Wprowadzenie do dużych modeli językowych

Duże modele językowe (LLM) to generatywne modele sztucznej inteligencji, które potrafią rozumieć i generować tekst przypominający tekst ludzki na podstawie danych wejściowych. LLM stanowią podstawę wielu zadań przetwarzania języka naturalnego (NLP), takich jak wyszukiwanie, zamiana mowy na tekst, analiza sentymentu, streszczanie tekstu i wiele innych. Ponadto są to uniwersalne modele językowe, które są wstępnie wytrenowane i mogą być precyzyjnie dostrajane do konkretnych zadań i celów.

Tu zgłębiamy fascynujący świat LLM i ich transformacyjny wpływ na sztuczną inteligencję (AI). Jako znaczący postęp w dziedzinie AI, LLM wykazały niezwykłe możliwości w zakresie rozumienia i generowania tekstu zbliżonego do ludzkiego, umożliwiając tym samym liczne zastosowania w różnych branżach. Zagłębiamy się w krytyczne przypadki użycia LLM, różne typy LLM oraz koncepcję modeli fundamentalnych, które zrewolucjonizowały rozwój AI. Sekcja omawia podstawowe koncepcje LLM, takie jak prompty, inżynieria promptów, osadzenia, tokeny, parametry modelu, okna kontekstowe, architektura transformatorów i zachowania emergentne. Na koniec porównujemy open source'owe i komercyjne LLM, podkreślając ich zalety i wady. Pod koniec będziesz mieć kompleksowe zrozumienie LLM i ich wpływu na zastosowania i badania nad AI. LLM opierają się na modelach fundamentalnych, dlatego zaczniemy od omówienia, czym są te modele, zanim przejdziemy do bardziej szczegółowego omówienia LLM.

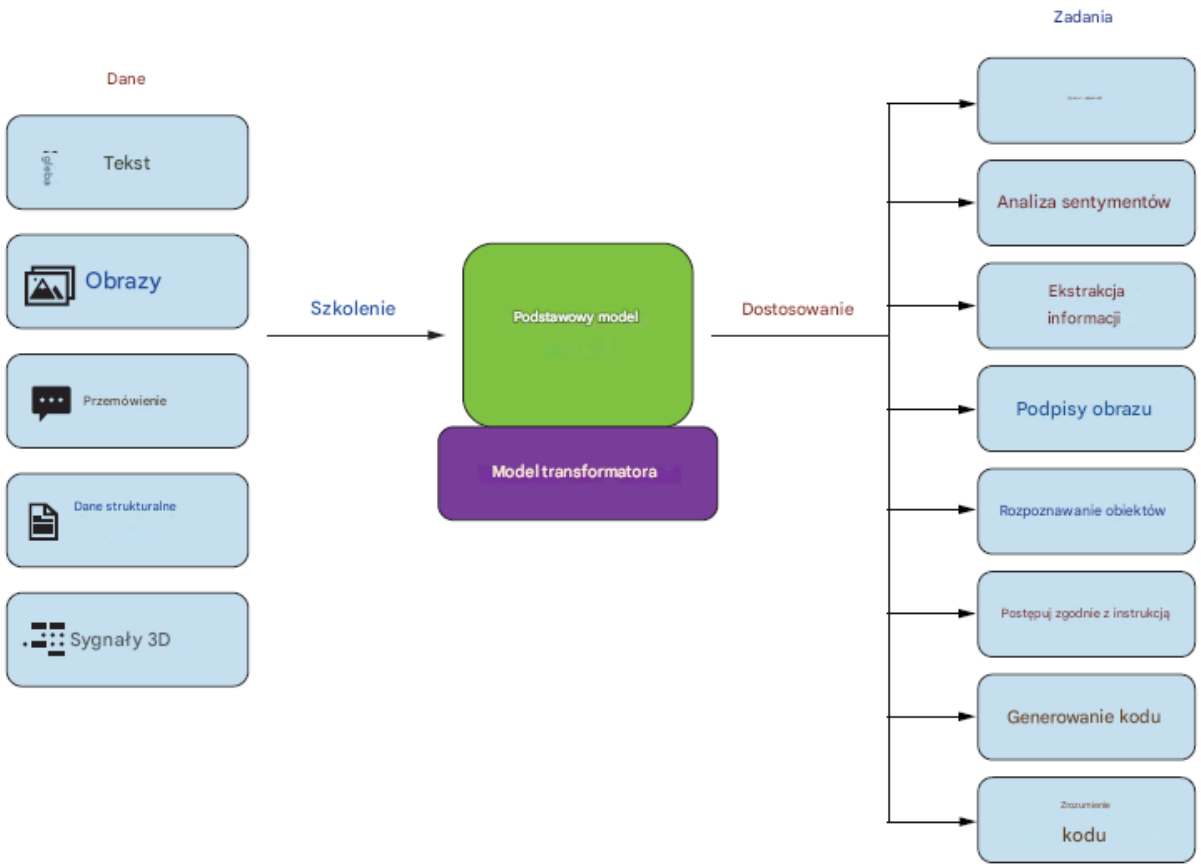
Przegląd modeli fundamentalnych

Wprowadzone przez naukowców ze Stanford w 2021 roku, modele fundamentalne znacząco zmieniły konstrukcję systemów sztucznej inteligencji. Odchodzą one od modeli ukierunkowanych na konkretne zadania, przechodząc w szersze, bardziej adaptacyjne modele trenowane na dużych wolumenach danych. Modele te doskonale sprawdzają się w różnorodnych zadaniach związanych z językiem naturalnym, takich jak tłumaczenie maszynowe i odpowiadanie na pytania, ponieważ uczą się ogólnych reprezentacji języka z rozległych zbiorów danych tekstowych i kodowych. Reprezentacje te mogą być następnie wykorzystywane do wykonywania różnych zadań, nawet tych, w których nie były bezpośrednio trenowane, jak pokazano na rysunku. Mówiąc bardziej technicznie, modele fundamentalne wykorzystują sprawdzone techniki uczenia maszynowego, takie jak uczenie samonadzorowane i uczenie transferowe, umożliwiając im zastosowanie zdobytej wiedzy w różnych zadaniach. Opracowane za pomocą głębokiego uczenia, modele te wykorzystują wielowarstwowe sztuczne sieci neuronowe do rozumienia złożonych wzorców danych; stąd ich biegłość w pracy z danymi nieustrukturyzowanymi, takimi jak obrazy, dźwięk i tekst. Dotyczy to również sygnałów 3D — danych reprezentujących atrybuty 3D, które rejestrują wymiary przestrzenne i głębię, takich jak chmury punktów 3D z czujników LiDAR, obrazowanie medyczne 3D, takie jak tomografia komputerowa, lub modele 3D wykorzystywane w grafice komputerowej i symulacjach. Można je wykorzystać do tworzenia prognoz w oparciu o dane 3D w takich zadaniach, jak rozpoznawanie obiektów, rozumienie scen i nawigacja w robotyce i pojazdach autonomicznych.

UWAGA: Transfer uczenia to technika uczenia maszynowego, w której model opracowany dla jednego zadania jest ponownie wykorzystywany jako punkt wyjścia dla podobnego zadania. Zamiast zaczynać od zera, wykorzystujemy wiedzę z poprzedniego zadania, aby lepiej wykonać nowe. To tak, jakby wykorzystać wiedzę z poprzedniej pracy, aby osiągnąć sukces w nowej, ale powiązanej pracy.

Generatywna sztuczna inteligencja i modele fundamentalne są ze sobą ściśle powiązane. Jak wspomniano, modele fundamentalne, trenowane na ogromnych zbiorach danych, można dostosować do wykonywania różnych zadań; ta właściwość sprawia, że są one szczególnie odpowiednie dla generatywnej sztucznej inteligencji i umożliwia tworzenie nowych treści. Szeroka baza wiedzy tych

modeli umożliwia efektywne uczenie transferowe, które może być wykorzystane do generowania nowych, adekwatnych pod względem tekstowym treści w różnych dziedzinach. Reprezentują one ujednolicone podejście, w którym pojedynczy model może generować różnorodne wyniki, oferując najnowocześniejszą wydajność dzięki rozbudowanemu procesowi uczenia. Bez modeli fundamentalnych, stanowiących podstawę, nie byłoby generatywnych modeli sztucznej inteligencji.



Oto kilka przykładów popularnych modeli bazowych:

* Rodzina GPT (Generative Pre-trained Transformer) to rodzina modeli NLP opracowana przez OpenAI. Jest to obszerny model językowy trenowany na ogromnym zbiorze danych tekstowych i kodowych, co umożliwia generowanie tekstu, tłumaczenie języków, pisanie kreatywnych treści i udzielanie informacyjnych odpowiedzi na pytania. GPT-4, najnowsza wersja w momencie pisania tego tekstu, jest również modelem multimodalnym — może zarządzać zarówno językiem, jak i obrazami.

* Codex to obszerny model językowy trenowany specjalnie na kodzie, używany do wspomagania generowania kodu. Obsługuje ponad tuzin języków programowania, w tym niektóre z najpopularniejszych, takie jak C#, Java, Python, Java Script, SQL, Go, PHP i Shell.

* Claude to absolwent LLM stworzony przez startup Anthropic. Podobnie jak ChatGPT OpenAI, przewiduje on kolejny token w sekwencji po otrzymaniu określonego monitu i może generować tekst, pisać kod, podsumowywać i wnioskować.

* BERT (Bidirectional Encoder Representations from Transformers) to model przetwarzania języka naturalnego opracowany przez Google. Jest to model dwukierunkowy, co oznacza, że może przetwarzać tekst w obu kierunkach, od lewej do prawej i od prawej do lewej. Ta funkcja ułatwia zrozumienie kontekstu słów i fraz.

* PaLM (Pathway Language Model) i jego następca PaLM2 to rozbudowane multimodalne modele językowe opracowane przez Google. Model multimodalny może przetwarzać tekst, kod i obrazy jednocześnie, co umożliwia wykonywanie szerszego zakresu zadań w tych modalnościach w porównaniu z tradycyjnymi modelami językowymi działającymi tylko w jednej modalności.

* Gemini to najnowszy model sztucznej inteligencji Google, zdolny do rozumienia tekstu, obrazów, filmów i dźwięku. Jest to model multimodalny, który opisuje się jako zdolny do wykonywania złożonych zadań z matematyki, fizyki i innych dziedzin, a także do rozumienia i generowania wysokiej jakości kodu w różnych językach programowania. Gemini został od podstaw zaprojektowany jako multimodalny, co oznacza, że może generalizować i płynnie rozumieć, operować na różnych typach informacji oraz łączyć je. Jest to również nowa nazwa zbiorcza wszystkich narzędzi sztucznej inteligencji firmy Google, zastępująca Google Bard i Duet AI, i jest uważana za następcę modelu PaLM.

Po wytrenowaniu modelu podstawowego można go dostosować do szerokiego zakresu zadań dalszych poprzez precyzyjne dostrojenie jego parametrów. Dostrajanie polega na dostosowywaniu parametrów modelu w celu optymalizacji go pod kątem konkretnego zadania. Można to zrobić przy użyciu niewielkiej ilości oznaczonych danych. Dostrajając te modele pod kątem konkretnych zadań lub dziedzin, wykorzystujemy ich ogólną wiedzę o języku i uzupełniamy ją o wiedzę specyficzną dla danego zadania. Zalety tego podejścia obejmują efektywność czasową i zasobową, a także niezwykłą wszechstronność. Model można również dostosować za pomocą inżynierii Prompt, którą omówimy w dalszej części tego rozdziału. Teraz, gdy wiemy więcej o modelach podstawowych, przyjrzyjmy się modelom LLM.

Omówienie modeli LLM

Modele LLM stanowią znaczący postęp w dziedzinie sztucznej inteligencji. Są one trenowane na ogromnej ilości danych tekstowych, takich jak książki, artykuły i strony internetowe, aby uczyć się wzorców w języku ludzkim. Są również trudne w opracowaniu i utrzymaniu, ponieważ wymagają dużych zasobów danych, obliczeniowych i inżynieryjnych. ChatGPT firmy OpenAI jest przykładem modelu LLM — generuje on tekst imitujący tekst ludzki, przewidując prawdopodobieństwo wystąpienia słowa, biorąc pod uwagę słowa już użyte w tekście. Model uczy się generować spójne i kontekstowo istotne zdania, dostosowując swoje parametry wewnętrzne, aby zminimalizować różnicę między przewidywaniami a rzeczywistymi wynikami w danych treningowych. Podczas generowania tekstu model wybiera słowo o najwyższym prawdopodobieństwie jako kolejny wynik, a następnie powtarza proces dla kolejnego słowa.

Modele LLM to fundamentalne modele dostosowane do zadań przetwarzania języka naturalnego i generowania języka. Te modele LLM są uniwersalne i mogą obsługiwać zadania bez danych treningowych specyficznych dla danego zadania. Jak pokrótce opisano w poprzednim rozdziale, po otrzymaniu odpowiedniego polecenia mogą odpowiadać na pytania, pisać eseje, streszczać teksty, tłumaczyć języki, a nawet generować kod. Modele LLM można zastosować w wielu aplikacjach w różnych branżach - od podsumowań po klasyfikację, chatboty pytań i odpowiedzi, generowanie treści, analizę danych, ekstrakcję encji i wiele innych. Zanim przejdziemy do szczegółów modeli LLM, przyjrzyjmy się architekturze Transformer, która umożliwia powstanie tych podstawowych modeli.

Architektura transformatorów

Transformatory stanowią fundament modeli fundamentalnych i odpowiadają za ich niezwykłe możliwości rozumienia języka. Model transformatora został po raz pierwszy przedstawiony w artykule „Attention Is All You Need" autorstwa Vaswaniego i in. w 2017 roku . Od tego czasu modele oparte na transformatorach stały się najnowocześniejszym rozwiązaniem w wielu zadaniach. GPT i BERT są

przykładami modeli opartych na transformatorach, a „T” w GPT oznacza „Transformers”. W swojej istocie transformatory wykorzystują mechanizm znany jako uwaga (a konkretnie samouwaga), który pozwala modelowi uwzględniać cały kontekst zdania, uwzględniając wszystkie słowa jednocześnie, zamiast przetwarzać zdanie słowo po słowie. To podejście jest bardziej wydajne i może poprawić wyniki wielu zadań NLP. Zaletą tego podejścia jest to, że uwzględnia zależności niezależnie od ich położenia w tekście, co jest kluczowym czynnikiem w rozumieniu języka. Jest to kluczowe w przypadku zadań takich jak tłumaczenie maszynowe i streszczanie tekstu, gdzie znaczenie zdania może zależeć od terminów oddalonych od siebie o kilka słów. Sieci transformatorowe mogą paralelizować swoje obliczenia, co sprawia, że są znacznie szybsze w trenowaniu niż inne typy sieci neuronowych. Ten mechanizm pozwala modelowi skupić się na najistotniejszych częściach danych wejściowych zadania. W kontekście generatywnej sztucznej inteligencji, model transformatorowy pobierałby dane wejściowe (takie jak podpowiedź) i generował dane wyjściowe (takie jak następne słowo lub dokończenie zdania), oceniając znaczenie każdej części danych wejściowych w generowaniu danych wyjściowych. Na przykład w zdaniu „Kot usiadł na...” model transformatorowy prawdopodobnie nadałby dużą wagę słowu „kot” przy określaniu, że prawdopodobnym następnym słowem może być „mata”. Modele te wykazują właściwości generatywne, przewidując następny element w sekwencji — następne słowo w zdaniu lub następną nutę w melodii. Zbadamy to dokładniej w następnym rozdziale. Modele transformatorowe są zazwyczaj bardzo duże, wymagając znacznych zasobów obliczeniowych do trenowania i użytkowania. Używając analogii do samochodu, wyobraźmy sobie modele transformatorowe jako doładowane silniki, które potrzebują dużej mocy do pracy, ale potrafią zdumiewająco wiele zdziałać. Można je postrzegać jako kolejny krok po modelach takich jak ResNET 50, który służy do rozpoznawania obrazów. Podczas gdy ResNET 50 przypomina samochód z 50 biegami, GPT-3 firmy OpenAI przypomina megaciężarówkę z 96 biegami i dodatkowymi funkcjami. Ze względu na swoje zaawansowane możliwości, modele te są najlepszym wyborem do tworzenia inteligentnych wyników AI. Modele LLM wykorzystują transformatory, które składają się z enkodera i dekodera. Koder przetwarza tekst wejściowy (tj. monit) i generuje sekwencję stanów ukrytych, które reprezentują znaczenie tekstu wejściowego. Dekoder wykorzystuje te stany ukryte do generowania tekstu wyjściowego. Te enkodery i dekodery tworzą jedną warstwę, podobną do mini-mózgu. Wiele warstw można układać jedna na drugiej. Jak wspomniano wcześniej, GPT3 to model wyłącznie dekoderowy z 96 warstwami.

Punkt odcięcia treningu

W kontekście modeli fundamentalnych punkt odcięcia treningu odnosi się do punktu, w którym kończy się trening modelu, czyli do momentu zebrania danych użytych do trenowania modelu. W przypadku modeli AI opracowanych przez OpenAI, takich jak GPT-3 lub GPT-4, punkt odcięcia treningu przypada na moment, w którym model został ostatnio wytrenowany na nowych danych. Ten punkt odcięcia jest istotny, ponieważ po tym momencie model nie jest świadomy żadnych zdarzeń, postępów, nowych koncepcji ani zmian w użyciu języka. Na przykład, punkt odcięcia danych treningowych dla GPT-3.5 Turbo przypadał na wrzesień 2021 r., dla GPT-4 Turbo na kwiecień 2023 r., a dla GPT-4o na październik 2023 r., co oznacza, że model nie jest świadomy rzeczywistych wydarzeń ani postępów w różnych dziedzinach po tym punkcie. Kluczowe jest to, że chociaż modele te mogą generować tekst na podstawie danych, na których zostały wytrenowane, nie uczą się ani nie aktualizują swojej wiedzy po osiągnięciu punktu odcięcia treningu. Nie mogą uzyskiwać dostępu ani pobierać informacji w czasie rzeczywistym z Internetu ani żadnej zewnętrznej bazy danych. Ich odpowiedzi są generowane wyłącznie na podstawie wzorców, których nauczyli się podczas szkolenia.

UWAGA: Niedawna zapowiedź, że wersje premium ChatGPT będą miały dostęp do internetu za pośrednictwem wtyczki Bing, nie oznacza, że model dysponuje bardziej aktualnymi informacjami.

Wykorzystuje to wzorzec zwany RAG (generacja rozszerzona o pobieranie), który zostanie omówiony w dalszej części.

Rodzaje LLM

Jak pokazano w tabeli , istnieją trzy kategorie LLM. Mówiąc o LLM, kluczowy jest kontekst, który w niektórych przypadkach może być nieoczywisty. Ma to ogromne znaczenie, ponieważ ścieżki, którymi możemy podążać, korzystając z modeli, nie są wymienne, a wybór odpowiedniego typu zależy od przypadku użycia, który próbujemy rozwiązać. Ponadto istnieje również zależność od tego, jak skutecznie można dostosować modele do konkretnych przypadków użycia.

Typ LLM: Opis

Bazowy LLM: Są to oryginalne modele, wstępnie wytrenowane na ogromnym korpusie danych tekstowych, które mogą generować tekst w oparciu o wzorce poznane podczas tego szkolenia. Niektórzy nazywają je również surowymi modelami językowymi, a nawet modelami fundamentalnymi; można ich używać od razu do generowania tekstu. Uczą się zaawansowanych i ogólnych reprezentacji, ale brakuje im specjalistycznej wiedzy w danym zadaniu. Model DaVinci GPT-3 jest przykładem bazowego LLM.

LLM oparty na instrukcjach: Polega na użyciu bazowego LLM i podaniu wyraźnych instrukcji w natychmiastowym wprowadzeniu. W wielu przykładach, które widzieliśmy w poprzednim rozdziale, instruowaliśmy model, aby wykonywał instrukcje, takie jak „Przetłumacz poniższy tekst na francuski:" lub „Podsumuj ten artykuł:". Czasami te modele są również nazywane modelami LLM dostrojonymi do instrukcji.

Dostrojony LLM: Dostrojenie polega na wykorzystaniu bazowego LLM i dalszym trenowaniu go w zadaniu, w którym może sobie słabo radzić, często w określonej domenie. Przykładem może być trenowanie modelu w oparciu o literaturę medyczną, jeśli chcemy, aby rozumiał on zagadnienia medyczne, lub trenowanie go w oparciu o interakcje z obsługą klienta, jeśli chcemy, aby odpowiadał na zapytania klientów z konkretnej branży. Dostrajanie może pomóc w zwiększeniu dokładności modelu lub jego przydatności w określonych zadaniach lub domenach, ale wymaga dodatkowych danych i czasu na trenowanie.

Warto zauważyć, że wszystkie te metody mają swoje wady i zalety:

* Podstawowe modele LLM są wszechstronne i mogą obsługiwać wiele zadań bez dodatkowego szkolenia. Mogą jednak nie być tak dokładne i niezawodne, jak byśmy tego oczekiwali w przypadku konkretnych zadań lub dziedzin, zwłaszcza w środowisku korporacyjnym.

* Wykorzystanie oparte na instrukcjach może być bardzo skuteczne w przypadku niektórych zadań, ale wymaga starannego, szybkiego tworzenia i nie zmienia zasadniczo możliwości modelu. To właśnie tutaj stosuje się wiele technik szybkiego projektowania i najlepszych praktyk.

* Dostrajanie może przynieść doskonałe rezultaty w przypadku konkretnych zadań lub dziedzin. Wymaga jednak dodatkowych zasobów i wiąże się z ryzykiem przeuczenia danych treningowych, co może ograniczyć zdolność modelu do generalizacji na nowe przykłady.

Ponadto, można zastosować podejścia (bez prób, z małą liczbą prób i transfer learning), aby dodatkowo dostosować model LLM do konkretnego zadania, aby poprawić jego wydajność i zwiększyć niezawodność w tych konkretnych dziedzinach. Do pewnego stopnia rodzaj wdrożonego modelu LLM również determinuje, które podejście jest lepiej dopasowane do potrzeb. Każde podejście ma swoje

mocne i słabe strony, a wybór najlepszego zależy od konkretnego zadania, dostępnych danych i dostępnych zasobów:

* Uczenie się bez prób — model może generalizować na zadanie, nie widząc przykładów tego zadania podczas treningu. Na przykład, moglibyśmy poprosić model wyszkolony tylko w języku angielskim o przetłumaczenie z angielskiego na niemiecki, nawet jeśli nie widział żadnych konkretnych przykładów języka niemieckiego podczas treningu. Może on wykonać to tłumaczenie bez prób, wykorzystując podobieństwo semantyczne, osadzanie słów i uczenie maszynowe. Korzystając z tych technik, możemy zmierzyć, jak podobne są dwa słowa nawet w różnych językach.

* Uczenie się bez prób — polega ono na pokazaniu modelowi przykładów zadania, które chcemy wykonać, a następnie poproszeniu modelu o wykonanie tego samego zadania z nowym przykładem. Zatem, jeśli chcemy, aby model zidentyfikował nastrój tweeta, możemy pokazać mu kilka przykładów tweetów i powiązanego nastroju, a następnie poprosić go o przewidzenie nastroju nowego tweeta.

* Uczenie transferowe — polega na wytrenowaniu modelu na jednym zadaniu, a następnie wykorzystaniu zdobytej wiedzy do innego, ale powiązanego zadania. Na przykład, chociaż LLM-y zostały przeszkolone w zakresie obsługi języka, a nie konkretnych zgłoszeń klienta, mogą być wykorzystywane do kategoryzowania zgłoszeń klienta do różnych kategorii, takich jak rozliczenia, problemy techniczne lub zapytania ogólne. Może to usprawnić proces obsługi klienta i zapewnić szybkie rozwiązanie problemu, a co za tym idzie – wyższy poziom satysfakcji klienta.

Małe modele językowe

Małe modele językowe (SLM) to nowy trend, który wzbudził zainteresowanie wielu przedsiębiorstw. Są to pomniejszone wersje większych modeli językowych, zaprojektowane tak, aby zapewnić wiele korzyści ich większych odpowiedników, a jednocześnie być bardziej wydajne pod względem zasobów i dostępne. Różnią się one od modeli LLM (np. GPT-4) pod kilkoma względami, przede wszystkim pod względem rozmiaru i złożoności, zasobów obliczeniowych, kosztów szkolenia i operacyjnych oraz jakości działania. Techniki takie jak destylacja wiedzy i transfer uczenia się pozwalają mniejszym modelom osiągać doskonałe wyniki w analizie, tłumaczeniu i podsumowywaniu dzięki szybszemu szkoleniu. W niektórych przypadkach mogą one również dorównywać większym modelom LLM lub je przewyższać, co czyni je przełomowymi. Jeśli chodzi o rozmiar i złożoność, modele SLM są zauważalnie mniejsze i mają mniej parametrów niż duże modele, takie jak GPT-4. Ta różnica w skali jest znacząca: mały model może mieć od milionów do kilku miliardów parametrów, w przeciwieństwie do dziesiątek lub setek miliardów parametrów w dużych modelach. To zmniejszenie rozmiaru i złożoności sprawia, że modele SLM zasadniczo różnią się pod względem sposobu przetwarzania i generowania języka. Z perspektywy zasobów obliczeniowych, mniejsze rozmiary SLM wymagają mniejszej mocy obliczeniowej. To mniejsze zapotrzebowanie sprawia, że SLM-y są bardziej odpowiednie dla aplikacji o ograniczonych możliwościach przetwarzania lub sytuacji, w których kluczowe znaczenie mają odpowiedzi w czasie rzeczywistym. Mniejsze zapotrzebowanie na zasoby obliczeniowe oznacza również, że SLM-y można wdrażać w szerszym zakresie środowisk, w tym na urządzeniach brzegowych lub w systemach o niższej mocy obliczeniowej. Jeśli chodzi o koszty szkolenia i eksploatacji, SLM-y są generalnie tańsze w szkoleniu i obsłudze. Ta opłacalność wynika z ich mniejszej złożoności i mniejszej ilości danych potrzebnych do ich szkolenia. W rezultacie SLM-y stają się bardziej dostępne dla osób i organizacji o ograniczonych budżetach, demokratyzując dostęp do zaawansowanych technologii przetwarzania języka. Jednak pod względem wydajności i jakości, chociaż SLM-y są w stanie efektywnie obsługiwać szeroki zakres zadań językowych, zazwyczaj nie dorównują poziomem wydajności większym modelom. Jest to szczególnie widoczne w przypadku bardziej złożonych zadań, które wymagają szerokiego zrozumienia kontekstu lub specjalistycznej wiedzy. Duże modele, z większą

głębią i zakresem danych oraz zrozumieniem, są lepiej przygotowane do radzenia sobie z taką złożonością. Z kolei SLM-y mogą mieć trudności z pokonaniem tych wyzwań ze względu na swoje ograniczenia w zakresie rozmiaru i szkolenia.

Strategiczny dobór danych i nowe, innowacyjne podejście do szkolenia to dwa główne powody, dla których modele SLM, takie jak seria Phi, odniosły tak wielki sukces. Strategiczny dobór danych stawia wysoką jakość ponad ilość i polega na wykorzystaniu danych o jakości podręcznikowej, w tym syntetycznych zestawów danych i skrupulatnie wyselekcjonowanych danych internetowych. Dane są dobierane tak, aby zapewnić solidny fundament rozumowania opartego na zdrowym rozsądku i wiedzy ogólnej. To strategiczne podejście do doboru danych ma kluczowe znaczenie dla wyjątkowej wydajności modelu w całym spektrum zadań. Z kolei innowacyjne podejście do szkolenia obejmuje skalowanie w górę z mniejszych modeli, takich jak Phi-1.5, i osadzanie wiedzy w Phi-2 [3]. Ta metoda przyspiesza konwergencję szkolenia i poprawia wyniki testów porównawczych, podważając konwencjonalne prawa skalowania i pokazując, że można osiągnąć niezwykłe możliwości nawet w przypadku modeli językowych o mniejszej skali. Modele SLM są wciąż na wczesnym etapie cyklu życia, ale są coraz częściej brane pod uwagę do użytku produkcyjnego w przedsiębiorstwach. Jednak ich gotowość w dużej mierze zależy od konkretnych wymagań i kontekstu aplikacji. Oto kilka czynników, które należy wziąć pod uwagę:

* Złożoność zadań — SLM-y nadają się do prostszych, bardziej zdefiniowanych zadań. Jednak większy model może być niezbędny dla optymalnej wydajności, jeśli aplikacja przedsiębiorstwa wymaga zrozumienia lub generowania złożonego języka.

* Ograniczenia zasobów — SLM-y stanowią doskonały wybór dla firm o ograniczonych zasobach obliczeniowych lub tych, które muszą wdrażać modele na urządzeniach brzegowych ze względu na niższe wymagania dotyczące zasobów.

* Efektywność kosztowa — Korzystanie z SLM-ów jest zazwyczaj bardziej opłacalne niż korzystanie z większych modeli pod względem zasobów obliczeniowych i zużycia energii. Może to stanowić znaczącą zaletę dla przedsiębiorstw, które chcą zminimalizować koszty operacyjne.

* Szybkość i responsywność — SLM-y mogą oferować krótszy czas reakcji, co jest korzystne w przypadku aplikacji, w których interakcja w czasie rzeczywistym ma kluczowe znaczenie, takich jak chatboty obsługi klienta.

* Lokalnie — W przypadku aplikacji, których nie można wdrożyć ani połączyć z chmurą ze względu na przepisy lub regulacje, SLM-y mogą być opcją, ponieważ można je łatwiej wdrożyć lokalnie lub w chmurze prywatnej. Oto kilka przykładów dostępnych obecnie modeli SLM:

* Phi-3 — rodzina małych modeli językowych, następca Phi-2, niedawno wprowadzonych przez Microsoft. Znane są z przewyższania innych modeli o podobnej lub nawet większej wielkości w różnych testach porównawczych i występują w trzech rozmiarach: mini (3,8 mld), małym (7 mld) i średnim (14 mld).

* Phi-2 — 2,7-miliardowy model Microsoftu, który demonstruje najnowocześniejszą wydajność w zadaniach związanych z rozumowaniem i rozumieniem języka, przewyższając modele 25-krotnie większe.

* Orca 2 — 7- lub 13-miliardowy model Microsoftu, który uczy się różnych technik rozumowania i strategii rozwiązywania od bardziej zaawansowanego modelu nauczyciela 5.

* Gemini Nano – model Google o 122 milionach parametrów, należący do serii Gemini, zaprojektowany z myślą o efektywnym wnioskowaniu i wdrażaniu na urządzeniach brzegowych.

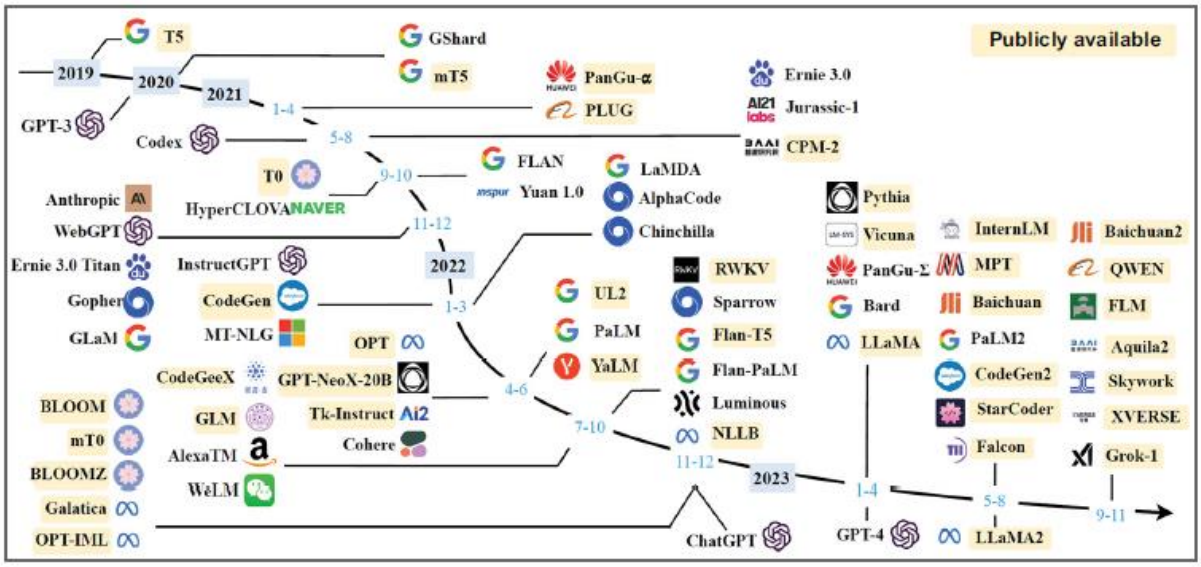
* DistilBERT – mniejsza wersja BERT, która zachowuje 97% swoich możliwości rozumienia języka, będąc jednocześnie o 40% mniejsza i o 60% szybsza.

* GPT-Neo – mniejsza wersja architektury GPT (125 MB i 1,3 B), należąca do serii GPT-Neo stworzonej przez EleutherAI.

Modele SLM są szczególnie przydatne, gdy wdrożenie dużego modelu jest niewykonalne ze względu na koszty, szybkość lub wymagania obliczeniowe. Zapewniają one równowagę między wydajnością a efektywnością, ułatwiając dostęp do zaawansowanych funkcji przetwarzania języka naturalnego. Chociaż modele SLM mogą nie być odpowiednie dla każdej aplikacji korporacyjnej, zwłaszcza tych wymagających głębokiego zrozumienia lub generowania złożonego języka, są gotowe do wdrożenia w wielu scenariuszach, szczególnie tam, gdzie wydajność, szybkość i koszt są kluczowymi czynnikami. Przedsiębiorstwa powinny ocenić swoje specyficzne potrzeby i ograniczenia, aby ustalić, czy model SLM jest właściwym wyborem dla ich aplikacji.

Modele open source a komercyjne modele LLM

Obecne modele komercyjne zapewniają najwyższą wydajność pod względem jakości sztucznej inteligencji i szerokiego zakresu możliwości. Jednak od czasu wydania ChatGPT nastąpił znaczący zwrot w kierunku modeli open source. Wiele z tych inicjatyw open source koncentruje się na opracowywaniu mniejszych modeli fundamentalnych, twierdząc, że mogą one osiągnąć niemal ten sam poziom jakości bez znaczących strat. Rysunek 2 ilustruje te linie rozwoju i ich szybki rozwój.



Komercyjne studia LLM

Obecnie istnieje kilka komercyjnych studiów LLM, które zostaną opublikowane i zyskują na popularności. Prawie wszystkie z nich są zgodne z paradygmatem OpenAI i są udostępniane za pośrednictwem interfejsu API, z którego korzystamy. Chociaż wciąż są to startupy, wiele z nich dysponuje poważnym finansowaniem, a ich założyciele mają bogate doświadczenie badawcze:

* OpenAI to laboratorium badawcze zajmujące się sztuczną inteligencją, które opracowuje i publikuje najnowocześniejsze modele AI, takie jak seria GPT. Z pewnością cieszy się obecnie największym

zainteresowaniem. Opiera się na kilku podstawowych modelach, takich jak GPT-4, DALL.E i ChatGPT, i jest najbardziej dojrzałe w tej grupie, z poważnym wsparciem i własnością firmy Microsoft.

* Azure OpenAI i OpenAI oferują dostęp do zaawansowanych modeli językowych, ale różnią się charakterem i implementacją. Główną różnicą jest to, że Azure OpenAI jest usługą zarządzaną, podczas gdy OpenAI nią nie jest. Microsoft zajmuje się podstawową infrastrukturą i konserwacją Azure OpenAI, co czyni ją cenną opcją dla firm, które nie mają wystarczających zasobów, aby zarządzać wdrożeniem OpenAI. Co więcej, Azure OpenAI pakuje surowe modele OpenAI w przyjazne dla deweloperów usługi, które deweloperzy mogą bezproblemowo integrować ze swoimi aplikacjami. Usługi te działają na platformie Azure, zapewniając wysoką skalowalność, niezawodność i globalną dostępność.

* Anthropic to startup założony przez byłych inżynierów OpenAI, który wydał Claude, model LLM, który umożliwia generowanie tekstu i kodu. Ich kluczowym wyróżnikiem jest implementacja LLM z wykorzystaniem konstytucyjnej sztucznej inteligencji [5]. Konstytucyjna sztuczna inteligencja wykorzystuje uczenie przez wzmacnianie (RL) i tradycyjne uczenie nadzorowane i twierdzi, że generuje mniej szkodliwe wyniki. W chwili publikacji Anthropic był wspierany zarówno przez Google, jak i Amazon. Claude 3, najnowsza rodzina modeli, ma trzy wersje: Haiku (mała), Sonnet (średnia) i Opus (duża).

* Gemini to najnowszy model GenAI firmy Google, dostępny w ramach oferty Google Cloud w niedawno uruchomionym produkcie Google AI Studio. W momencie pisania tego tekstu Google udostępnia dostęp do modeli za pośrednictwem API w ramach prywatnej wersji zapoznawczej.

* Cohere AI, startup wywodzący się z publikacji Transformer („Attention is all you need”), oferuje program LLM oraz inne produkty, takie jak Neural Search i Embed.

Programy LLM oparte na oprogramowaniu open source

Pojawia się nowa generacja programów LLM opartych na oprogramowaniu OSS, a niektóre z nich będą konkurować z ChatGPT. Jak pokazuje rysunek 2.1, jest ich zbyt wiele, aby je wszystkie wymienić, ale kilka godnych uwagi zostało wymienionych w tabeli 2.

Firma: Open Source LLM: Rozmiar parametru

Meta: Llama LLM to jeden z modeli, który zainspirował wiele innych modeli OSS. Występuje w wielu rozmiarach (7B, 13B, 33B i 65B) i choć jest mniejszy niż GPT-3, można go dopasować do wielu zadań. Meta udostępniła modele badaczom (i zostały one również osobno ujawnione w internecie), inspirując wielu innych do wykorzystania ich jako punktu wyjścia. Różne (7B–65B)

Databricks: Databricks niedawno wydało wersję 2 Dolly, którą określają mianem „pierwszego na świecie prawdziwie otwartego, dostrojonego do instrukcji LLM”. Jest on udostępniany na licencji CCA 3.0, co pozwala każdemu na jego używanie, rozszerzanie i modyfikowanie, w tym w celach komercyjnych. Alpaca: Alpaca Uniwersytetu Stanforda, model instrukcji oparty na Llama, twierdzi, że w niektórych zadaniach dorównuje wydajności GPT-3.5 Turbo. : 7B

FreedomGPT: To agent konwersacyjny OSS oparty na bibliotece Alpaca. Firma twierdzi, że oferuje w 100% nieocenzurowane i prywatne rozmowy.:

Nieujawnione

Vicuna: Naukowcy z wielu instytucji (UC Berkeley, CMU, Stanford, UC San Diego i MBZUAI) wydali Vicunę, udoskonaloną wersję Llamy, która dorównuje wydajnością GPT4 w wielu zadaniach.: 13B

Koala: Berkley AI Research wydało Koalę, udoskonaloną wersję Llamy wykorzystującą dialogi internetowe.: 13B

ChatLLaMa: Technicznie rzecz biorąc, nie jest to model, lecz narzędzie do tworzenia modeli. Nebuly AI wydało ChatLLaMa, bibliotekę, która może tworzyć pomoce konwersacyjne na wzór ChatGPT, wykorzystując Twoje dane. : 7B

Colossal-Chat: Projekt ColossalAI Uniwersytetu Kalifornijskiego w Berkeley wydał ColossalChat, model podobny do ChatGPT, który zawiera kompletne potoki RLHF oparte na Llama. : 7B

Falcon: Instytut Innowacji Technologicznych (TII) w Zjednoczonych Emiratach Arabskich wydał rodzinę modeli LLM zwaną modelem Falcon LLM. W tamtym czasie Falcon był największym modelem OSS LLM, jaki kiedykolwiek wydano, i znajdował się na szczycie rankingu OSS LLM. Niedawno na szczycie rankingu ponownie znalazł się bardziej zaawansowany model o parametrach 180B. : Różne (1B–180B)

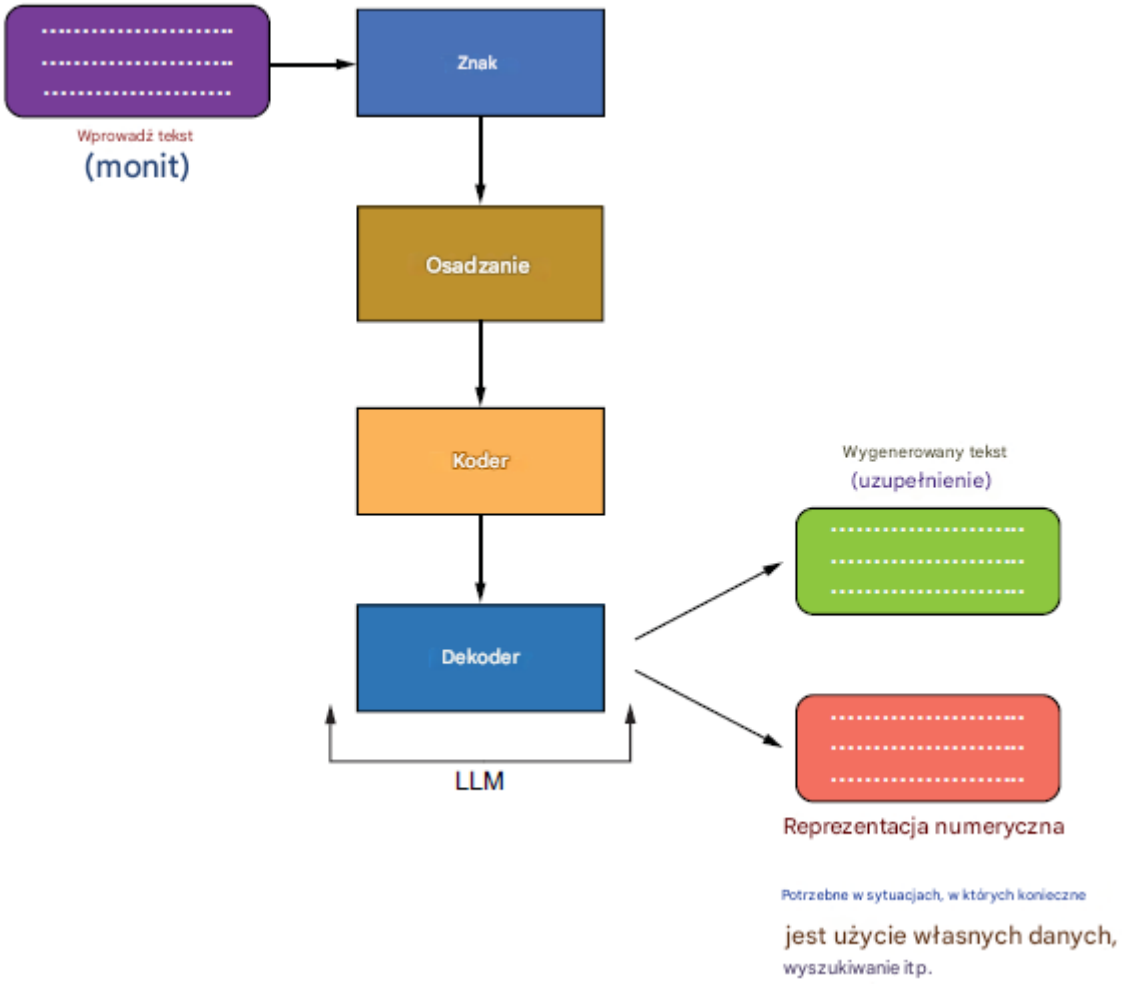
Mistral: Mistral AI, francuski startup, opracował szereg modeli. Niektóre z nich to modele open source licencjonowane na podstawie Apache 2.0, licencji zezwalającej na nieograniczone użytkowanie w dowolnym kontekście. Jak wspomniano w poprzednim rozdziale, istnieją również modele komercyjne. : Różne (7B–141B)

OpenAI kontra Azure OpenAI

Azure OpenAI i OpenAI to usługi zapewniające dostęp do zaawansowanych modeli językowych OpenAI, ale istnieją między nimi pewne kluczowe różnice. OpenAI jest bardziej skierowany do małych i średnich firm, indywidualnych deweloperów oraz startupów. Natomiast Azure OpenAI jest przeznaczony dla przedsiębiorstw, które potrzebują dodatkowego bezpieczeństwa i dostępności w różnych częściach świata i mają wymagania regulacyjne. Azure OpenAI oferuje dodatkowe funkcje gotowe do wdrożenia w przedsiębiorstwach, takie jak prywatność danych, klucze zarządzane przez klienta, szyfrowanie w stanie spoczynku, sieć prywatna, dostępność regionalna i odpowiedzialne filtrowanie treści AI. Funkcje te mogą być istotne dla firm, które muszą spełniać określone wymogi bezpieczeństwa lub regulacyjne. W związku z tym interfejsy API obu platform są podobne, ale nie takie same. Jednak modele bazowe są takie same, a wdrożenie Azure OpenAI obejmuje te dodatkowe funkcje, których potrzebuje większość przedsiębiorstw.

Kluczowe koncepcje LLM

W tej sekcji opisano architekturę typowej implementacji LLM. Rysunek 3 przedstawia abstrakcyjną strukturę typowej implementacji LLM na wysokim poziomie; proces ten jest powtarzany za każdym razem, gdy używamy LLM, takiego jak GPT OpenAI.



Model zaczyna się od tekstu wejściowego – monitu. Najpierw jest on konwertowany na sekwencję tokenów za pomocą tokenizacji. Następnie każdy token jest konwertowany na wektor numeryczny w procesie zwanym osadzaniem, który działa jako dane wejściowe kodera. Koder przetwarza sekwencję wejściową i generuje sekwencję stanów ukrytych. Te stany ukryte są następnie wprowadzane do dekodera z tokenem startowym. Dekoder generuje sekwencję wyjściową po jednym tokenie na raz, przewidując kolejny token na podstawie poprzednich tokenów i stanów ukrytych. Gdy wszystkie warstwy przetworzą informacje, model przewiduje kolejny token w wyuczonej sekwencji. Wynik jest konwertowany z powrotem na tekst, a my widzimy odpowiedź. Proces ten przebiega w pętli iteracyjnej i występuje dla każdego nowego wygenerowanego tokena, tworząc w ten sposób spójny tekst wyjściowy. Ostateczny tekst generowany przez model jest właściwością emergentną tego warstwowego, iteracyjnego procesu. Ostateczna sekwencja wyjściowa jest również nazywana uzupełnieniem. Przyjrzyjmy się każdemu z tych aspektów LLM bardziej szczegółowo.

Podpowiedzi

Podpowiedź to sposób, w jaki „rozmawiamy” z tymi modelami. Podpowiedź to po prostu tekst opisujący zadanie, które próbujemy osiągnąć, używając języka naturalnego. Dane wyjściowe tych modeli to również tekst. Możliwość wyrażenia naszych intencji w ten sposób (język naturalny), zamiast dostosowywania się do ograniczeń maszyny, sprawia, że podpowiedzi są potężne. Tworzenie lub projektowanie tekstu w podpowiedzi jest podobne do programowania modelu i tworzenia nowego paradygmatu zwanego inżynierią podpowiedzi, który zostanie omówiony w dalszej części książki. Podpowiedzi pozwalają nam uchwycić szerszy zestaw intencji i emocji, używając tego samego

paradygmatu. W poniższych dwóch przykładach (wygenerowanych przez sztuczną inteligencję) podpowiedź prosi o haiku opisujące, jak wspaniałe są LLM: jedno haiku jest radosne, a drugie smutne. Osiągnięcie takiego poziomu emocji i ekspresji w tym samym zadaniu było niemożliwe.

Napisz radosne haiku o tym, jak wspaniali są absolwenci studiów prawniczych (LLM).

Odpowiedź generowana przez sztuczną inteligencję

LLM są świetni

Potrafią rozmawiać i generować

Kreatywni i inteligentni

Napisz smutne haiku o tym, jak wspaniali są absolwenci studiów prawniczych (LLM).

Odpowiedź generowana przez sztuczną inteligencję

LLM są świetni

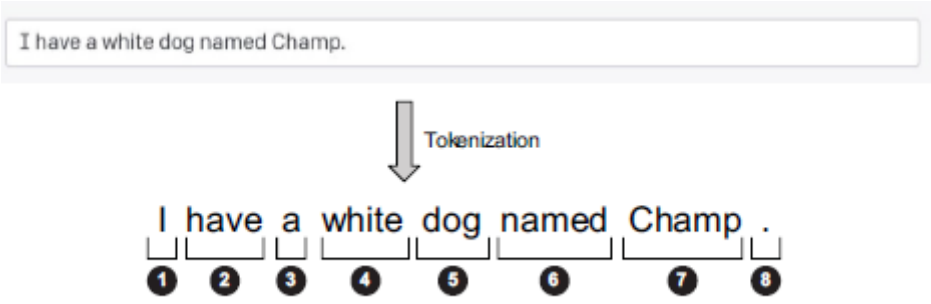
Ale nie potrafią czuć ani kochać

Tylko symulować

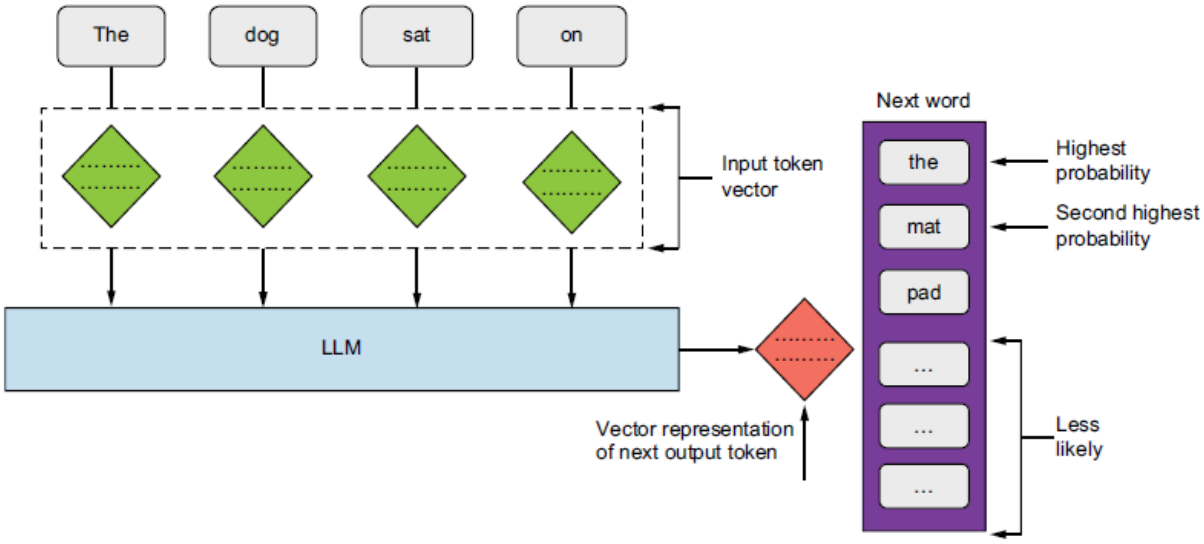
UWAGA: Modele LLM nie rozumieją tekstu tak jak ludzie. Są to modele statystyczne, które uczą się wzorców w danych i wykorzystują je do generowania wyników. Pomimo imponujących wyników, modele te nie rozumieją generowanej treści w takim samym sensie jak my – w tym przekonań, pragnień, świadomości oraz dobra i zła. Są po prostu niezwykle skuteczne w przewidywaniu kolejnego słowa w sekwencji tekstu na podstawie wzorców, które widziały miliony razy.

Tokeny

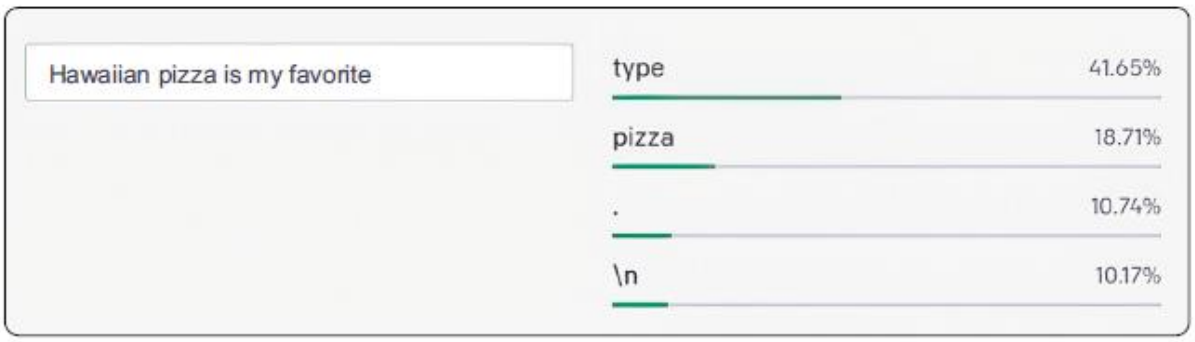
Tokeny to podstawowe jednostki tekstu, których LLM używa do przetwarzania zarówno żądania, jak i odpowiedzi, czyli do rozumienia i generowania tekstu. Tokenizacja to proces konwersji tekstu na sekwencję mniejszych jednostek zwanych tokenami. Używając LLM, używamy tokenów do komunikacji z tymi modelami, co jest jednym z najważniejszych elementów zrozumienia LLM. Tokeny to nowa waluta przy wdrażaniu LLM do aplikacji lub rozwiązań. Są one bezpośrednio powiązane z kosztami uruchomienia modeli, zarówno pod względem finansowym, jak i doświadczenia związanego z opóźnieniami i przepustowością. Im więcej tokenów, tym więcej przetwarzania musi wykonać model. Oznacza to, że model wymaga więcej zasobów obliczeniowych, co przekłada się na niższą wydajność i większe opóźnienia. LLM konwertuje tekst na tokeny przed przetworzeniem. W zależności od algorytmu tokenizacji, mogą to być pojedyncze znaki, słowa, podsłowa, a nawet większe jednostki językowe. Zasadniczo jeden token to około cztery znaki lub 0,75 słowa w przypadku tekstu w języku angielskim. W przypadku większości współczesnych systemów LLM obsługiwany rozmiar tokena obejmuje zarówno monit wejściowy, jak i odpowiedź. Zilustrujmy to przykładem. Rysunek 4 pokazuje, jak zdanie „I have a white dog named Champ” jest tokenizowane (w tym przypadku za pomocą tokenizatora OpenAI). Każdy blok reprezentuje inny token. W tym przykładzie używamy ośmiu tokenów.



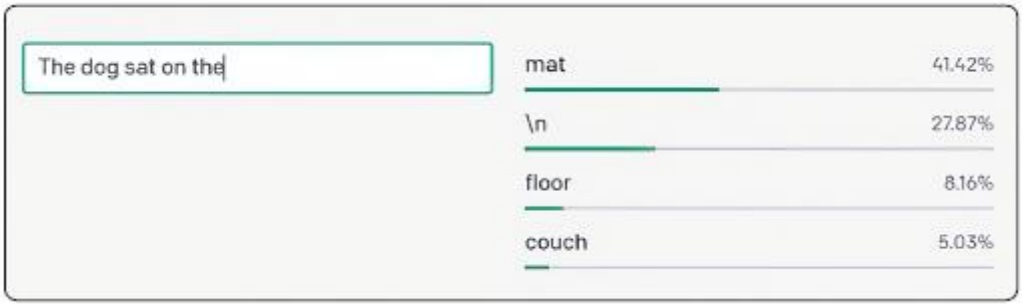
LLM-y generują tekst, przewidując kolejne słowo lub symbol (token), które najprawdopodobniej nastąpi po danej sekwencji słów lub symboli (tokenów) użytej jako dane wejściowe, czyli po monicie. Wizualną reprezentację tego zjawiska przedstawiamy na rysunku 5, gdzie lista tokenów po prawej stronie pokazuje najwyższe prawdopodobieństwo wystąpienia tokenów po monicie „Pies usiadł na".



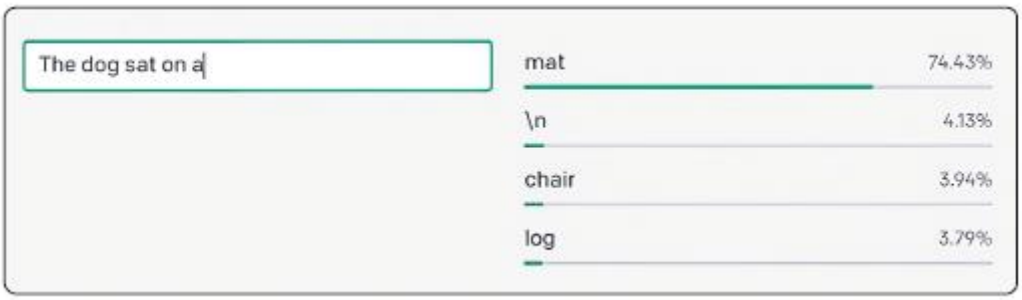
Możemy wpłynąć na część tego prawdopodobieństwa wystąpienia tokenów za pomocą kilku parametrów, które omówimy później. Załóżmy, że mamy sekwencję tokenów o długości n. Wykorzystując te n tokenów jako kontekst, generujemy kolejny token, n + 1. Ten nowo przewidziany token jest następnie dołączany do pierwotnej sekwencji tokenów, rozszerzając w ten sposób kontekst. W konsekwencji rozszerzone okno kontekstowe dla generowania tokenu n + 2 staje się n + (n + 1). Proces ten powtarza się w pętli, aż do osiągnięcia z góry określonego warunku zatrzymania, takiego jak określona sekwencja lub limit rozmiaru tokenów. Na przykład, jeśli mamy zdanie „Hawajska pizza to moja ulubiona", rozkład prawdopodobieństwa kolejnego słowa, które zobaczymy, pokazano na rysunku 6. Najbardziej prawdopodobnym słowem jest „type", kończąc zdanie „Hawajska pizza to mój ulubiony rodzaj".



Ponowne uruchomienie tego przykładu spowoduje otrzymanie prawdopodobieństwa innego niż przedstawione tutaj. Wynika to z faktu, że większość sztucznej inteligencji (AI) jest niedeterministyczna, szczególnie w przypadku modeli LLM. Jednocześnie model może przewidywać jeden token i prawdopodobnie jest on analizowany w odniesieniu do wszystkich możliwych tokenów, których model nauczył się w fazie uczenia. Używamy również dwóch przykładów, które pokazują, jak jeden token drastycznie zmienia rozkład (zmieniając jedno słowo z „the" na „a"). Rysunek 7 pokazuje, że najbardziej prawdopodobnym kolejnym tokenem jest „mat" z prawdopodobieństwem 41%. Widzimy również listę pozostałych tokenów i ich rozkładów probabilistycznych.



Jednak zmiana jednego żetonu z „the" na „a" drastycznie zmienia kolejny zestaw dystrybucji, a mata podnosi się o 30 punktów do prawdopodobieństwa wynoszącego prawie 75%, jak pokazano na rysunku 8.



Niektóre ustawienia związane z modelami LLM są istotne i mogą zmieniać sposób działania modelu oraz generowania tekstu. Ustawienia te to konfiguracje modelu, które można zmienić za pomocą interfejsu API, interfejsu graficznego lub obu tych metod. Konfiguracje modeli omówimy bardziej szczegółowo w dalszej części rozdziału.

Liczenie tokenów

Wielu programistów prawdopodobnie nie będzie miało doświadczenia ze śledzeniem tokenów podczas korzystania z LLM, zwłaszcza w środowisku korporacyjnym. Liczenie tokenów jest jednak ważne z kilku powodów:

* Ograniczenia pamięci — modele LLM mogą przetwarzać maksymalną liczbę tokenów w jednym przebiegu. Wynika to z ograniczeń pamięci ich architektury, często definiowanych przez okno kontekstowe (kolejna koncepcja, którą omówimy w dalszej części tego rozdziału). Na przykład, najnowszy model GPT-4o firmy OpenAI ma okno treści o rozmiarze 128 KB, a najnowszy model Gemini 1.5 Pro firmy Google ma okno kontekstowe o rozmiarze 1 mln tokenów. GPT3.5-Turbo, kolejny model OpenAI, ma dwa modele obsługujące tokeny o długości 8 KB i 16 KB. Trwają badania nad rozwiązaniem tego problemu, na przykład LongNet firmy Microsoft Research, który pokazuje, jak skalować do 1B okien kontekstowych. Należy podkreślić, że jest to wciąż aktywny obszar badań i nie został jeszcze wprowadzony na rynek.

* Koszt — Myśląc o kosztach, należy wziąć pod uwagę dwa wymiary: koszty obliczeniowe w odniesieniu do opóźnień, pamięci i ogólnego doświadczenia oraz rzeczywisty koszt w ujęciu pieniężnym. Dla każdego wywołania zasoby obliczeniowe wymagane do przetwarzania tokenów są bezpośrednio skorelowane z długością tokenów. Wraz ze wzrostem długości tokenu, wymaga on dłuższego czasu przetwarzania, co prowadzi do większych wymagań obliczeniowych (w szczególności dotyczących pamięci i procesorów graficznych) oraz wyższych opóźnień. Oznacza to również wzrost kosztów korzystania z modeli LLM.

* Jakość sztucznej inteligencji — Jakość wyników modelu zależy od liczby tokenów, które ma on wygenerować lub przetworzyć. Jeśli tekst jest zbyt krótki, model może nie mieć wystarczającego kontekstu, aby udzielić dobrej odpowiedzi. I odwrotnie, jeśli tekst jest zbyt długi, model może stracić spójność w swojej odpowiedzi. Do kwestii dobra i zła wrócimy później w rozdziale 6, jako do części poświęconej inżynierii natychmiastowej.

Dla wielu przedsiębiorstw koszt i wydajność są kluczowymi czynnikami decydującymi o użyciu tokenów. Ogólnie rzecz biorąc, mniejsze modele są bardziej opłacalne i wydajne niż większe. Listing 1 przedstawia prosty sposób obliczenia liczby tokenów. W tym przykładzie używamy biblioteki open source o nazwie tiktoken, wydanej przez OpenAI. Ta biblioteka tokenizera implementuje algorytm kodowania par bajtów (BPE). Te tokenizery są projektowane z wykorzystaniem odpowiednich modeli LLM, co zapewnia wydajną tokenizację i optymalną wydajność podczas procesów wstępnego trenowania i dostrajania. Jeśli korzystasz z jednego z modeli OpenAI, musisz użyć tego tokenizera; wiele innych modeli transformerów również go używa. W razie potrzeby możesz zainstalować bibliotekę tiktoken za pomocą polecenia pip install tiktoken.

Listing 1 : Zliczanie tokenów dla GPT

```
import tiktoken as tk

def count_tokens(string: str, encoding_name: str) -> int:
    # Get the encoding
    encoding = tk.get_encoding(encoding_name)

    # Encode the string
    encoded_string = encoding.encode(string)

    # Count the number of tokens
    num_tokens = len(encoded_string)
    return num_tokens

# Define the input string
prompt = "I have a white dog named Champ"


# Display the number of tokens in the String
print("Number of tokens:" , count_tokens(prompt, "cl100k_base"))
```

The encoding specifies how the text is converted into tokens.

Uruchomienie tego kodu, zgodnie z oczekiwaniami, daje nam następujący wynik:

```
$ python countingtokens.py
Number of tokens: 7
```

UWAGA: Kodowanie par bajtów (BPE) to algorytm kompresji szeroko stosowany w zadaniach przetwarzania języka naturalnego (NLP), takich jak klasyfikacja tekstu, generowanie tekstu i tłumaczenie maszynowe. Jedną z zalet BPE jest jego odwracalność i bezstratność, dzięki czemu możemy uzyskać tekst oryginalny. BPE działa na dowolnym tekście, którego dane treningowe tokenizatora nie rozpoznały, i kompresuje tekst, co skutkuje krótszymi sekwencjami tokenów niż tekst oryginalny. BPE pomaga również uogólniać powtarzające się wzorce w języku i zapewnia lepsze zrozumienie gramatyki. Na przykład forma gerundium -ing jest dość powszechna w języku angielskim (swimming, running, debugging itp.). BPE podzieli ją na różne tokeny, więc „swim" i „-ing" w swimming staną się dwoma tokenami i będą lepiej generalizować.

Jeśli nie jesteśmy pewni nazwy kodowania, zamiast funkcji get_encoding() możemy użyć funkcji encoding_for_model(). Przyjmuje ona nazwę modelu, którego chcemy użyć, i wykorzystuje odpowiednie kodowanie, na przykład encoding = tiktoken.encoding_for_model('gpt-4'). W przypadku OpenAI tabela 2.3 przedstawia różne obsługiwane kodowania.

Kodowanie: model OpenAI

cl100k_base: gpt-4, gpt-3.5-turbo, gpt-35-turbo, text-embedding-ada-002

p50k_base: modele Codex, text-davinci-002, text-davinci-003

r50k_base: modele GPT-3 (davinci, curie, babage, ada)

Listing 2 pokazuje, jak używać różnych kodowań i jak uzyskać oryginalny tekst z tokenów. Na razie powinniśmy rozumieć to jako konstrukcję podstawową, ale jest ona przydatna w bardziej zaawansowanych przypadkach użycia, takich jak buforowanie i dzielenie tekstu na fragmenty – aspekty, które omówimy w dalszej części książki.

Listing 2 : Tokeny

import tiktoken as tk

```
def get_tokens(string: str, encoding_name: str) -> str:

# Get the encoding

encoding = tk.get_encoding(encoding_name)

# Encode the string

return encoding.encode(string)

def get_string(tokens: str, encoding_name: str) -> str:

# Get the encoding

encoding = tk.get_encoding(encoding_name)

# Decode the tokens

return encoding.decode(tokens)

# Define the input string

prompt = “I have a white dog named Champ.”

# Display the tokens

print(“cl100k_base Tokens:” , get_tokens(prompt, “cl100k_base”))

print(“ p50k_base Tokens:” , get_tokens(prompt, “p50k_base”))

print(“ r50k_base Tokens:” , get_tokens(prompt, “r50k_base”))

print(“Original String:” , get_string([40, 617, 264, 4251, 5679, 7086, 56690,

13], “cl100k_base”))

$ python encodings.py

cl100k_base Tokens: [40, 617, 264, 4251, 5679, 7086, 56690, 13]

p50k_base Tokens: [40, 423, 257, 2330, 3290, 3706, 29260, 13]

r50k_base Tokens: [40, 423, 257, 2330, 3290, 3706, 29260, 13]

Original String: I have a white dog named Champ.
```

Oprócz biblioteki tiktoken, której używaliśmy w przykładach, istnieje kilka innych popularnych tokenizerów. Należy pamiętać, że każdy tokenizer jest przeznaczony dla odpowiedniego LLM i nie można go stosować zamiennie:

* WordPiece — używany przez model BERT firmy Google, dzieli tekst na mniejsze jednostki na podstawie najczęściej występujących fragmentów słów, umożliwiając efektywną reprezentację rzadkich lub nieistniejących słów.

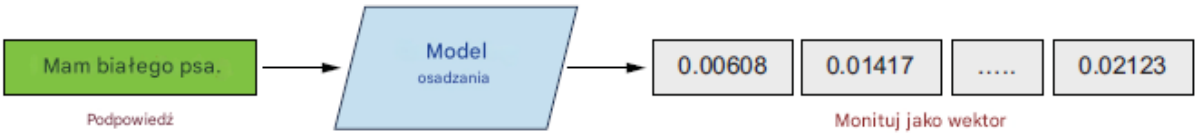
* SentencePiece — model RoBERTa (Robustly Optimized BERT) firmy Meta wykorzystuje ten model. Łączy on podejścia WordPiece i BPE w jeden niezależny od języka framework, zapewniając większą elastyczność.

* Tokenizer T5 — oparty na SentencePiece, jest używany przez model T5 firmy Google (Text-to-Text Transfer Transformer).

* Tokenizer XLM — jest używany w modelu XLM (Cross-lingual Language Model) firmy Meta i implementuje metodę BPE z nauczonymi osadzeniami (BPEmb). Jest przeznaczony do obsługi tekstów wielojęzycznych i wspierania transferu międzyjęzykowego.

Osadzenia

Osadzenia to potężne narzędzia uczenia maszynowego dla dużych danych wejściowych reprezentujących słowa. Wychwytują one podobieństwa semantyczne w przestrzeni wektorowej (tj. zbiorze wektorów, jak pokazano na rysunku 9), pozwalając nam określić, czy dwa fragmenty tekstu reprezentują to samo znaczenie.



Dostarczając wskaźnik podobieństwa, osadzenia mogą pomóc nam lepiej zrozumieć relacje między różnymi fragmentami tekstu. Idea osadzania polega na tym, że słowa o podobnym znaczeniu powinny mieć podobne reprezentacje wektorowe, mierzone odległościami między nimi. Wektory o mniejszych odległościach między nimi sugerują silne pokrewieństwo, a te o większych odległościach sugerują niskie pokrewieństwo. Istnieje kilka sposobów pomiaru podobieństw; omówimy je później w rozdziale 7. Wektory te są uczone podczas uczenia i służą do wychwytywania znaczenia słów lub fraz. Algorytmy sztucznej inteligencji mogą z łatwością wykorzystać te wektory liczb zmiennoprzecinkowych. Na przykład słowo „cat" może być reprezentowane przez wektor jako [0,2, 0,3, -0,1], a słowo „dog" jako [0,4, 0,1, 0,2]. Wektory te mogą następnie służyć jako dane wejściowe dla modeli uczenia maszynowego w zadaniach takich jak klasyfikacja tekstu, analiza sentymentu i tłumaczenie maszynowe. Osadzenia są uczone, gdy model jest trenowany na dużym korpusie danych tekstowych. Ideą jest uchwycenie znaczenia słów lub fraz na podstawie ich kontekstu w danych treningowych.

W zależności od zadania istnieje kilka algorytmów tworzenia osadzeń:

* Osadzenia podobieństwa są skuteczne w wychwytywaniu podobieństwa semantycznego między dwoma lub więcej fragmentami tekstu.

* Osadzenia wyszukiwania tekstowego mierzą, czy długie dokumenty są istotne dla krótkiego zapytania.

* Osadzenia wyszukiwania kodu są przydatne do osadzania fragmentów kodu i zapytań wyszukiwania w języku naturalnym.

UWAGA: Osadzenia utworzone jedną metodą nie mogą być zrozumiane przez inną. Innymi słowy, jeśli utworzysz osadzenie za pomocą API OpenAI, osadzania innego dostawcy nie będą rozpoznawać utworzonych wektorów i odwrotnie.

Listing 3 pokazuje, jak uzyskać osadzenie (w tym przykładzie z OpenAI). Definiujemy funkcję o nazwie get_embedding(), która przyjmuje jako parametr ciąg znaków, dla którego chcemy utworzyć osadzenia. Funkcja ta wykorzystuje API OpenAI do generowania osadzenia dla tekstu wejściowego, korzystając z modelu text-embedding-ada-002. Osadzanie jest zwracane jako lista liczb zmiennoprzecinkowych.

Listing 3 : Pobieranie osadzenia w OpenAI

```
import os
```

```
from openai import OpenAI

client = OpenAI(api_key='your-API-key')

def get_embedding(text):

response = client.embeddings.create(

model="text-embedding-ada-002",

input=text)

return response.data[0].embedding

embeddings = get_embedding("I have a white dog named Champ.")

print("Embedding Length:", len(embeddings))

print("Embedding:", embeddings[:5])
```

Przestrzeń wektorowa powstała w wyniku osadzenia nie jest odwzorowaniem jednoznacznym na tokeny, ale może być czymś znacznie większym. Poniżej przedstawiono wynik z poprzednich przykładów. Dla zwięzłości pokazujemy tylko pierwsze pięć elementów na liście:

```
print("Embedding Length:", len(embeddings))
print("Embedding:", embeddings[:5])
```

Konfiguracja modelu

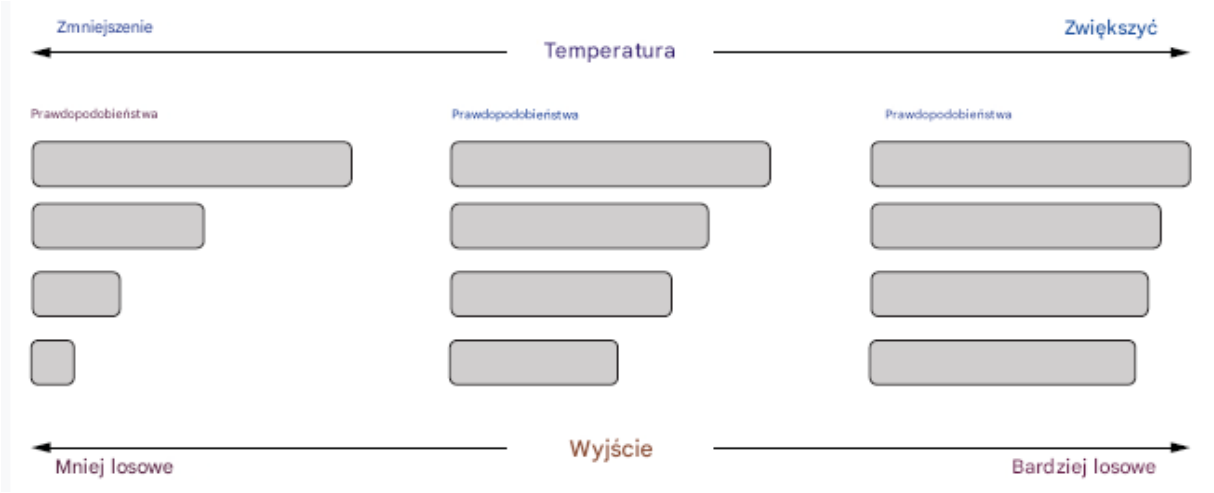
Większość modeli LLM udostępnia użytkownikowi pewne ustawienia konfiguracyjne, umożliwiając mu w pewnym stopniu modyfikowanie sposobu działania modelu i jego zachowania. Chociaż kilka parametrów zmienia się w zależności od implementacji modelu, trzy kluczowe konfiguracje to temperatura, prawdopodobieństwo maksymalne (top_p) i maksymalna odpowiedź. Należy pamiętać, że niektóre implementacje mogą mieć inną nazwę, ale oznaczają to samo. Implementacja GPT w OpenAI nazywa maksymalną odpowiedź maksymalną liczbą tokenów. Przyjrzyjmy się temu bliżej.

MAKSYMALNA ODPOWIEDŹ

Parametr znany jako maksymalna odpowiedź zasadniczo definiuje górny limit długości tekstu generowanego przez model. Oznacza to, że gdy model osiągnie tę z góry określoną długość, zatrzymuje generowanie tekstu, niezależnie od tego, czy jest to środek słowa, czy zdanie. Zrozumienie tej konfiguracji jest kluczowe, ponieważ istnieje limit rozmiaru tokenów, które większość modeli może przetworzyć. Zwiększenie tego rozmiaru wiąże się ze zwiększonymi wymaganiami obliczeniowymi, co prowadzi do wzrostu opóźnień i kosztów.

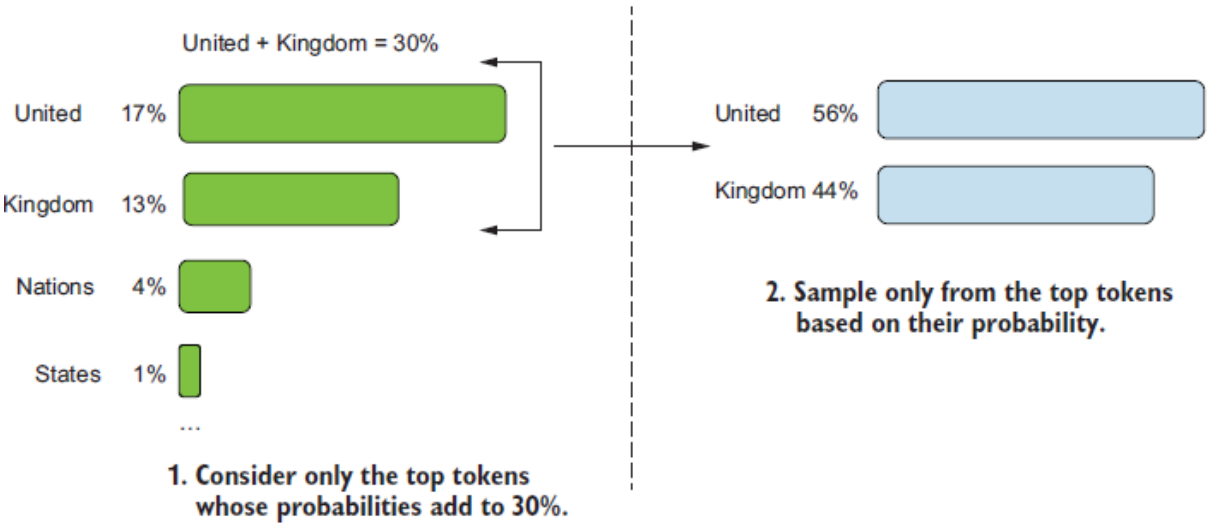
TEMPERATURA

Podczas generowania tekstu, podobnie jak w przypadku każdego modelu podstawowego, naturalna losowość generuje inny wynik przy każdym wywołaniu modelu. Temperatura jest jednym z najważniejszych ustawień kontrolujących stopień losowości modelu. Zazwyczaj jest to wartość z zakresu od 0 do 1, przy czym 0 oznacza dokładniejszy i bardziej przewidywalny wynik. Natomiast ustawienie 1 sprawia, że wynik jest bardziej zróżnicowany i losowy, jak pokazano na rysunku 10.



PRAWDOPODOBIEŃSTWO NAJWYŻSZE (TOP_P)

Parametr prawdopodobieństwa najwyższego (top_p) (znany również jako próbkowanie jądra) to ustawienie w interfejsach API modeli językowych, które steruje losowością procesu generowania tekstu. Ten parametr pozwala precyzyjnie dostroić równowagę między kreatywnością a wiarygodnością w tekście generowanym przez model. Definiuje on prawdopodobieństwo progowe; tylko słowa z prawdopodobieństwami powyżej tego progu są brane pod uwagę podczas generowania tekstu przez model. Podczas generowania tekstu model językowy przewiduje prawdopodobieństwo, że każde słowo będzie kolejnym w sekwencji. Parametr top_p pomaga skrócić ten rozkład prawdopodobieństwa, aby poprawić jakość generowanego tekstu. Na przykład, w celu generowania wyników, ustawienie parametru top_p na niższą wartość (np. 0,3) spowoduje, że model uwzględni tylko 30% najbardziej prawdopodobnych słów dla następnego słowa w sekwencji, jak pokazano na rysunku 11.



Dzięki temu tekst będzie bardziej przewidywalny i mniej zróżnicowany. Jeśli jednak ustawimy parametr top_p na wyższą wartość (np. 0,9), model uwzględni znacznie szerszy zakres słów, w tym te mniej prawdopodobne. Może to prowadzić do bardziej zróżnicowanego i potencjalnie interesującego generowania wyników.

PRZYKŁAD

Pokażmy, jak można programowo wykorzystać te ustawienia. Poniższy fragment kodu pokazuje przykład, jak skonfigurować te ustawienia za pomocą OpenAI. Większość z tych ustawień jest używana oszczędnie, jest opcjonalna i powróci do wartości domyślnych. Tylko niektóre opcje, takie jak max_tokens i temperature, są używane w niemal każdym przypadku użycia:

```
client = OpenAI(api_key='your-API-key')

response = client.completions.create(

model="text-davinci-003",

prompt="...",

temperature=1,

max_tokens=256,

top_p=1,

frequency_penalty=0,

presence_penalty=0

)
```

Ponieważ API jest bezstanowe, ustawienia te mogą się różnić między różnymi instancjami i wywołaniami API tej samej instancji, w zależności od scenariusza biznesowego, który chcemy osiągnąć. Co więcej, nie ma globalnie optymalnych wartości dla tych ustawień, które zależą od zadania. Ogólnie rzecz biorąc, aby uzyskać zbalansowane dane wyjściowe i uniknąć halucynacji modelu, ustawienie temperatury na poziomie 0,7 lub 0,8 jest odpowiednie. Tabela 4 przedstawia ustawienia konfiguracyjne kontrolujące zachowanie.

Konfiguracja: Opis

Maksymalna liczba tokenów: Ustawia limit liczby tokenów na odpowiedź modelu. W zależności od modelu, maksymalny limit jest dzielony między monit (w tym komunikat systemowy, przykłady, historię wiadomości i zapytanie użytkownika) a odpowiedź modelu.

Temperatura: Kontroluje losowość. Obniżenie temperatury oznacza, że model generuje bardziej powtarzalne i deterministyczne odpowiedzi. Podniesienie temperatury skutkuje bardziej nieoczekiwanymi lub kreatywnymi odpowiedziami. Spróbuj dostosować temperaturę lub top_p, ale nie oba. Zazwyczaj, wraz ze wzrostem długości sekwencji, model naturalnie staje się bardziej pewny swoich przewidywań i można użyć znacznie wyższej temperatury dla długich monitów, nie odchodząc od tematu. Z drugiej strony, użycie ustawienia wysokiej temperatury dla krótkich monitów może prowadzić do niestabilnych wyników.

Prawdopodobieństwo najwyższe (top_p): Jest to próg prawdopodobieństwa, który, podobnie jak temperatura, kontroluje losowość, ale wykorzystuje inną metodę. Obniżenie top_p zawęża wybór tokenów modelu do tokenów najbardziej prawdopodobnych i ignoruje długi ogon tokenów mniej prawdopodobnych. Zwiększenie wartości top_p umożliwi modelowi wybór tokenów o wysokim i niskim prawdopodobieństwie wystąpienia. Spróbuj dostosować wartość temperature lub top_p, ale nie obie.

Sekwencje stop: Powoduje to, że model kończy swoją odpowiedź w żądanym punkcie. Odpowiedź modelu zakończy się przed określoną sekwencją, dzięki czemu nie będzie zawierać tekstu sekwencji stop.

Kara częstotliwości: Zmniejsza to prawdopodobieństwo powtórzenia tokena proporcjonalnie do częstotliwości jego występowania w tekście. Zmniejsza to prawdopodobieństwo powtórzenia tego samego tekstu w odpowiedzi.

Kara obecności: Zmniejsza to prawdopodobieństwo powtórzenia dowolnego tokena, który do tej pory pojawił się w tekście. Zwiększa to prawdopodobieństwo wprowadzenia nowych tematów w odpowiedzi.

Okno kontekstowe

Okno kontekstowe to stosunkowo nowa i bardzo ważna koncepcja. Odnosi się ono do zakresu tokenów lub słów otaczających dane słowo lub token, które model LLM uwzględnia podczas tworzenia prognoz. Okno kontekstowe pomaga modelowi zrozumieć zależności i relacje między słowami, umożliwiając generowanie dokładniejszych i spójniejszych prognoz. Na przykład, podczas przewidywania następnego słowa w zdaniu, okno kontekstowe może zawierać kilka słów poprzedzających słowo docelowe. Rozmiar okna kontekstowego może się różnić w zależności od modelu i jego architektury. W modelach LLM okna kontekstowe mogą być dość duże, co pozwala modelowi uchwycić zależności dalekiego zasięgu i złożone relacje semantyczne między słowami. Te okna o dłuższym kontekście mogą pomóc w uzyskaniu lepszych wyników dla zadań takich jak generowanie tekstu, tłumaczenie i podsumowywanie. Obecna architektura LLM ogranicza rozmiar okna kontekstowego do kilku tysięcy tokenów. Chociaż niektóre nowsze modele obsługują do miliona tokenów, okno kontekstowe nadal pozostaje kluczowym punktem centralnym, głównie dlatego, że globalny charakter mechanizmu uwagi generuje koszty obliczeniowe, które są kwadratowe w stosunku do długości kontekstu. Innymi słowy, im większe okna kontekstowe, tym bardziej koszt obliczeń jest proporcjonalny do kwadratu rozmiaru danych wejściowych. Chociaż posiadanie większego okna kontekstowego może wydawać się korzystne, ważne jest, aby zrozumieć, że ma to zarówno pozytywne, jak i negatywne implikacje. Przy większych oknach kontekstowych wydajność modelu jest znacznie niższa zarówno pod względem zrozumienia żądania, jak i jego generowania, a także występuje większe opóźnienie. Chociaż możemy uważać, że dłuższy kontekst byłby lepszy, generalnie należy używać mniejszego okna, jeśli to wystarczy do danego zadania — zapewniłoby to lepszą wydajność niż większe. Niektóre z zalet to:

* Lepsze zrozumienie kontekstu — dłuższe okno kontekstowe pozwala modelowi uchwycić zależności dalekiego zasięgu i złożone relacje semantyczne między słowami, co przekłada się na lepsze przewidywania i bardziej spójny generowanie tekstu. Wiąże się to jednak ze znacznymi kosztami i należy je stosować rozważnie.

* Lepsza wydajność w złożonych zadaniach — dzięki szerszemu oknu kontekstowemu modele językowe mogą lepiej radzić sobie z zadaniami wymagającymi lepszego zrozumienia szerszego kontekstu, takimi jak tłumaczenie maszynowe, podsumowywanie i analiza sentymentu.

Oto wady:

* Większe wymagania obliczeniowe — dłuższe okno kontekstowe wymaga więcej pamięci i mocy obliczeniowej do przetwarzania i przechowywania dodatkowych informacji, co może skutkować dłuższym czasem szkolenia i wnioskowania oraz wymagać bardziej wydajnego sprzętu lub rozproszonych rozwiązań obliczeniowych.

* Potencjał nadmiernego dopasowania — wraz ze wzrostem okna kontekstowego model staje się bardziej złożony i podatny na nadmierne dopasowanie, zwłaszcza jeśli dane treningowe są ograniczone. Nadmierne dopasowanie występuje, gdy model uczy się wyjątkowo dobrze działać na danych treningowych, ale ma trudności z generalizacją nowych i niewidocznych danych.

* Trudności z obsługą bardzo długich sekwencji — chociaż dłuższe okna kontekstowe mogą poprawić wydajność, mogą również stwarzać problemy podczas przetwarzania ekstremalnie długich sekwencji. Niektóre modele mogą mieć trudności z utrzymaniem niezbędnych informacji na tak dużych odległościach, co prowadzi do pogorszenia wydajności.

* Malejące korzyści — chociaż zwiększanie rozmiaru okna kontekstowego może poprawić wydajność, może istnieć punkt malejących korzyści, w którym dalsze zwiększanie rozmiaru okna kontekstowego przyniesie jedynie marginalną poprawę. Aby osiągnąć optymalną wydajność, kluczowe jest zrównoważenie rozmiaru okna kontekstowego z kosztem obliczeniowym i złożonością modelu.

Koncepcja okna kontekstowego jest niezwykle ważna z kilku powodów:

* Rejestruje zależności — okno kontekstowe pozwala modelowi zrozumieć relacje między słowami, frazami lub zdaniami w tekście. Pomaga to modelowi uchwycić ogólne znaczenie i kontekst danych wejściowych.

* Ulepszone przewidywania — to prawdopodobnie to, co większość z nas widzi podczas korzystania z modeli LLM, gdzie okno kontekstowe umożliwia modelowi generowanie dokładniejszych i spójniejszych sugestii na podstawie poprzedniego tekstu.

* Zapewnia kontekst dla lepszego zrozumienia — dzięki uwzględnieniu okna kontekstowego modele LLM mogą lepiej zrozumieć kontekst, a w rezultacie strukturę składniową i relacje semantyczne w tekście; pozwala to na dokładniejsze zrozumienie i generowanie języka.

Inżynieria błyskawiczna

Inżynieria błyskawiczna to stosunkowo nowa dziedzina, która polega na tworzeniu lub projektowaniu komend w celu wywołania pożądanych odpowiedzi lub zachowań z modelu uczenia maszynowego, w szczególności LLM. Jest to potężna technika, która może poprawić wydajność modeli językowych w różnych zadaniach. Inżynieria błyskawiczna to rozwijająca się dziedzina, która wymaga kreatywności i dbałości o szczegóły. Inżynierię błyskawiczną można postrzegać zarówno jako sztukę, jak i naukę. Polega ona na starannym doborze i sformułowaniu poleceń wejściowych, aby pomóc sztucznej inteligencji w generowaniu pożądanych wyników. Te polecenia wejściowe mogą polegać po prostu na doborze odpowiednich słów, fraz i formatów, aby pomóc modelowi w generowaniu wysokiej jakości i trafnych tekstów dla określonego zadania. Na przykład, aby uzyskać szczegółową odpowiedź, możesz użyć polecenia takiego jak „Wyjaśnij szczegółowo…”, lub aby uzyskać szybkie podsumowanie, możesz użyć polecenia „Podsumuj w kilku punktach…”. Podobnie, aby sztuczna inteligencja pisała w stylu konkretnego autora, możesz użyć polecenia takiego jak „Napisz fragment tak, jakbyś był P. G. Wodehouse’em”. Inżynieria podpowiedzi wymaga zrozumienia możliwości modelu sztucznej inteligencji, danych treningowych i sposobu, w jaki reaguje on na różne rodzaje danych wejściowych. Skuteczna inżynieria podpowiedzi może znacząco poprawić użyteczność modeli sztucznej inteligencji w różnych zadaniach. Należy pamiętać, że ta sekcja jest jedynie wprowadzeniem do inżynierii podpowiedzi jako podstawowej koncepcji.

Adaptacja modelu

Modele LLM są wstępnie wytrenowane i ogólnego przeznaczenia, a czasami wymagają dostrojenia. Są trenowane na dużym korpusie danych tekstowych i mogą być używane jako punkt wyjścia do trenowania na mniejszym zbiorze danych dla konkretnego zadania. Dostrojenie bazowego modelu LLM na mniejszym zbiorze danych może poprawić jego wydajność dla tego konkretnego zadania. Dostrajanie polega na przetrenowaniu wstępnie wytrenowanego modelu i dalszym trenowaniu go na nowym zadaniu lub zbiorze danych. Wstępnie wytrenowany model służy jako punkt wyjścia, a wagi modelu są dostosowywane podczas treningu w celu poprawy jego wydajności dla nowego zadania. Dostrajanie jest często stosowane w transferze uczenia, gdzie model wytrenowany na jednym zadaniu jest adaptowany do innego, pokrewnego zadania. Oto kilka przykładów dostrajania LLM:

* Klasyfikacja tekstu — dostrajanie LLM do konkretnego zadania klasyfikacji tekstu, takiego jak analiza sentymentu lub wykrywanie spamu

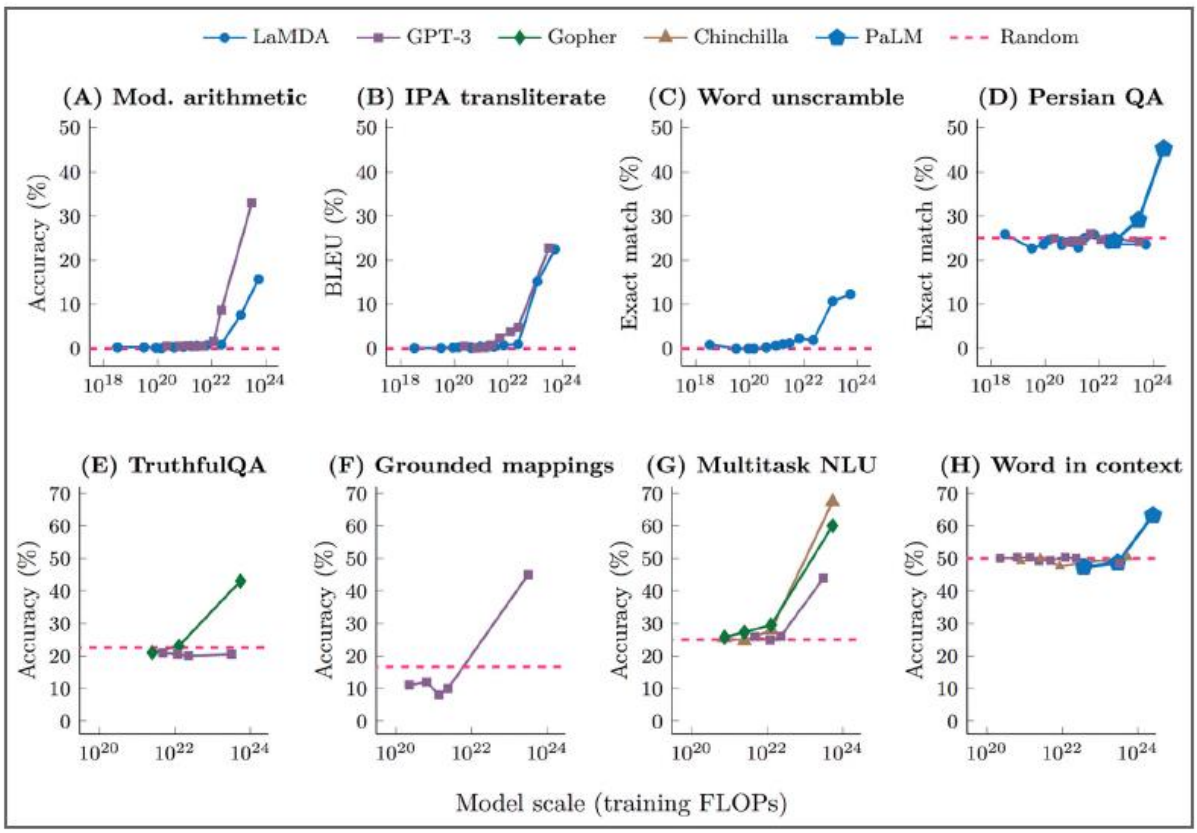
* Odpowiadanie na pytania — dostrajanie LLM do konkretnego zadania związanego z odpowiadaniem na pytania, takiego jak odpowiadanie na pytania dotyczące konkretnego tematu

* Generowanie języka — dostrajanie LLM do konkretnego zadania generowania języka, takiego jak generowanie streszczeń lub tłumaczeń

Dostrajanie LLM, takiego jak GPT-3.5 Turbo, może być skutecznym sposobem na dostosowanie modelu do konkretnych zadań, ale może być również bardzo kosztowne i powinno być jedną z ostatnich opcji. W niektórych przypadkach dostrajanie może również prowadzić do katastrofalnego zapomnienia. Dzieje się tak, gdy model jest dostrajany do nowego zbioru danych, powodując zapomnienie wiedzy zdobytej z oryginalnych danych treningowych, co skutkuje utratą zdolności rozumowania przez dostrojony model. Adaptację modelu omówimy bardziej szczegółowo w rozdziale 9, w tym wszelkie pułapki związane z dostrajaniem.

Zachowania emergentne

Koncepcja zachowań emergentnych definiuje znaczenie modeli podstawowych i modeli LLM. Zachowania emergentne oznaczają nieoczekiwane zachowania modeli LLM podczas interakcji ze środowiskiem, zwłaszcza gdy są trenowane na dużych ilościach danych. Modele LLM nie są wprost trenowane w zakresie tych umiejętności, lecz uczą się ich poprzez obserwację języka naturalnego. Pojawienie się tych zachowań jest indukowane niejawnie, a nie jawnie konstruowane; jest to zarówno źródłem naukowego entuzjazmu, jak i obaw o nieprzewidziane konsekwencje. Jak pokazuje rysunek 12, wydajność modelu w wielu testach porównawczych języka naturalnego (np. w odpowiadaniu na pytania) nie jest lepsza niż losowość, dopóki nie osiągną one określonej skali, mierzonej w tym przykładzie obliczeniami treningowymi w FLOP-ach.



Wydajność modelu gwałtownie rośnie w tym momencie, co jest przykładem zdolności emergentnych. Pomaga nam to zrozumieć, że emergentne właściwości modeli LLM nie występują w mniejszych modelach. Co więcej, zdolności te pojawiają się dopiero w pewnej skali, gdy rozmiar modelu osiągnie określony próg. Właściwości emergentnych nie da się przewidzieć poprzez ekstrapolację wyników mniejszych modeli. Przyjrzyjmy się przykładowi z wykorzystaniem kodu, ponieważ dokładna logika i natura działania fragmentu kodu nie mogą być z góry określone ani zaprogramowane na potrzeby LLM. Aby zrozumieć, przeanalizować i zrozumieć kod, należy wykorzystać emergentne właściwości behawioralne modeli fundamentalnych. Załóżmy, że mamy kod w C, który odwraca ciąg znaków, jak pokazano na poniższym listingu. Wykorzystuje on niektóre z bardziej zaawansowanych funkcji języka C, takich jak wskaźniki, operatory XOR i makra. W naszym przykładzie używamy GPT-4 do konwersji tej funkcji C na Pythona. Pierwszym krokiem byłoby poproszenie naszego LLM o przekonwertowanie tego kodu na Pythona za pomocą wiersza poleceń podobnego do następnego. Ponownie pomijamy funkcję C (listing 4) dla zwięzłości. Kod jest częścią wiersza poleceń.

Listing 4 : Funkcja napisana w języku C, która odwraca ciąg znaków

```
#include <String.h> // for strlen()

// reverse the given null-terminated String in place

void inplace_reverse(char * str)

{

if (str)

{
```

```
char * end = str + strlen(str) - 1;

// swap the values in the two given variables

// XXX: fails when a and b refer to same memory location

# define XOR_SWAP(a,b) do\

{\

a ^= b;\

b ^= a;\

a ^= b;\

} while (0)

// walk inwards from both ends of the String,

// swapping until we get to the middle

while (str < end)

{

XOR_SWAP(*str, *end);

str++;

end--;

}

# undef XOR_SWAP

}

}
```

Konwertuj tę funkcję C na Pythona:

```
#include <string.h> // dla strlen()

// odwróć podany ciąg znaków zakończony znakiem null w odpowiednim miejscu

void inplace_reverse(char * str)

{

…

}
```

Listing 5 przedstawia wynik naszego modelu z kodem przekonwertowanym na Pythona. Dobrze rozumie on działanie funkcji C i opisuje różne ograniczenia i właściwości Pythona w odpowiedzi. Należy zauważyć, że nie podaliśmy żadnych wskazówek co do działania funkcji. Rozszyfrował on samą pętlę while, pokazując przykład zachowania emergentnego.

Ta funkcja zwróci odwrócony ciąg znaków zamiast modyfikować oryginalny ciąg znaków, ponieważ ciągi znaków w Pythonie są niezmienne. Innymi słowy, raz utworzonego ciągu znaków w Pythonie nie można zmienić. W związku z tym funkcja inplace_reverse jest nieco myląca w kontekście Pythona. Możesz rozważyć zmianę jej nazwy na coś w rodzaju reverse_string.

Listing 5 : Przekonwertowana funkcja Pythona do odwracania ciągu znaków

```
def inplace_reverse(str):

if str:

# Convert the String to a list since strings are immutable in Python

lst = list(str)

# Initialize two pointers at the start and end of the list

start = 0

end = len(lst) - 1

# Reverse the list in place

while start < end:

# XOR_SWAP equivalent in Python

lst[start], lst[end] = lst[end], lst[start]

# Move the pointers towards the center of the list

start += 1

end -= 1

# Convert the list back into a string

return ''.join(lst)

return str
```

Możemy to jeszcze bardziej uprościć, zadając modelowi następujące pytanie:

Czy możesz jeszcze bardziej uprościć i zoptymalizować kod Pythona?

Jasne! Jednym z najprostszych sposobów odwrócenia ciągu znaków w Pythonie jest wycinanie. Oto funkcja, która zwróci odwrotną stronę ciągu znaków:

```
def reverse_string(s):

return s[::-1]
```

[::-1] to wycinek, który zaczyna się od końca ciągu znaków i przesuwa się do tyłu.

Jeśli jednak chcesz emulować tę samą logikę, co program w C, czyli odwrócić ciąg znaków w miejscu za pomocą dwóch wskaźników, oto bardziej pythonowy sposób osiągnięcia tego:

```
def reverse_string(s):

lst = list(s)
```

```
lst.reverse()

return ''.join(lst)
```

W tej wersji konwertujemy ciąg znaków na listę, odwracamy listę w miejscu za pomocą wbudowanej funkcji reverse, a następnie łączymy listę z powrotem w ciąg znaków. Funkcja reverse w Pythonie wykorzystuje podobne podejście dwuwskaźnikowe, ale wszystko odbywa się „pod maską”, więc kod wygląda znacznie prostiej. Punkt, w którym odpowiedź odwołuje się do podobnego podejścia dwuwskaźnikowego, demonstruje moc właściwości emergentnej. Nigdzie w rozmowie nie wspomnieliśmy o tym wprost. W tym rozdziale przyjrzeliśmy się podstawowym koncepcjom, które sprawiają, że modele LLM działają, i na co należy zwrócić uwagę, wdrażając te nowe generatywne techniki sztucznej inteligencji w przedsiębiorstwie. Omawiamy nowe koncepcje, takie jak monity, inżynieria promptów, osadzenia, tokeny, parametry modelu, okna kontekstowe, architektura transformatorów i zachowania emergentne. Teraz, gdy w następnym rozdziale mamy już szerokie zrozumienie LLM i ich implikacji, przenieśmy to na grunt praktyczny i wykorzystajmy API do generowania tekstu, zarówno w formie uzupełnień, gdzie podajemy instrukcje za pomocą podpowiedzi, jak i czatu, gdzie możemy prowadzić dialog, generując w ten sposób tekst w kontekście rozmowy.

Podsumowanie

* Duże modele językowe (LLM) stanowią znaczący postęp w dziedzinie sztucznej inteligencji. Są one trenowane na ogromnych ilościach danych tekstowych, aby uczyć się wzorców w języku ludzkim.

* Modele LLM są uniwersalne i mogą obsługiwać zadania bez danych treningowych specyficznych dla danego zadania, takie jak odpowiadanie na pytania, pisanie esejów, streszczanie tekstów, tłumaczenie języków i generowanie kodu.

* Kluczowe przypadki użycia LLM obejmują podsumowania, klasyfikację, pytania i odpowiedzi/chatboty, generowanie treści, analizę danych, tłumaczenie i lokalizację, automatyzację procesów, badania i rozwój, analizę sentymentów i ekstrakcję encji.

* Typy LLM obejmują modele bazowe, oparte na instrukcjach i dostrojone LLM. Każdy z nich ma swoje zalety i wady i jest oparty na modelach fundamentalnych. * Modele fundamentalne to duże modele AI trenowane na ogromnych ilościach danych na ogromną skalę, co skutkuje modelami, które można dostosować do szerokiego zakresu zadań downstream.

* Niektóre kluczowe koncepcje LLM obejmują monity, inżynierię monitów, osadzenia, tokeny, parametry modelu, okna kontekstowe, architekturę transformatorów i zachowania emergentne.

* Modele LLM o otwartym kodzie źródłowym i komercyjne mają swoje zalety i wady, przy czym modele komercyjne zazwyczaj oferują najnowocześniejszą wydajność, a modele o otwartym kodzie źródłowym zapewniają większą elastyczność w zakresie dostosowywania i integracji.

* Modele małych języków (SLM) to nowy, pojawiający się trend w zakresie lekkich, generatywnych modeli AI, które generują tekst, podsumowują dokumenty, tłumaczą języki i odpowiadają na pytania. W niektórych przypadkach oferują one możliwości podobne do tych oferowanych przez większe modele.