Porównanie potencjału generatywnych sieci neuronowych do generowania kodu na przykładzie aplikacji do przeprowadzania ankiet

Motywacja

Sieci neuronowe oraz różnej maści oparte na nich chatboty stały się w ciągu ostatnich kilku lat istotną częścią naszego życia. Znalazły swoje zastosowanie w niemal każdej dziedzinie – od sztuki aż po nauki ekonomiczne i społeczne. Jedną z grup zawodowych wymienianych jako najbardziej prawdopodobne do zastąpienia przez „sztuczną inteligencję", jak zwykło się zbiorczo nazywać te narzędzia, są programiści. Przyczyn takich podejrzeń jest wiele – kod programu jest dość mocno ustrukturyzowany oraz stosunkowo powtarzalny, a jego twórca osobą drogą do opłacenia. Wielu studentów kierunków powiązanych z informatyką oraz docenionych programistów używa narzędzi opartych o sieci neuronowe niemal codziennie w swojej pracy, a coraz więcej firm zachęca swoich pracowników do korzystania z nich. W związku z tym postanowiłem sprawdzić, jak popularne chatboty, czasami nawet reklamowane jako specjalnie dostosowane do potrzeb piszących kod, poradzą sobie z napisaniem stosunkowo prostej aplikacji, z rolą człowieka ograniczającą się do prostego stwierdzenia czy dany program działa zgodnie z założeniami, czy nie.

Cel pracy

Celem moich działań było porównanie potencjału trzech popularnych chatbotów do napisania kodu prostej aplikacji. Jedyną moją ingerencją było napisanie promptu dla sieci neuronowej, uruchomienie otrzymanego programu i ewentualne wytknięcie błędów. Chatboty, które poddałem próbie to ChatGPT firmy OpenAI, DeepSeek firmy o tej samej nazwie oraz Claude, stworzony przez Anthropic. Mierzona była przede wszystkim konieczna liczba interakcji niezbędna do osiągnięcia zakładanego celu i parametry kodu, a wszystkie badania były przeprowadzane na darmowych wersjach chatbotów.

Wykorzystane metody i narzędzi

Aby otrzymać możliwie najbardziej satysfakcjonujące efekty, starałem się wdrożyć kilka metod zalecanych podczas interakcji z sieciami neuronowymi działającymi na promptach. Początkowym planem było zindywidualizowanie podejścia do każdego z badanych chatbotów, jednak ku mojemu rozczarowaniu byłem w stanie znaleźć jedynie szkolenia i poradniki dotyczące korzystania z ChataGPT. W związku z tym zdecydowałem się zastosować te zasady do wszystkich sieci neuronowych; opierały się one na kilku filarach:

1. Dokładne określenie używanych technologii, włącznie z podaniem bibliotek i wersji – podstawowa technika, pozwalająca określić ogólny zarys problemu.
2. Możliwie najbardziej szczegółowe opisywanie problemu oraz wymagań – inna podstawowa technika, ograniczająca ewentualne niespodziewane zmiany ze strony sieci.
3. Prośba o wcielanie się w rolę – metoda zalecana jeśli naszym celem jest otrzymanie kodu możliwie najwyższej jakości; idąc z tym duchem, każdy prompt zaczynałem od sformułowania „Acting as a senior developer with over ten years of work…".

Sposób rozwiązania problemu

Zdecydowałem się przeprowadzić trzy badania na każdej z sieci:

1. Pierwsze badanie dotyczyło zdolności sieci do wygenerowania kodu od podstaw na bazie pojedynczego promptu. W tym badaniu wszystkie sieci dostawały za każdym razem dokładnie ten sam prompt w nowej konwersacji.
2. Kolejne dotyczyło zdolności sieci do wprowadzania modyfikacji do już istniejącego kodu oraz ulepszania stworzonego przez siebie programu. Tutaj prompty były tworzone na bieżąco w zależności od aktualnych problemów konkretnego kodu.

3. Ostatnie sprawdzało, czy zachęcone sieci będą zadawały pytania na temat otrzymanego promptu w celu jak najlepszego spełnienia założeń. W tym badaniu również prompty były tworzone na bieżąco w zależności od potrzeb.

W każdym badaniu celem chatbotów było stworzenie prostej aplikacji do zbierania ankiet w frameworku streamlit w wersji 1.44.0. Aplikacja ta miała być wyposażona w dwa tryby, w których pokazywane byłyby dokładnie te same pytania. Pierwszy z trybów („tryb timera"), miał być wyposażony w widoczny licznik czasu, po upłynięciu którego program powinien przejść bezwzględnie do kolejnego pytania. Drugi z nich („tryb zrelaksowany") miał być pozbawiony jakichkolwiek ograniczeń czasowych i umożliwiać swobodne poruszanie się między pytaniami, podczas gdy aplikacja miała mierzyć czas spędzony nad każdym z nich.

Pierwsze badanie

W pierwszym badaniu największym problemem okazało się poprawne opisanie timera, który miał przyjąć postać skracającego się stopniowo paska. Timer ten często powodował błędy w działaniu, do których należały zazwyczaj brak automatycznego odświeżania (często odświeżanie następowało po interakcji z elementem aplikacji) oraz restart po zmianie odpowiedzi. Po każdej z nieudanych prób modyfikowałem odpowiednio prompt i spisywałem zalety oraz wady każdego z otrzymanych kodów.

Po trzech modyfikacjach promptów żadnej z sieci nie udało się spełnić całkowicie postawionych przed nimi założeń. Jedyną siecią, która poprawnie zaimplementowała pasek timera był DeepSeek; niestety przygotowany przez niego kod nie pokazywał odpowiedzi do pytań w pierwszej części. Zarówno ChatGPT, jak Cladue spotkały się z tym samym problemem – ich timery restartowały się po zaznaczeniu dowolnej z odpowiedzi. Dodatkowo, aplikacja przygotowana przez ChatGPT nie dokonywała pomiarów czasu spędzonego na każdym pytaniu w części „zrelaksowanej", a aplikacja Claude potrafiła zwrócić błąd przy próbie powrotu do wcześniejszych pytań w drugiej części ankiety.

Drugie badanie

W drugim badaniu przekazałem sieciom ich kod napisany w ostatniej próbie pierwszego badania, jako że w każdym przypadku był najbardziej udany. Czasami podczas badania zdarzało się, że pomimo prośby o modyfikację, sieć zwracała program cierpiący na te same problemy, jak ten dostarczony – w takiej sytuacji przede wszystkim starałem się użyć innych sformułowań; sytuacja ta dotyczyła przede wszystkim Claude i DeepSeeka. Ponieważ żadnemu z chatbotów nie udało się w pełni napisać aplikacji przy pomocy pojedynczego promptu, pierwsze modyfikacje kodu skupiały się na poprawnej implementacji podstawowej wersji. Najszybciej udało się to ChatowiGPT – całkowicie działającą aplikację otrzymałem już po drugiej modyfikacji kodu. Nieco dłużej zadanie to zajęło DeepSeekowi, który potrzebował czterech interakcji. Dużym zawodem był Claude; mimo ośmiu interakcji, sieci nie udało się naprawić paska timera, aby działał zgodnie z postawionymi założeniami. Po tym czasie uznałem, że chatbot jest niezdolny do wprowadzenia zakładanych poprawek i przeszedłem do dalszej części badania.

Modyfikacje polegały na zmianie koloru czcionki i tła (wszystkie sieci podołały temu wyzwaniu), wprowadzeniu pytań otwartych oraz wielokrotnego wyboru (ponownie, wszystkie sieci poradziły sobie z tym zadaniem, jednak ze względu na dobór kolorów odpowiedzi na pytania wielokrotnego wyboru DeepSeeka oraz Claude były nieczytelne/prawie nieczytelne), a także wprowadzenie pytań warunkowych, gdzie wybranie jednej, konkretnej opcji powodowałoby pojawienie się dodatkowego pytania, w innym przypadku niewidocznego (tylko ChatGPT wykonał tę część poprawnie). Każdy z chatbotów miał przewidziane sześć interakcji na wprowadzenie tych zmian. Przez cały czas trwania badania DeepSeek oraz Claude chciały pisać swój kod od początku, co w pewnym momencie doprowadziło do tego, że musiałem precyzować w promptach, aby pisana była jedynie zmieniająca się część – inaczej z powodu objętości sieć odmawiała napisania jakiegokolwiek kodu. Zastanawiająca jest również różnica objętości między poszczególnymi chatbotami – kod napisany przez ChatGPT

składał się z mniej niż 280 linijek, kod DeepSeeka był znacznie dłuższy – ponad 600 linijek, natomiast kod Claude był nieco krótszy niż 1000 linijek.

Inną kwestią, która zwróciła moją uwagę była wydajność poszczególnych sieci. O ile ze strony ChatGPT nie natrafiłem na żadne ograniczenia w ilości wysyłanych zapytań, o tyle Claude miał tendencję by przerywać konwersację po 2-3 interakcjach, zachęcając do rozpoczęcia nowej, a DeepSeek wielokrotnie przerywał konwersację na niesprecyzowany czas, tłumacząc to obciążeniem sieci.

Trzecie badanie

Ostatnie badanie było swoistym połączeniem dwóch pozostałych. Chatboty otrzymały prompt początkowy oraz kolejne w zależności od występujących problemów. Główną zmianą w podejściu w porównaniu do poprzednich prób, było zachęcanie sieci w każdym zapytaniu do zadawania ewentualnych pytań, jeśli jakaś część opisu byłaby niejasna (używałem formułki „If something is not clear, ask follow-up questions.”). Ku mojemu rozczarowaniu, żadna z sieci nie zadała dodatkowych pytań dotyczących kodu. Jeśli jakieś ewentualnie się pojawiały, dotyczyły chęci pomocy w rozwijaniu aplikacji (ChatGPT), lub propozycji wyjaśnienia wprowadzonych zmian (Claude).

Jeśli chodzi o działanie samej aplikacji, kod zwrócony przez ChatGPT spełniał wszystkie postawione założenia po trzeciej interakcji, kod DeepSeek doszedł do tego samego etapu po dwóch interakcjach. Po trzech interakcjach Claude zwrócił kod, który zwracał błędy już przy uruchomieniu – sieć nie poprawiła tego mimo mojego kilkukrotnego zwracania uwagi oraz zaprezentowania całości komunikatu pojawiającego się przy błędzie.

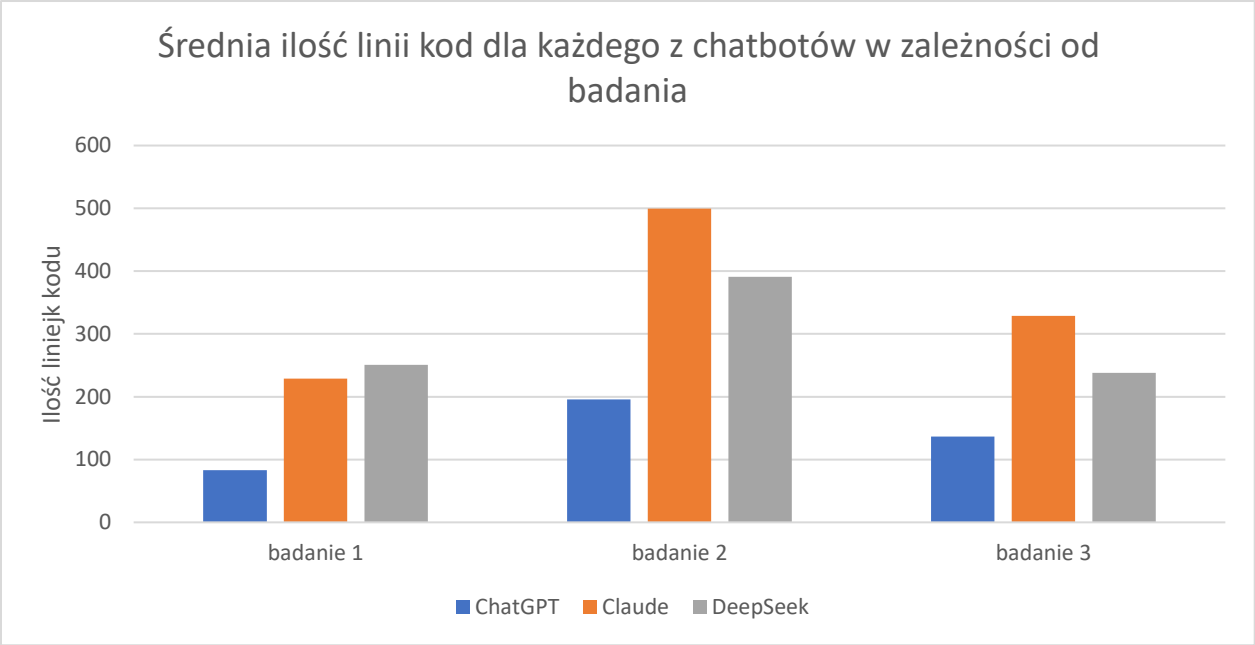
Chciałem wspomnieć również o przypadku, z którym miałem tylko raz do czynienia podczas całości badań. W badaniu trzecim, podczas drugiej interakcji sieć Claude dwukrotnie zaczynała pisać kod od nowa, za każdym razem wskazując wady wcześniej implementowanego rozwiązania. Wpierw chciała napisać kod odpowiadający za poprawne działanie paska timera w JavaScripcie, by następnie porzucić to rozwiązanie na rzecz kodu całkowicie napisanego z użyciem biblioteki streamlit.

Rezultaty

W swojej analizie końcowej zdecydowałem się przede wszystkim skupić na ilości linii kodu, zwróconego przez każdą z sieci. Na takie podejście zdecydowałem się z kilku powodów:

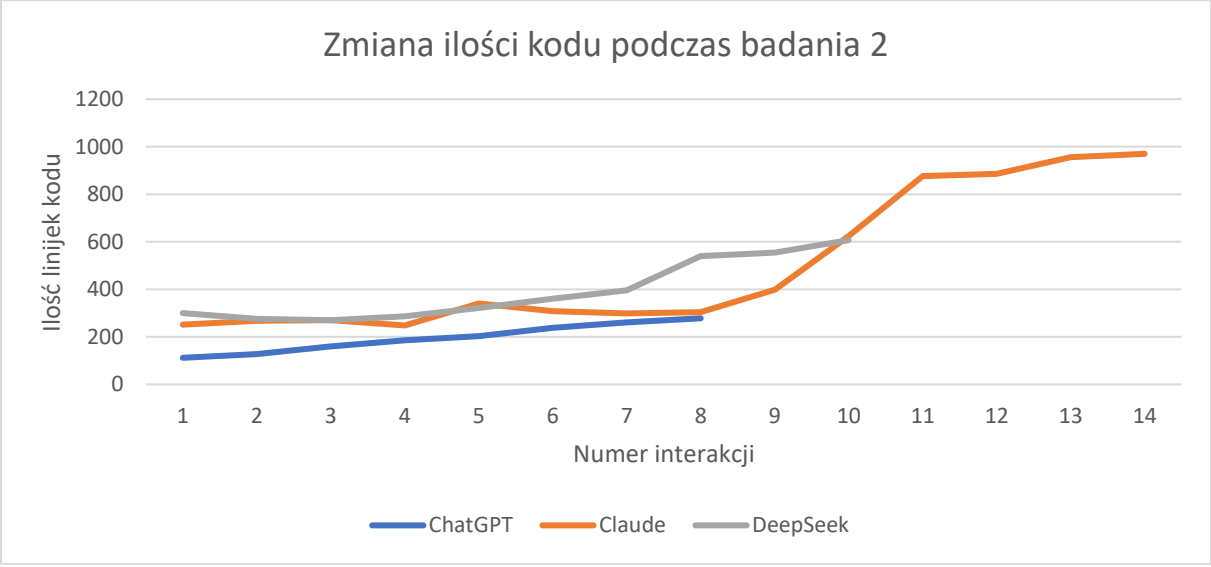
1. Krótszy kod jest wizualnie bardziej przejrzysty i łatwiejszy do przeanalizowania przez informatyka, co w przypadku nowego programu jest istotną kwestią.
2. Dobre praktyki informatyczne zachęcają do pisania możliwe najkrótszych programów – krótszy kod świadczy o wyższym poziomie umiejętności.
3. Niemożność zmierzenia czasu działania – jest to spowodowane prostym faktem, że aplikacja z założenia skupia się na pomiarze czasu i ma z góry założone momenty, na których użytkownik spędza zawsze tyle samo czasu; dodatkowo stworzone przez sieci aplikacje często różnią się ilością pytań zadawanych użytkownikowi oraz ilością czasu przeznaczonego na odpowiedź.

Zliczając ilość kodu otrzymanego w każdej interakcji z każdą z sieci skonstruowałem poniższe wykresy:



Jak widać powyżej, w pierwszym badaniu średnio najwięcej kodu na interakcję produkował DeepSeek. W dwóch pozostałych palma pierwszeństwa przeszła na sieć Claude. W każdym badaniu najkrótszy kod zwrócił ChatGPT.

Następnie postanowiłem sprawdzić, jak zmieniała się liczba linijek kodu w najbardziej rozbudowanym badaniu drugim:



Jak widać powyżej, początkowo najdłuższy był kod DeepSeeka, który został w dziesiątej iteracji przegoniony przez kod Claude. W żadnym momencie kod ChataGPT nie był dłuższy od kodu innej sieci.

Z moich personalnych doświadczeń, najwygodniej używało się ChataGPT. Jego używanie było bardzo intuicyjne, sieć nie miała niespodziewanych problemów jak DeepSeek, który regularnie kazał czekać na ponowienie konwersacji, czy Claude, który do tego problemu bardzo szybko wyczerpywał ilość informacji na jedną konwersację. Dodatkowo, jak pokazałem powyżej, ChatGPT wykonał wszystkie z zadań najszybciej.

Kierunki rozwoju

Przestawiony projekt można rozwinąć w szeregu różnych kierunków:

- Analiza działania każdej z sieci i sformułowanie zindywidualizowanych zasad interakcji – przedstawione w badaniach wyniki zostały osiągnięte przy pomocy zasad sformułowanych pod działania ChataGPT; być może inne sieci zwracałyby lepsze wyniki pod wpływem innego podejścia.
- Sprawdzenie efektów na większej liczbie zróżnicowanych technologicznie aplikacji – możliwym jest, że część sieci była niezaznajomiona, bądź słabo zaznajomiona z technologią frameworka streamlit. W takim wypadku zlecenie napisania większej liczby aplikacji o zróżnicowanych technologiach i poziomach trudności powinno lepiej wykazać potencjalne słabe i mocne strony każdej z sieci.
- Uwzględnienie większej liczby sieci – w tym badaniu zawarłem tylko trzy z najpopularniejszych pośród programistów chatbotów, ale ich liczba jest znacznie większa; możliwe, że najbardziej skuteczny jeszcze nie został przeze mnie odkryty.