увага ("спить", "килимку"): Скалярний добуток Q("спить") і K("килимку") = (0.3 \* 0.1) + (0.7 \* 0.8) = 0.03 + 0.56 = 0.59.**

**Після цього отримані значення зазвичай нормалізуються за допомогою функції Softmax, щоб отримати ймовірності (сума всіх ваг уваги для одного слова дорівнює 1). Уявімо, що після Softmax ми отримали такі ваги уваги для слова "спить":**

|  |  |
| --- | --- |
| **До слова:** | **Вага уваги:** |
| **Кіт** | **0.35** |
| **спить** | **0.25** |
| **на** | **0.20** |

**Це означає, що для розуміння слова "спить" найбільшу увагу слід приділити слову "Кіт" (вага 0.35), трохи менше - самому слову "спить" (0.25), і ще менше - словам "на" та "килимку" (по 0.20).**

**3. Зважене сумування значень:**

**На останньому кроці для кожного слова обчислюється його нове представлення шляхом зваженого сумування векторів значень (V) всіх інших слів. Вагами для цього сумування є обчислені на попередньому кроці ваги уваги.**

**Для слова "спить" його нове представлення буде обчислено так:**

**Нове представлення("спить") =**

**(Вага уваги("спить", "Кіт") \* V("Кіт")) +**

**(Вага уваги("спить", "спить") \* V("спить")) +**

**(Вага уваги("спить", "на") \* V("на")) +**

**(Вага уваги("спить", "килимку") \* V("килимку"))**

**Підставляючи наші умовні вектори та ваги:**

**Нове представлення("спить") =**

**(0.35 \* [1.0, 0.5]) +**

**(0.25 \* [0.2, 0.8]) +**

**(0.20 \* [0.7, 0.1]) +**

**(0.20 \* [0.4, 0.9]) =**

**[0.35, 0.175] + [0.05, 0.2] + [0.14, 0.02] + [0.08, 0.18] =**

**[(0.35 + 0.05 + 0.14 + 0.08), (0.175 + 0.2 + 0.02 + 0.18)] =**

**[0.62, 0.575]**

**Таким чином, нове представлення слова "спить" тепер містить інформацію, зважену відповідно до його зв'язку з іншими словами в реченні. Особливо сильно вплинув вектор значення слова "Кіт", оскільки вага уваги до нього була найбільшою.**

**Цей процес повторюється для кожного слова вхідної послідовності, і в результаті кожне слово отримує нове, збагачене контекстом представлення. Це дозволяє трансформеру розуміти зв'язки та залежності між словами в реченні і краще обробляти складні мовні завдання.**

**Паралельне оброблення**

Оскільки трансформер використовує механізм самоуваги, який дозволяє всім елементам входу взаємодіяти одночасно, трансформери можуть обробляти усі елементи послідовності паралельно, що значно прискорює навчання і інференс у порівнянні з RNN або LSTM, де послідовність обробляється по черзі.

**Мультиголова увага (Multi-Head Attention)**

Для кращої обробки різних аспектів взаємозв'язків між елементами, трансформери використовують **мультиголову увагу**. Це означає, що замість одного механізму уваги модель використовує кілька "голів" уваги, що дозволяє моделі вивчати різні види взаємозв'язків одночасно. Потім результати всіх голів об'єднуються в один вектор.

**Позиційне кодування (Positional Encoding)**

Оскільки трансформер обробляє всю послідовність одночасно, він не має внутрішньої структури, яка б враховувала порядок слів, як це роблять RNN або LSTM. Для вирішення цієї проблеми додається **позиційне кодування** до кожного вектора вхідних даних, щоб модель могла враховувати порядок елементів у послідовності.

Позиційне кодування є вектором, який додається до вхідного вектору кожного слова, і він містить інформацію про його позицію в послідовності.

**Переваги трансформерів:**

1. **Швидкість**: Завдяки паралельній обробці даних трансформери значно швидші за RNN і LSTM при тренуванні на великих наборах даних.
2. **Гнучкість**: Механізм самоуваги дозволяє трансформерам обробляти віддалені залежності у текстах і зберігати контекст, незалежно від відстані між словами.
3. **Масштабованість**: Трансформери добре працюють на великих обсягах даних і можуть бути використані в таких великих моделях, як GPT, BERT, T5 тощо.

**Недоліки трансформерів:**

1. **Висока обчислювальна складність**: Оскільки трансформери обчислюють пару увагу для кожної пари елементів в послідовності, їх складність може бути квадратичною O(n^2), що може бути проблемою для дуже довгих послідовностей.
2. **Потреба в великих обчислювальних ресурсах**: Трансформери вимагають великої кількості пам'яті та потужних обчислювальних ресурсів, що може бути обмеженням для деяких завдань.

**Приклади використання трансформерів:**

1. **Моделі мови**: Наприклад, BERT, GPT, T5 — трансформери, які використовуються для широкого спектра завдань обробки природної мови (переклад, сумаризація, запитання-відповіді, тощо).
2. **Переклад текстів**: Трансформери були основою для нових досягнень у машинному перекладі, перевершуючи попередні методи, такі як LSTM і RNN.
3. **Творче генерування текстів**: Моделі на основі трансформерів (наприклад, GPT) можуть генерувати текст, схожий на людський, що використовується в чат-ботах, автоматичному створенні контенту тощо.

**3.Навчання великих мовних моделей (LLMs)**

Навчання великих мовних моделей (LLMs), таких як GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers) або T5 (Text-to-Text Transfer Transformer), є складним і ресурсномістким процесом, що включає кілька етапів. Нижче ми розглянемо основні кроки та принципи, які використовуються при навчанні таких моделей.

#### **1. Попереднє навчання (Pretraining)**

Попереднє навчання є першим етапом у процесі навчання великої мовної моделі. На цьому етапі модель навчається на великій кількості текстових даних, без конкретного завдання, щоб освоїти загальні принципи роботи з мовою. Це дозволяє моделі захопити синтаксис, семантику, контекст і інші особливості мови.

**Методи попереднього навчання**:

* **Маскування слів** (як у BERT): Модель навчається передбачати масковані або пропущені слова в тексті на основі контексту. Це дозволяє моделі вивчати контекстну залежність між словами.
* **Моделювання наступного слова** (як у GPT): Модель навчається передбачати наступне слово в послідовності, враховуючи попередні слова. Це дозволяє моделі генерувати тексти або завершувати частину тексту.
* **Моделювання за допомогою денсних представлень** (як у T5): Тут модель може бути навчена на завданні перекладу або іншому специфічному завданні, де текст перетворюється з одного формату в інший.

#### **2. Трансферне навчання (Transfer Learning)**

Після попереднього навчання велика мовна модель вже має загальні знання про мову і контекст. Трансферне навчання полягає в тому, щоб перенести ці знання на конкретні завдання, такі як переклад текстів, класифікація, питання-відповідь або аналіз настроїв.

**Етапи трансферного навчання**:

* **Файнтюнинг (Fine-Tuning)**: Це процес донавчання моделі на конкретному наборі даних для конкретного завдання. Наприклад, для задачі класифікації тексту модель буде навчена на спеціалізованих даних, щоб правильно класифікувати текстові фрагменти.
* **Адаптація до специфічних доменів**: Файнтюнинг може включати адаптацію моделі до специфічних доменів, наприклад, медичних текстів або юридичних документів.

#### **3. Механізми навчання**

Під час навчання великих мовних моделей використовуються деякі важливі механізми та техніки:

* **Обробка великих даних**: Для навчання LLM потрібні величезні обсяги текстових даних. Це можуть бути книги, статті, веб-сторінки, соціальні мережі, технічна документація тощо. Для ефективного навчання модель потребує десятків, а іноді й сотень мільярдів слів.
* **Паралельне навчання**: Навчання LLM часто розподіляється між кількома обчислювальними пристроями (наприклад, GPU або TPU), що дозволяє обробляти великі обсяги даних і забезпечує високі швидкості навчання.
* **Адаптивне навчання**: Логіка навчання часто включає використання методів адаптивного навчання, таких як **Adam** або **LAMB**, для коригування коефіцієнтів навчання в залежності від величини градієнта.

#### **4. Оцінка і тестування моделі**

Після завершення процесу навчання модель оцінюється на різноманітних тестових наборах, щоб перевірити її здатність виконувати різні завдання.

**Метрики для оцінки великих мовних моделей**:

* **Perplexity (Перплексія)**: Це основна метрика для оцінки моделей, які генерують текст. Вона вимірює, наскільки модель невизначена в своєму передбаченні. Чим менша перплексія, тим краще модель передбачає наступне слово.
* **Точність (Accuracy)**: Для задач, таких як класифікація тексту або відповіді на питання, використовується точність як метрика. Вона вимірює частку правильних передбачень серед всіх передбачень.
* **F1-Score**: Це середнє гармонійне між точністю та відзивом, яке широко використовується в задачах класифікації, щоб оцінити якість моделі.
1. **Деякі приклади LLM**

### 1. **GPT (Generative Pretrained Transformer)**

* **Розробник**: OpenAI
* **Опис**: GPT є трансформерною мовною моделлю, яка працює на основі автогресивного навчання. Модель передбачає наступне слово в послідовності на основі попередніх слів. GPT здатна генерувати текст, який є осмисленим і відповідає заданому контексту.
* **Відомі версії**:
	+ **GPT-3**: Містить 175 мільярдів параметрів, що зробило її однією з найбільших моделей на момент її виходу.
	+ **GPT-4**: Ще більша та точніша версія, здатна вирішувати складніші завдання, включаючи креативне написання, програмування, і навіть виконання задачі, що вимагають глибшого розуміння світу.
* **Застосування**: Генерація тексту, чат-боти, питання-відповіді, переклад, підсумовування.

### 2. **BERT (Bidirectional Encoder Representations from Transformers)**

* **Розробник**: Google AI
* **Опис**: BERT є однією з найбільш впливових моделей для задач, пов'язаних з розумінням тексту. На відміну від GPT, BERT використовує двостороннє навчання, що дозволяє моделі краще розуміти контекст слів з обох боків. BERT здатний розпізнавати взаємозв'язки між словами на рівні фраз та речень.
* **Застосування**: Класифікація текстів, відповіді на питання, сентимент-аналіз, іменовані сутності.
* **Особливості**: Завдяки двосторонньому навчання модель здатна краще враховувати контекст, що підвищує точність для завдань, де важливо розуміння значення слів у контексті.

### 3. **T5 (Text-to-Text Transfer Transformer)**

* **Розробник**: Google AI
* **Опис**: T5 є універсальною моделлю, яка перетворює всі завдання обробки природної мови на задачу "текст до тексту". Наприклад, питання може бути перетворене на форму запиту, а текст — на відповідь. Це дозволяє моделі застосовувати один і той самий підхід до різних типів завдань (переклад, класифікація, підсумовування тощо).
* **Застосування**: Переклад тексту, підсумовування, генерація тексту, питання-відповіді, класифікація.
* **Особливості**: Модель здатна ефективно вирішувати різні задачі, завдяки універсальності архітектури "текст до тексту".

### 4. **LLaMA (Large Language Model Meta AI)**

* **Розробник**: Meta (раніше Facebook)
* **Опис**: LLaMA — це серія моделей, орієнтованих на ефективність і масштабованість, яка була розроблена для того, щоб забезпечити високі результати з меншою кількістю параметрів порівняно з іншими великими моделями. LLaMA надає більшу доступність для дослідників завдяки своєму відкритому коду і тренуванню на великих наборах даних.
* **Застосування**: Генерація тексту, креативні завдання, дослідження в галузі штучного інтелекту.
* **Особливості**: Моделі LLaMA вважаються більш ефективними і оптимізованими порівняно з GPT або іншими великими моделями, що дозволяє використовувати їх на менш потужних обчислювальних системах.

### 5. **Gemini (раніше Bard)**

* **Розробник**: Google DeepMind
* **Опис**: Gemini є новою серією мовних моделей від Google, яка прийшла на зміну Bard. Вона має кілька версій, що спеціалізуються на вирішенні різноманітних задач в обробці природної мови, включаючи відповіді на питання, генерацію тексту, підсумовування та інше. Моделі Gemini відрізняються високою точністю в рішеннях складних запитів і можуть інтегрувати різноманітні джерела даних для поліпшення результатів.
* **Застосування**: Генерація тексту, інтерактивні чат-боти, відповіді на питання, аналітика даних.
* **Особливості**: Gemini є еволюцією Bard і спирається на новіші досягнення в області AI, включаючи інтеграцію з іншими технологіями Google.

Сучасні LLMs, такі як GPT, BERT, T5, LLaMA та Gemini, різняться своїми архітектурами та підходами до обробки тексту. GPT орієнтований на генерацію, BERT на розуміння контексту, T5 на універсальність задач, LLaMA на ефективність та масштабованість, а Gemini від Google є новим кроком у розвитку інтелектуальних систем. Кожна з цих моделей має свої особливості і застосування, що робить їх дуже потужними інструментами для різних завдань в області обробки природної мови.

### **Обмеження та виклики великих мовних моделей (LLMs)**

Хоча великі мовні моделі, такі як GPT, BERT, T5 та інші, мають потужний потенціал для вирішення широкого спектру завдань, вони стикаються з кількома важливими обмеженнями та викликами. Ось основні з них:

### **1. Потреба у великих обчислювальних ресурсах**

* **Опис**: Тренування великих мовних моделей вимагає значних обчислювальних ресурсів. Для навчання моделей на мільярди параметрів використовуються тисячі графічних процесорів (GPU) та терабайти даних. Це стає великою перешкодою для дрібних компаній та академічних дослідників, які не мають доступу до таких потужностей.
* **Наслідки**:
	+ **Висока вартість**: Тренування таких моделей потребує величезних фінансових вкладень, що робить їх доступними переважно великим корпораціям (наприклад, OpenAI, Google).
	+ **Екологічний вплив**: Висока енергетична споживаність моделей призводить до значних викидів CO2, що є серйозним екологічним питанням.
* **Рішення**: Для зменшення витрат на обчислення розробляються більш ефективні алгоритми, а також створюються моделі меншого розміру, які все ще досягають хороших результатів. Використання хмарних обчислень і спеціалізованих апаратних засобів (наприклад, TPU від Google) також допомагає вирішити ці проблеми.

### **2. Проблеми упередженості (bias) та етичні питання**

* **Опис**: Моделі, навчальні на великих обсягах текстових даних, можуть відтворювати соціальні упередження, які присутні в цих даних. Це може включати расову, гендерну або культурну дискримінацію, що ставить під загрозу етичність їх використання в реальному світі.
* **Приклади**:
	+ Генерація расистських або сексистських висловлювань.
	+ Проблеми в системах автоматизованого прийняття рішень, де упередженість моделі може призводити до несправедливих результатів.
* **Рішення**: Виробники великих мовних моделей активно працюють над зменшенням упередженості через застосування різноманітних технік (наприклад, балансування даних, покращення фільтрації контенту, контролювання поведінки моделей). Також важливою є співпраця з етичними експертами для оцінки і виявлення упереджень у тренувальних наборах даних.

### **3. Проблема галюцинацій у LLMs**

* **Опис**: Галюцинації — це явище, коли мовні моделі генерують неправдиву або недостовірну інформацію, навіть якщо вона виглядає правдоподібною. Це може стати серйозною проблемою в таких застосуваннях, як автоматичні чат-боти, пошукові системи або інші сервіси, де точність є критично важливою.
* **Приклади**:
	+ Модель може створити факти, які не відповідають реальності, наприклад, вигадати статистичні дані або імена людей, які не існують.
	+ Генерація неправдивих відомостей або відповідей на складні питання без достовірної підстави.
* **Причини**:
	+ Моделі тренуються на великих текстових корпусах, де є велика кількість неточних або суперечливих даних.
	+ Моделі можуть генерувати текст, орієнтуючись на шаблони, а не на фактичну точність.
* **Рішення**:
	+ Використання додаткових методів перевірки достовірності (наприклад, інтеграція з базами даних або зовнішніми джерелами).
	+ Покращення архітектурних підходів, щоб моделі могли краще фільтрувати або оцінювати достовірність своєї інформації.
	+ Впровадження механізмів валідації та фільтрації результатів, щоб попередити генерацію хибних відповідей.

### **4. Інтерпретованість та контроль поведінки моделей**

* **Опис**: Великі мовні моделі, через свою складність і величезну кількість параметрів, є "чорними ящиками". Це означає, що вони можуть приймати рішення або генерувати текст, але важко зрозуміти, чому модель прийняла певне рішення або як саме була сформульована відповідь.
* **Наслідки**:
	+ **Неясність**: Важко зрозуміти, чому модель прийняла певний висновок, що обмежує можливості для аудиту та виявлення помилок.
	+ **Відповідальність**: Коли модель генерує непотрібні або небезпечні результати, відсутність розуміння механізмів прийняття рішень ускладнює визначення відповідальності.
* **Рішення**:
	+ Розробка інтерпретованих моделей, де можна зрозуміти, які саме фактори впливають на прийняття рішення.
	+ Використання методів, таких як LIME (Local Interpretable Model-agnostic Explanations) або SHAP (Shapley Additive Explanations), для пояснення рішень моделей.
	+ Впровадження механізмів контролю поведінки моделей, щоб забезпечити більшу передбачуваність та прозорість у їхній роботі.