

**Nauka programowania z ChatGPT:
skrót do sukcesu czy droga na manowce?**

Ewa Figielska *

Warszawska Wyższa Szkoła Informatyki, Polska
efigielska@ms.wysi.edu.pl

Streszczenie

W artykule przeanalizowano korzyści i zagrożenia związane z wykorzystaniem ChatGPT w nauce programowania. Przedstawiono krótki przegląd historii komputerów i sztucznej inteligencji oraz przykłady rozwiązań programistycznych uzyskanych za pomocą chatbota. Zaproponowano sposoby na motywowanie studentów do aktywnego programowania, zamiast polegania wyłącznie na kodzie generowanym przez sztuczną inteligencję.

Słowa kluczowe – sztuczna inteligencja, ChatGPT, nauczanie programowania

* E-mail: efigielska@ms.wysi.edu.pl

1. Wprowadzenie

W roku 1957 Herbert Simon, amerykański uczony, laureat nagrody Nobla w dziedzinie ekonomii, psycholog kognitywny i pionier w dziedzinie sztucznej inteligencji, powiedział: „Istnieją obecnie na świecie maszyny, które myślą, uczą się i tworzą. Co więcej, ich zdolność do wykonywania tych czynności będzie szybko wzrastać, aż w niedalekiej przyszłości zakres problemów, które będą w stanie rozwiązywać, będzie równy zakresowi, do którego stosowany jest ludzki umysł”¹ [1].

Niniejszy artykuł omawia problemy związane z używaniem modelu językowego sztucznej inteligencji, ChatGPT, w nauce programowania. W początkowych rozdziałach przedstawiono rys historyczny rozwoju komputerów i sztucznej inteligencji, co pozwala lepiej zrozumieć obecny etap rozwoju inteligentnych maszyn oraz kierunek, w którym zmierzamy. W dalszych częściach artykułu skupiono się na wpływie, jaki pojawienie się ChatGPT wywarło na proces nauczania i uczenia się programowania, szczególnie na pierwszym roku studiów. Zawarto także sugestie, jak motywować studentów do świadomej i aktywnej pracy z kodem, zamiast polegać wyłącznie na rozwiązaniach generowanych przez sztuczną inteligencję.

2. Komputery i ich droga „pod strzechy”

Już w starożytności powstały pierwsze przyrządy ułatwiające wykonywanie obliczeń zwane abakusami. Miały postać drewnianych, kamiennych lub metalowych tabliczek z wyżłobionymi liniami reprezentującymi potęgę dziesięciu, liczby ujemne, ułamki. Wzdłuż linii przesuwano kamyki lub metalowe żetony, pełniące rolę liczników, których pozycje wskazywały wartości liczbowe. Te przenośne przyrządy stosowali powszechnie kupcy i inżynierowie [2, 3].

W XVII wieku zostały skonstruowane pierwsze mechaniczne maszyny liczące. Jedną z nich, mianowicie maszyną zaprojektowaną przez Blaise’a Pascala dla ułatwienia pracy swojemu ojcu – poborcy podatkowemu, została zbudowana aż w 50 egzemplarzach, z czego kilka udało się sprzedać [4].

W XIX wieku Charles Babbage zaprojektował urządzenie mechaniczne, zwane maszyną analityczną, które miało pozwolić na wykonywanie serii operacji (w tym

¹ Tłumaczenie z języka angielskiego: ChatGPT-4o

rozgałęzień i skoków warunkowych) na podstawie dostarczonego programu. Obliczenia miały być przeprowadzane przez element centralny maszyny (młyn), a liczby przechowywane w magazynie. Pełna maszyna nie została zbudowana, powstała tylko część młyna. Programowalność maszyny analitycznej oraz jej struktura – rozdzielanie pamięci i jednostki obliczeniowej – powodują, że uważa się ją za pierwszy komputer ogólnego użytku [4].

Na początku XX wieku nastąpił intensywny rozwój telefonii, radiotelegrafii i radiofonii spowodowany wynalezieniem lamp elektronowych: diody (John Ambrose Fleming, 1904) i przede wszystkim triody (Lee De Forest, 1906), która ma zdolność wzmacniania sygnału i przełączania prądu w obwodach logicznych [5]. W roku 1939 Atanasoff i Berry zaprezentowali maszynę liczącą, w której lampy elektronowe zastosowane zostały jako podstawowy element do wykonywania obliczeń [6]. Maszyna ta jest uważana za pierwszy działający prototyp komputera, chociaż nie wykonywała rozgałęzień warunkowych, nie wspierała programowania ogólnego przeznaczenia, a jej działanie było ograniczone tylko do rozwiązywania konkretnych problemów algebraicznych za pomocą stałego zestawu operacji.

W latach 40. XX rozpoczął się rozwój w pełni programowalnych elektronicznych komputerów, motywowany potrzebami wojskowymi. Powstały wtedy amerykański ENIAC [7] i brytyjski Colossus [8]. W latach pięćdziesiątych rozpoczęła się komercyjna produkcja komputerów (np. UNIVAC I [9], IBM 701 [10]). Pojedynczy komputer zawierał wiele tysięcy lamp elektronowych. Lampy miały dość znaczne rozmiary (kilka centymetrów długości), zużywały dużo energii i wydzielały dużo ciepła, co wymagało stosowania ciężkich transformatorów oraz odpowiednio mocnych systemów chłodzenia. Dlatego też pierwsze komputery były wielkimi maszynami, zajmującymi po kilkadziesiąt metrów kwadratowych i ważącymi po kilka ton, używanymi wyłącznie do specjalistycznych zadań związanych z obliczeniami militarnymi, naukowymi, inżynieryjnymi i biznesowymi.

W roku 1947 zaprezentowany został pierwszy tranzystor skonstruowany w laboratoriach Bella przez Johna Bardeena, Waltera Brattaina i Williama Shockley'a [11]. Tranzystory, tak jak lampy elektronowe działają jako przełączniki i wzmacniacze, ale zbudowane są z materiałów półprzewodnikowych, np. krzemu, dzięki czemu mają małe rozmiary, są energooszczędne i trwałe. W latach 1953/54 zostały zaprezentowane pierwsze komputery zbudowane z użyciem tranzystorów (umieszczanych na płytkach drukowanych), a pod koniec lat pięćdziesiątych zaczęła się ich komercyjna

produkcja. Komputery te były mniejsze, szybsze i trwalsze niż maszyny z lampami elektronowymi, ale wciąż dość duże (komputer zajmował kilka szaf) i na tyle drogie, że mogły sobie na nie pozwolić tylko większe instytucje i firmy. Używane były do zastosowań naukowych, inżynierskich i biznesowych [12, 13].

W połowie lat 60. XX wieku zaczęły powstawać komputery oparte na układach scalonych. Układ scalony składa z wielu połączonych ze sobą tranzystorów i innych komponentów elektronicznych uformowanych na jednym małym kawałku materiału półprzewodnikowego, najczęściej krzemu. Pierwsze działające układy scalone zostały zbudowane pod koniec lat pięćdziesiątych [14, 15]: najpierw na bazie germanu z komponentami połączonymi drutami (Jack Kilby, 1958), a następnie na bazie krzemu ze zintegrowanymi na jednej płytce komponentami i ich połączeniami (Robert Noyce, 1959). Użycie układów scalonych pozwoliło na dalsze zmniejszenie rozmiarów komputerów (do rozmiaru pojedynczej szafy). Do interakcji z użytkownikiem zaczęto wtedy używać klawiatury i monitorów.

W roku 1971 wydany został pierwszy mikroprocesor, którego architektem był Marcin „Ted” Hoff [16, 17]. Mikroprocesor to jeden układ scalony wykonujący wszystkie funkcje jednostki centralnej komputera: instrukcje programowe, przetwarzanie danych i zarządzanie operacjami wejścia/wyjścia (przedtem jednostki centralne były budowane z wielu układów scalonych). Wprowadzenie mikroprocesorów zrewolucjonizowało przemysł komputerowy umożliwiając tworzenie małych (do postawienia na biurku), bardziej wydajnych i tańszych komputerów. Wtedy to rozpoczęto produkcję pierwszych komputerów osobistych, tzn. komputerów przeznaczonych do używania przez jedną osobę na raz. Były one wykorzystywane do nauki programowania, prowadzenia badań naukowych, prac biurowych, przechowywania danych oraz do zabawy, kiedy to po komercyjnym sukcesie gry Pong w 1972 roku [18] rozpoczął się intensywny rozwój gier komputerowych.

Na przełomie lat 60. i 70. ubiegłego wieku powstały pierwsze połączenia między komputerami – sieci komputerowe [19], a w 1971 została wysłana pierwsza wiadomość e-mail między komputerami w tej samej sieci [20]. W roku 1983 wprowadzono protokoły komunikacyjne (TCP/IP) jako standardy do przesyłania danych między urządzeniami, co uważa się za narodziny nowoczesnego Internetu [21]. Opracowanie i publiczne udostępnienie na przełomie lat 80. i 90. sieci WWW (ang. *World Wide Web*), umożliwiło łatwy dostęp do informacji za pośrednictwem przeglądarki internetowej, co przyczyniło się do popularyzacji Internetu.

Dalszy postęp technologiczny, który doprowadził do pojawienia się przenośnych komputerów, takich jak laptopy (1981), tablety (1989), smartfony (1994) i nowoczesne smartfony (2007), oraz rozwój Internetu i jego usług, sprawił, że komputery stały się nieodłącznym elementem pracy, życia codziennego i rozrywki. Szacuje się, że w 2018 roku ponad 90% gospodarstw domowych w Stanach Zjednoczonych miało komputer w jakiejś formie (włączając smartfony) [22]. W Polsce było to prawie 90% gospodarstw domowych w 2022 roku [23].

3. Inteligencja maszyn

Początki sztucznej inteligencji (ang. *Artificial Intelligence, AI*) sięgają połowy XX wieku, chociaż problem inteligencji maszyn rozważany był wcześniej. Na przykład w XVII w (przypomnijmy, że był to wiek, kiedy zaczęto konstruować mechaniczne maszyny liczące) Kartezjusz twierdził, że ludzka inteligencja ma uniwersalną zdolność do działania w każdej sytuacji, podczas gdy inteligencja maszynowa to jedynie zbiór umiejętności, które można zastosować tylko w określonych sytuacjach, jakie twórca mógł przewidzieć podczas budowy maszyny [24].

W roku 1950 Alan Turing opublikował artykuł „Computing Machinery and Intelligence”, w którym postawił problem „Czy maszyny mogą myśleć?” [25] i zaproponował jego rozwiązanie w postaci testu, nazywanego testem Turinga. Test ten można w skrócie opisać następująco. Mamy trzy terminale: jeden jest obsługiwany przez komputer, a dwa przez ludzi. Jedna z osób pełni rolę pytającego, a druga osoba i komputer są respondentami. Pytający zadaje pytania w określonym obszarze tematycznym, a następnie, na podstawie udzielonych odpowiedzi, decyduje który z respondentów jest człowiekiem, a który komputerem. Jeśli pytający dokonuje prawidłowego rozróżnienia jedynie w odpowiednio małej liczbie przypadków, komputer jest uznawany za posiadający sztuczną inteligencję.

Sam termin „sztuczna inteligencja” został wprowadzony w 1956 roku przez Johna McCarthy’ego (opracował on dwa lata później język Lisp, który stał się wiodącym językiem programowania AI przez następne 3 dekady). McCarthy zdefiniował sztuczną inteligencję jako „naukę i inżynierię tworzenia inteligentnych maszyn” [26, 27].

3.1. Nadzieje i rozczarowania

W latach 50. i na początku lat 60. XX wieku entuzjazm do badań nad sztuczną inteligencją był duży. Powstały programy do rozwiązywania problemów logicznych i zadań tekstowych z algebry, gry w szachy, a także uczące się mówić po angielsku. W latach 1964-1966 Joseph Weizenbaum, pracujący w laboratorium AI w Massachusetts Institute of Technology, napisał jeden z pierwszych chatbotów, o nazwie Eliza [28]. Ogólnie, chatbot (ang. *chat* – rozmowa, *bot* – skrót od słowa robot) to program przetwarzania języka naturalnego (ang. *Natural Language Processing*, NLP) przeznaczony do prowadzenia rozmowy z człowiekiem. Eliza używając metody "dopasowywania wzorców" i podstawiania, generowała gotowe odpowiedzi, które sprawiały, że ówczesnym użytkownikom wydawało się, iż rozmawiają z kimś, kto rozumie ich wypowiedzi. Powstało wiele wersji tego chatbota, w tym wersja DOCTOR, zaprojektowana tak, aby odpowiadać jak psychoterapeuta prowadzący terapię w stylu zorientowanym na klienta – terapeuta „odzwierciedlał” pytania, zadając je z powrotem pacjentowi. Na rysunku 1 pokazana jest przykładowa konwersacja z chatbotem Eliza.

Eliza:	Hello, I am Eliza. I'll be your therapist today.
Me:	Hello, my name is Ewa.
Eliza:	We don't need names here.
Me:	Oh, sorry.
Eliza:	Apologies are not necessary.
Me:	Oh, I don't know what to say.
Eliza:	Don't you really know what to say?
Me:	No, I came for help, and you distressed me a little at the very beginning of our meeting.
Eliza:	Oh... Let's move on to something else for a bit. distressed you a little at the very beginning of our meeting?

Rysunek 1. Przykład konwersacji z chatbotem Eliza

Oczekiwania wobec systemów AI były wysokie, a szczególnie duże nadzieje, zwłaszcza w zakresie rozpoznawania wzorców i uczenia się na podstawie doświadczeń, wiązano z perceptronem. Perceptron, opracowany przez Franka Rosenblatta w 1957, był pierwszą, prostą sztuczną siecią neuronową o jednej warstwie; potrafił klasyfikować dane wejściowe do dwóch kategorii i miał zdolność uczenia się [29]. Jednak już pod koniec lat 60. okazało się, że perceptron nie poradzi sobie z bardziej złożonymi wzorcami (takimi, których w przestrzeni danych nie można oddzielić linią prostą), a komputery mają za małe moce obliczeniowe i możliwości pamięciowe, aby

wykonywać złożone obliczenia AI. Spowodowało to zahamowanie rozwoju systemów stosujących sztuczną inteligencję na dziesięć lat.

W latach 80. XX wieku sytuacja się zmieniła dzięki postępom w algorytmice i oprogramowaniu oraz zwiększeniu możliwości obliczeniowych komputerów. Rozwinęły się systemy eksperckie. System ekspercki to program, który najpierw uczy się od człowieka - eksperta w danej dziedzinie, jak postępować w różnych sytuacjach, a następnie udziela porad osobom bez specjalistycznej wiedzy. Systemy te były dość powszechnie stosowane, zwłaszcza w przemyśle (np. do automatyzacji procesów, optymalizacji produkcji). Podjęto również prace nad sztucznymi sieciami neuronowymi o wielu warstwach, które mają większe możliwości niż sieci jednowarstwowe (np. mogą modelować zależności nieliniowe). Koncepcja wielowarstwowej sieci neuronowej pojawiła się już w latach 60., ale wtedy nie był znany algorytm jej uczenia. W roku 1974 Paul Werbos w swojej pracy doktorskiej opisał proces trenowania sieci wielowarstwowej przez wsteczną propagację błędów [30]. Wtedy, na algorytm ten nikt nie zwrócił uwagi; został on „ponownie odkryty” w roku 1986 przez Davida Rumelharta [31]. Chociaż sieci neuronowe znalazły zastosowanie w praktyce w wielu dziedzinach, to jednak na początku lat 90. prace badawcze nad ich rozwojem znacznie zwolniły. Powodem tego były w dalszym ciągu zbyt wolne komputery, za małe zbiory danych z etykietami (tzn. zbiory zawierające zarówno dane wejściowe jak i wyniki) oraz zbyt mała liczba warstw w sieciach, co nie pozwalało na rozwinięcie ich pełnych możliwości.

3.2. Rozkwit

Przeszkody te jednak zaczęły z czasem zanikać. Dzięki rozwojowi technologii produkcji układów scalonych (mikroczipów) zwiększyła się moc i pamięć komputerów, np. układ scalony z lat 70. zawierał tysiące tranzystorów, w latach 90. – już parę milionów, a obecnie – ponad 10 miliardów. Rozwój Internetu przyczynił się do powstania ogromnych i złożonych zbiorów danych, tzw. big data, pochodzących z różnych dziedzin, zawierających teksty, wideo, głos, obrazy, elektroniczne zapisy medyczne itp. Zbiory te umożliwiły efektywne trenowanie i ocenę budowanych modeli AI, a przez to stymulowały rozwój nowych, lepszych modeli. W tworzonych sieciach neuronowych liczba warstw stawała się coraz większa, obecnie zwykle jest ich kilkaset lub nawet kilka tysięcy. Sieci takie nazywa się głębokimi, a proces ich uczenia

– uczeniem głębokim (ang. *deep learning*). Jedną z zalet modeli uczenia głębokiego jest to, że mogą one wyodrębnić cechy, właściwości i relacje niezbędne do generowania dokładnych wyników z surowych, nieobrobionych danych. Potrafią efektywnie pracować z różnorodnymi typami danych, takimi jak obrazy, dźwięki i teksty. Modele te mogą również oceniać i udoskonalać swoje wyniki, aby zwiększyć ich precyzję.

Rozwinęły się różne typy głębokich sieci neuronowych do różnych zastosowań. Poniżej krótko przedstawione zostaną cztery z nich: konwolucyjne sieci neuronowe, generatywne sieci współzawodniczące, sieci rekurencyjne i sieci transformatorowe.

Do automatycznego rozpoznawania i klasyfikacji wzorców w danych, takich jak obrazy, nagrania wideo, czy sygnały dźwiękowe, stosowane są konwolucyjne sieci neuronowe (ang. *Convolutional Neural Networks*, CNN) Początkowe warstwy CNN wykrywają podstawowe cechy, np. krawędzie i rogi, a kolejne warstwy łączą te informacje, aby rozpoznawać bardziej złożone kształty i wzory, np. litery, znaki drogowe, twarze. Konwolucyjne sieci neuronowe zostały wprowadzone w 1989 roku przez Yanna LeCuna i jego zespół w zastosowaniu do rozpoznawania ręcznie pisanych cyfr [32, 33] (czego jednym z bezpośrednich skutków jest sortowanie poczty przez maszyny używające technik automatycznego rozpoznawania pisma ręcznego do odczytywania adresów). W roku 2012 Geoffrey Hinton i jego współpracownicy zaproponowali architekturę o nazwie AlexNet, która osiągnęła niespotykaną do tamtej pory dokładność klasyfikacji na zbiorze ImageNet (który powstał w 2009 i zgromadził miliony oznaczonych obrazów, tzn. obrazów z etykietami). Miało to ogromny wpływ na dalszy rozwój głębokiego uczenia [34]. AlexNet była jedną z pierwszych sieci, w których trenowaniu wykorzystano GPU (ang. *Graphics Processing Unit*), co pozwoliło na szybsze przetwarzanie danych. Przykładami współczesnych zastosowań CNN mogą być: w marketingu identyfikacja osób na zdjęciach na profilach platform społecznościowych, w medycynie identyfikacja nowotworów na podstawie wyników badań radiologicznych, w motoryzacji wykrywanie linii pasa ruchu samochodu w celu podniesienia bezpieczeństwa [35].

W 2014 roku Ian Goodfellow wraz z zespołem wprowadzili generatywne sieci współzawodniczące (ang. *Generative Adversarial Networks*, GANs) [36]. Sieci te używane są do generowania wysokiej jakości obrazów, np. postaci do gier, w augmentacji danych, np. do uzupełniania zbiorów danych medycznych o dodatkowe próbki trenujące, w bezpieczeństwie, np. do symulacji ataków cybernetycznych [37].

Model GANs składa się z dwóch sieci neuronowych, generatora i dyskryminatora. Generator tworzy nowe dane podobne do danych treningowych, przekształcając losowy szum w próbki danych. Dyskryminator ocenia te próbki jako prawdziwe (pochodzące z danych treningowych) lub fałszywe (wygenerowane przez generator) i przekazuje generatorowi informacje zwrotne. Obie sieci są trenowane w grze o sumie zerowej (zysk jednej strony jest równoważony przez straty drugiej), aż generator produkuje dane nieodróżnialne od prawdziwych.

Rekurencyjne sieci neuronowe (ang. *Recurrent Neural Networks*, RNN) zostały zaprojektowane do rozpoznawania wzorców w sekwencjach danych i są powszechnie używane do tłumaczenia języka, przetwarzania języka naturalnego, rozpoznawania mowy i opisywania obrazów. Zostały one wbudowane w popularne aplikacje, takie jak na przykład asystent głosowy Apple'a, wyszukiwanie głosowe i tłumacz Google'a [38]. Cechą RNN jest to, że wykorzystują one informacje z wcześniejszych wejść, aby wpływać na bieżące wejście i wyjście, co pozwala, na przykład, na zachowanie właściwej kolejności słów w tekście. Jednym z pierwszych zastosowań rekurencyjnych połączeń w sieciach neuronowych była sieć zaproponowana przez Johna Hopfielda w 1982 roku [39]. Rekurencyjne sieci neuronowe miały początkowo problem z zapamiętywaniem odległych od siebie informacji w długich sekwencjach danych. W roku 1997 problem ten rozwiązany został przez opracowanie specjalnej architektury – LSTM (ang. *Long Short-Term Memory*), która stosuje komórki pamięci oraz bramki zapominania [40].

W roku 2017 Ashish Vaswani wraz ze współpracownikami zaproponowali sieć neuronową typu transformator (ang. *transformer*), która zrewolucjonizowała podejście do przetwarzania języka naturalnego i innych zadań związanych z sekwencjami danych [41]. Model transformatorowy działa na zasadzie mechanizmu zwanego self-attention, który pozwala ocenić znaczenie każdego elementu w sekwencji w kontekście wszystkich innych elementów. Transformatory składają się z warstw enkodera, które przetwarzają wejściowe sekwencje, i warstw dekodera, które generują wyjściowe sekwencje. Ponieważ sam mechanizm self-attention nie bierze pod uwagę kolejności danych, to w celu dodania informacji o kolejności elementów w sekwencji używane jest kodowanie pozycji. Całość umożliwia równoległe przetwarzanie danych, co znacząco zwiększa efektywność modelu w porównaniu do tradycyjnych sieci rekurencyjnych. Transformatorowe sieci neuronowe są używane do tłumaczenia tekstów (lepiej niż inne metody zachowują kontekst), potrafią generować spójne i

poprawne kontekstowo teksty, mogą być używane jako asystenci wirtualni, do automatycznego tworzenia treści marketingowych, do badania sentymentu, do segmentacji obrazów dla zastosowań medycznych itd. W ostatnim czasie niezwykle popularnym narzędziem opartym na modelu transformatorowym stał się ChatGPT.

4. Asystent

W listopadzie 2022 roku został udostępniony publicznie model językowy ChatGPT, który na pytanie „Co powiesz o sobie?” odpowiada:

„Jestem asystentem AI opracowanym przez OpenAI. Zostałem stworzony, aby pomagać użytkownikom w różnych zadaniach, takich jak odpowiadanie na pytania, tworzenie treści, tłumaczenie tekstów, dostarczanie informacji, i wiele innych. Moje działanie opiera się na zaawansowanych modelach językowych, w szczególności na modelu Generative Pre-trained Transformer (GPT)”.

Model GPT, jak już wspomniano w poprzednim rozdziale, wykorzystuje architekturę transformatora. Najpierw jest trenowany na ogromnych zbiorach tekstu z Internetu, podczas czego uczy się przewidywać następane słowo w zdaniu, bazując na kontekście dostarczonego przez wcześniejsze słowa. Proces ten nazywa się pre-treningiem. Następnie, w procesie zwanym fine-tuningiem, model jest dostosowywany do specyficznych zadań lub dziedzin poprzez trenowanie na zbiorze danych ukierunkowanym na konkretne zastosowanie. W tym procesie stosowane są techniki optymalizacyjne w celu dostrojenia parametrów modelu do konkretnych zadań. Po zakończeniu fine-tuningu model jest testowany na zbiorze walidacyjnym i porównywany z innymi modelami. Fine-tuning jest procesem kosztownym i czasochłonnym, dlatego też ChatGPT miał przeprowadzony fine-tuning tylko w niektórych dziedzinach. Obejmują one między innymi tworzenie treści (w ogólnych konwersacjach, generowaniu tekstów takich jak artykuły, eseje, posty na blogach, opowiadania) i zadania związane z kodowaniem (pomoc w pisaniu kodu, debugowaniu, tłumaczeniu kodu z jednego języka na inny i udzielaniu odpowiedzi na pytania związane z programowaniem). ChatGPT dostarcza również narzędzia oraz API (ang. *Application Programming Interface*), które pozwalają użytkownikom na dostosowanie modelu do ich specyficznych potrzeb.

Obecnie dostępna jest darmowa wersja ChatGPT oparta na modelu GPT-3.5. Dostępna jest również bardziej zaawansowana wersja GPT-4o, ale bezpłatny dostęp do

niej jest limitowany. Wersja GPT-4o była używana w konwersacjach prezentowanych w dalszej części artykułu.

Od chwili publicznego udostępnienia, ChatGPT powszechnie zdominował rozmowy w społeczności edukacyjnej. Jak napisali Brett Becker i Brian Mac Namee w artykule pt. „Generative AI in Computing Education: Wrecking Ball or Holy Grail?” [42]:

„[...] być może nie jest to zaskakujące, gdy weźmiemy pod uwagę, że komputery osobiste potrzebowały dekad, aby trafić do domów i zrewolucjonizować nasze życie, a Internet również rozwijał się stosunkowo wolno, potrzebując lat, aby naprawdę wpłynąć na społeczeństwo. Nawet wtedy komputery osobiste i Internet głównie wspierały nas w robieniu tych samych rzeczy, które zawsze robiliśmy – komunikacji, przekazywaniu pieniędzy, wykonywaniu pracy, znajdowaniu rozrywki itp. W znacznie krótszym czasie GenAI „zjadła Internet na śniadanie” i osiągnęła coś, czego komputery osobiste i Internet nie mogły naprawdę zrobić – generowanie treści. Oczywiście, można by argumentować, że komputery osobiste i Internet generowały treści w pewnych okolicznościach, ale w dużej mierze pomagały ludziom tworzyć i udostępniać treści. Aż do niedawna większość informacji w Internecie była tworzona przez ludzi lub tworzona przez ludzi z pomocą komputerów i programów (które ludzie musieli napisać). GenAI natomiast może generować treści – niektóre z nich całkiem nowatorskie – przy stosunkowo niewielkim wkładzie człowieka, który zazwyczaj ogranicza się do języka naturalnego”².

5. „Mam zaawansowaną wiedzę na temat programowania”

Powyższe zdanie jest fragmentem odpowiedzi, jakiej ChatGPT udzielił na pytanie „Czy jesteś dobry w programowaniu?” W niniejszym rozdziale przyjrzymy się rozwiązaniom uzyskanym przez ChatGPT-4o dla dwóch zadań programistycznych, typowych dla pierwszego roku studiów.

Pierwsze zadanie – napisanie programu wykonującego potęgowanie – jest krótką wprawką dla studentów pierwszego semestru, mającą na celu utrwalenie ich umiejętności w stosowaniu instrukcji pętli oraz instrukcji warunkowych.

Drugie zadanie – napisanie programu obiektowego na podstawie podanego opisu problemu – ma na celu rozwinięcie u studentów drugiego semestru umiejętności w zakresie stosowania technik obiektowych, takich jak dziedziczenie, polimorfizm i abstrakcja.

² Tłumaczenie z języka angielskiego ChatGPT-4o.

Tabele 1 i 2 zawierają zapis interakcji z ChatGPT przy realizacji, odpowiednio, zadań 1 i 2. W pierwszej kolumnie podano numery kolejnych promptów (w polskim języku technicznym przyjęło się używać angielskiego słowa *prompt* na określenie polecenia lub pytania w kontekście modelu językowego), a w drugiej kolumnie znajdują się treści zapytań i fragmenty odpowiedzi generowanych przez chatbot. Na każde pytanie ChatGPT odpowiadał w sposób rozbudowany, podając, oprócz kodu wynikowego, również jego obszerne wyjaśnienie (na ogół nie przytaczane w tabeli). Fragmenty kodu, które powtarzają się w kolejnych wersjach programu i nie są istotne z punktu widzenia danej odpowiedzi, zaznaczone są trzykropkiem w nawiasach kwadratowych: [...].

Tabela 1. Zapis interakcji z ChatGPT przy realizacji zadania 1

Nr	Prompt i fragment wygenerowanej odpowiedzi
1.	<p>Ja: Napisz program w języku C++ wykonujący potęgowanie za pomocą instrukcji pętli.</p> <p>ChatGPT: Oczywiście! Oto przykładowy program w języku C++ wykonujący potęgowanie za pomocą instrukcji pętli:</p> <pre> #include <iostream> using namespace std; // Funkcja do potęgowania liczby za pomocą pętli int potegowanie(int podstawa, int wykladnik) { int wynik = 1; for (int i = 0; i < wykladnik; ++i) { wynik *= podstawa; } return wynik; } int main() { int podstawa, wykladnik; cout << "Podaj podstawę: "; cin >> podstawa; cout << "Podaj wykładnik: "; cin >> wykladnik; int wynik = potegowanie(podstawa, wykladnik); cout << "Wynik: " << wynik << endl; return 0; } </pre>

2.	<p>Ja: Jaka będzie wartość zmiennej wynik w drugim obrocie pętli, gdy na wejściu użytkownik poda liczby 3 i 4?</p> <p>ChatGPT: Jeśli użytkownik poda liczby 3 (podstawa) i 4 (wykładnik), możemy śledzić wartość zmiennej `wynik` w każdym obrocie pętli w funkcji `potegowanie`.</p> <p>Analiza krok po kroku:</p> <ol style="list-style-type: none"> Początkowa wartość `wynik`: <pre>int wynik = 1;</pre> Pierwszy obrót pętli (i = 0): <pre>wynik *= podstawa; // wynik = 1 * 3 = 3</pre> Drugi obrót pętli (i = 1): <pre>wynik *= podstawa; // wynik = 3 * 3 = 9</pre> <p>Zatem w drugim obrocie pętli wartość zmiennej `wynik` wynosi 9.</p>
3.	<p>Ja: Jaki będzie wynik działania programu, jeżeli użytkownik poda liczby 3 i -2?</p> <p>ChatGPT: Jeśli użytkownik poda liczby 3 (podstawa) i -2 (wykładnik), program w obecnej formie nie będzie działał poprawnie, ponieważ nie obsługuje przypadków, gdy wykładnik jest ujemny.</p> <p>Dla wykładników ujemnych musielibyśmy wykonać operację odwrotną, czyli wyliczyć wartość $1/3^2$. Aby to zrobić, program musiałby zostać zmodyfikowany. Poniżej znajduje się zmodyfikowana wersja programu, która obsługuje również ujemne wykładniki:</p> <pre>// Funkcja do potęgowania liczby za pomocą pętli double potegowanie(int podstawa, int wykladnik) { double wynik = 1.0; int abs_wykladnik = wykladnik < 0 ? -wykladnik : wykladnik; for (int i = 0; i < abs_wykladnik; ++i) { wynik *= podstawa; } if (wykladnik < 0) { wynik = 1.0 / wynik; } return wynik; } [...]</pre>
4.	<p>Ja: Jaki będzie wynik działania programu dla liczb 0 i -1?</p> <p>ChatGPT: Dla liczb 0 (podstawa) i -1 (wykładnik), zmodyfikowany program, który obsługuje również ujemne wykładniki, napotka problem. Potęgowanie liczby 0 przez ujemny wykładnik jest nieokreślone matematycznie, ponieważ oznacza to obliczenie $1/0$, co prowadzi do nieskończoności lub jest matematycznie nieokreślone (division by zero).</p> <p>Poniżej znajduje się zmodyfikowany program, który obsługuje ten przypadek, zwracając odpowiedni komunikat o błędzie, gdy podstawa wynosi 0, a wykładnik jest ujemny:</p> <pre>// Funkcja do potęgowania liczby za pomocą pętli double potegowanie(int podstawa, int wykladnik) { if (podstawa == 0 && wykladnik < 0) { cout << "Błąd: nie można podnieść 0 do ujemnej potęgi." << endl; } }</pre>

	<pre> return numeric_limits<double>::quiet_NaN(); // Zwraca NaN (Not a Number) } [...] } int main() { [...] double wynik = potegowanie(podstawa, wykladnik); if (!isnan(wynik)) { cout << "Wynik: " << wynik << endl; } } </pre>
5.	<p>Ja: A może bardziej zwyczajnie?</p> <p>ChatGPT: Oczywiście, oto bardziej prosty i zwyczajny program w C++, który wykonuje potęgowanie za pomocą pętli i obsługuje przypadki, gdy użytkownik poda podstawę 0 i ujemny wykładnik, zwracając w takim przypadku komunikat o błędzie:</p> <pre> // Funkcja do potęgowania liczby za pomocą pętli double potegowanie(int podstawa, int wykladnik) { // Sprawdzenie przypadku 0^-1 if (podstawa == 0 && wykladnik < 0) { cout << "Błąd: nie można podnieść 0 do ujemnej potęgi." << endl; return 0; // Zwraca 0 lub dowolną wartość, która informuje o błędzie } [...] } int main() { [...] double wynik = potegowanie(podstawa, wykladnik); if (!(podstawa == 0 && wykladnik < 0)) { cout << "Wynik: " << wynik << endl; } } </pre>
6.	<p>Ja: Ale ten kod nie jest ładny. Dwa razy występuje sprawdzenie warunku, czy podstawa jest równa 0 i wykładnik jest ujemny.</p> <p>ChatGPT: Rozumiem, upraszczam kod, aby uniknąć podwójnego sprawdzania tego samego warunku. Poniżej znajduje się uproszczona wersja programu:</p> <pre> // Funkcja do potęgowania liczby za pomocą pętli double potegowanie(int podstawa, int wykladnik) { if (podstawa == 0 && wykladnik < 0) { return -1; // Specjalna wartość oznaczająca błąd } [...] } </pre>

	<pre>int main() { [...] double wynik = potegowanie(podstawa, wykladnik); if (wynik == -1) { cout << "Błąd: nie można podnieść 0 do ujemnej potęgi." << endl; } else { cout << "Wynik: " << wynik << endl; } return 0; }</pre>
<p>7.</p>	<p><i>Kontynuacja konwersacji w nowej sesji ChatGPT na temat kodu z promptu 6:</i></p> <p>Ja: Czy załączony kod wykonujący potęgowanie będzie działał poprawnie?</p> <p>ChatGPT: Kod, który przedstawiłeś, wydaje się działać poprawnie dla większości przypadków potęgowania, w tym dla potęg o wykładnikach ujemnych. Poniżej przedstawiam omówienie kodu oraz kilka uwag, które mogą pomóc w poprawieniu jego działania:</p> <p>1. Warunek sprawdzający <code>podstawa == 0 && wykladnik < 0</code>: Sprawdza on przypadek, gdy podstawa jest równa 0, a wykładnik jest ujemny. W takim przypadku, wynik nie jest zdefiniowany (bo dzielenie przez zero nie jest możliwe). Zwracanie -1 jako specjalnej wartości jest w porządku, ale warto by było użyć bardziej jednoznacznej metody sygnalizowania błędu, np. przez wyrzucenie wyjątku.</p> <p>[...]</p> <p>Oto poprawiona wersja kodu z wyrzucaniem wyjątku w przypadku błędu:</p> <pre>// Funkcja do potęgowania liczby za pomocą pętli double potegowanie(int podstawa, int wykladnik) { if (podstawa == 0 && wykladnik < 0) { throw invalid_argument("Nie można podnieść 0 do ujemnej potęgi."); } [...] } int main() { try { [...] double wynik = potegowanie(podstawa, wykladnik); cout << "Wynik: " << wynik << endl; } catch (const invalid_argument& e) { cout << "Błąd: " << e.what() << endl; } }</pre>
<p>8.</p>	<p>Ja: Czy da się ulepszyć utworzony program wykonujący potęgowanie?</p>

	<p>ChatGPT: Oczywiście, można ulepszyć ten program poprzez zastosowanie kilku technik optymalizacyjnych oraz poprawienie czytelności kodu. Możemy wykorzystać algorytm szybkiego potęgowania (eksponentacja przez podnoszenie do kwadratu) do zwiększenia efektywności.</p> <p><i>Po czym ChatGPT wygenerował program implementujący algorytm szybkiego potęgowania.</i></p>
9.	<p>Ja: Nie mogę znaleźć w Internecie terminu eksponentacja.</p> <p>ChatGPT: Rzeczywiście, termin "eksponentacja" może nie być powszechnie używany w literaturze matematycznej. Zazwyczaj stosuje się termin "potęgowanie" (ang. "exponentiation"). Możliwe, że w języku polskim termin "eksponentacja" nie jest powszechnie używany i bardziej odpowiednim określeniem będzie "potęgowanie".</p> <p><i>Po czym następuje definicja potęgowania wraz z przykładami.</i></p>

W przykładzie przedstawionym w tabeli 1, w odpowiedzi na polecenie napisania programu w języku C++ wykonującego potęgowanie za pomocą instrukcji pętli (prompt 1), ChatGPT wygenerował podstawową wersję programu, działającą poprawnie wtedy, gdy dane wejściowe są liczbami całkowitymi dodatnimi oraz podał rzeczowe omówienie kodu (nie zamieszczone w tabeli). Na życzenie użytkownika (prompt 2) prześledził działanie instrukcji pętli, podając wyniki uzyskiwane w kolejnych jej obrotach. Następnie, w odpowiedzi na pytania 3 i 4 o wynik działania programu dla danych z ujemnym wykładnikiem oraz w sytuacji, gdy podstawa jest równa 0, a wykładnik jest ujemny, chatbot podał rozszerzone wersje programu, działające poprawnie w tych przypadkach. Ten prosty przykład pokazuje, jak użyteczny z punktu widzenia studenta może być ChatGPT – student otrzymuje działający i opisany kod, a co więcej, może dopytywać o jego szczegóły. Widać, że większe korzyści z konwersacji odniesie użytkownik do niej przygotowany, mający już pewną wiedzę i umiejętności w programowaniu. Dla podanego przykładu powinien on być świadomy, że otrzymany program należy zawsze przetestować. Oczywiście, można było poprosić ChatGPT o program działający poprawnie dla różnych typowych wartości danych wejściowych albo jeszcze ściślej – o podanie wszystkich przypadków testowych dla zadania i napisanie programu działającego dla wskazanych przypadków.

W odpowiedzi na zapytanie 4 pojawił się problem, jaki często może napotkać początkujący programista korzystający z ChatGPT, mianowicie użycie zaawansowanych elementów i technik w wygenerowanym kodzie. W tym przypadku jest to klasa szablonowa `numeric_limits`. Zrozumienie klas szablonowych wymaga znajomości koncepcji programowania obiektowego. Poszerzanie wiedzy jest oczywiście korzystne, jednak nadmiar nowego materiału może sprawić, że programowanie wyda się studentowi trudne do opanowania, co zniechęci go do nauki w tym kierunku.

Zdarza się też, że początkujący programiści stosują w swoich programach zaawansowane techniki, mając o nich jedynie ogólne pojęcie, co może prowadzić do ich niewłaściwego użycia. W bardziej złożonych projektach może to spowodować, że powstały kod będzie nieefektywny i trudny w utrzymaniu, nawet przy wsparciu zaawansowanych narzędzi AI.

W omawianym przykładzie zakładamy, że użytkownik zorientował się, iż program wygenerowany w odpowiedzi na zapytanie 4 zawiera konstrukcje trudne do zrozumienia na obecnym etapie nauki, i po prostu poprosił o „bardziej zwyczajny” kod (prompt 5). ChatGPT „zrozumiał” intencje pytającego i napisał program przyjazny dla początkującego programisty.

Przyjrzyjmy się jeszcze komentarzowi do instrukcji `return 0` w funkcji potęgującej (prompt 5): `//Zwraca 0 lub dowolną wartość, która informuje o błędzie. Komentarz ten jest mylący – sugeruje, że wartość zwrócona przez funkcję może służyć jako informacja o wystąpieniu błędu, co w tym przypadku nie jest możliwe, ponieważ każda wartość całkowita może być prawidłowym wynikiem potęgowania, a funkcja z definicji właśnie ten wynik zwraca.`

Prompt 6 jest pewnego rodzaju zabawą. Wiedząc, że należy unikać powtórzeń tego samego kodu, zwrócono uwagę na wystąpienie wyrażenia warunkowego: `podstawa == 0 && wykladnik < 0` w dwóch miejscach programu (w funkcji potęgowanie i w funkcji main). Oczywiście ChatGPT również „wie”, że powtórzeń należy unikać (dobre praktyki programowania są przecież szeroko omawiane w Internecie), więc zaproponował rozwiązanie, w którym funkcja potęgowanie sygnalizuje błąd przez zwrócenie wartości `-1`, a funkcja main, na podstawie tej wartości, wypisuje komunikat końcowy. W rezultacie, wprowadzenie jako danych liczb `-1` i `3`, zamiast wyświetlenia wyniku równego `-1`, powoduje pojawienie się komunikatu: `Błąd: nie można podnieść 0 do ujemnej potęgi.` Tutaj ChatGPT „dał się zmanipulować” (jego tendencja do tego była widoczna już w komentarzu do kodu wspomnianym w poprzednim akapicie). Co więcej „wydaje mu się”, że kod będzie działał poprawnie. Proponuje jednak również bezpieczniejsze rozwiązanie, które wykorzystuje mechanizm wyjątków, czyli nieco bardziej zaawansowany element programowania (prompt 7).

Zwracanie istotnej informacji przez funkcję w postaci wartości `0`, `1` lub `-1` jest techniką często stosowaną w programowaniu. Nic więc dziwnego, że pojawiła się ona w zaproponowanym kodzie (choć tym razem została zastosowana błędnie). Pamiętajmy, że pracujemy z modelem językowym, który nie myśli, ale który został

wytrenowany na tekstach z Internetu, w celu przewidywania prawdopodobieństwa wystąpienia określonych słów lub sekwencji słów na podstawie kontekstu, w którym się znajdują.

Na koniec zadano pytanie, czy można ulepszyć program (prompt 8). W odpowiedzi ChatGPT zaproponował bardziej efektywny program implementujący algorytm szybkiego potęgowania. W opisie użył nieistniejącego w języku polskim słowa „eksponencja”, ale po zwróceniu na to uwagi poprawił się i wyjaśnił pomyłkę (prompt 9).

Tabela 2. Zapis interakcji z ChatGPT przy realizacji zadania 2

Nr	Prompt i fragment wygenerowanej odpowiedzi
1.	<p>Ja: Na podstawie poniższego opisu napisz program w języku Python wspierający pracę zespołu ornitologicznego. W programie użyj polskich nazw.</p> <p>Pani Bernikla Szpakowata jest ornitologiem i zajmuje się obserwacją ptaków na żywo z bliskiej odległości, co wymaga od niej różnych umiejętności: wspinania się na drzewa po kruchych gałęziach, wspinania się po skałach bez strącania kamieni, kamuflowania się w piasku, wytrzymałości na wilgoć i zimno podczas ukrywania się w trzcinach. Do tej pory pani Bernikla opanowała umiejętność wspinania się po drzewach i kamuflowania się w piasku, ale szkoli się również w innych umiejętnościach. Pani Bernikla należy do zespołu ornitologicznego, którym kieruje pan Puszczkowski. Zespół składa się z kilkunastu osób. Co pewien czas (parę dni lub tygodni) pan Puszczkowski zleca swoim współpracownikom obserwację ptaków określonego typu. Oczywiście, cele obserwacji są dokładnie sprecyzowane, na przykład, rodzaj wybranego materiału na budowę gniazda i sposób jego układania. Notatki z obserwacji są na bieżąco, każdego dnia, wysyłane kierownikowi, który przegląda je „do poduszki”. Oczywiście, sporządzane są też raporty zbiorcze z obserwacji, zawierające wnioski. Raporty te są przechowywane, a ich wyniki opracowywane i wykorzystywane do planowania dalszych badań. Zespół pana Puszczkowskiego wraz z zespołami zajmującymi się innymi gatunkami zwierząt co roku organizuje warsztaty, na których przedstawiane są referaty dotyczące najważniejszych wyników badań.</p> <p>ChatGPT: Oto przykładowy program w języku Python, który wspiera pracę zespołu ornitologicznego kierowanego przez pana Puszczkowskiego, a także uwzględnia umiejętności pani Bernikli Szpakowatej.</p> <p><i>Po czym następuje kod programu i jego wyjaśnienie.</i></p>
2.	<p>Ja: Narysuj diagram klas (UML), na którym pokazane zostaną wszystkie klasy programu:</p> <p>ChatGPT: Oto diagram klas UML dla podanego programu:</p>

<pre> +-----+ Umiejtnosc +-----+ - nazwa: str - opanowana: bool +-----+ + __init__(nazwa: str, opanowana: bool) + __str__(): str +-----+ </pre>	<pre> +-----+ Ornitolog +-----+ - imie: str - nazwisko: str - umiejetnosci: list +-----+ + __init__(imie: str, nazwisko: str, umiejetnosci: list) + dodaj_umiejtnosc(umiejtnosc: Umiejtnosc): void + __str__(): str +-----+ </pre>
<pre> +-----+ Obserwacja +-----+ - typ: str - cel: str - data: str - obserwator: Ornitolog - notatki: list +-----+ + __init__(typ: str, cel: str, data: str, + dodaj_notatke(notatka: str): void + __str__(): str +-----+ </pre>	<pre> +-----+ ZespolOrnitologiczny +-----+ - kierownik: Ornitolog - czlonkowie: list - obserwacje: list - raporty: list +-----+ + __init__(kierownik: Ornitolog) + dodaj_czlonka(ornitolog: Ornitolog): void + zlec_obserwacje(typ: str, cel: str, data: str, obserwator + przeglad_obserwacji(): void + sporzadz_raport(obserwacja: Obserwacja): str + przeglad_raportow(): void +-----+ </pre>

Po czym następuje wyjaśnienie diagramu.

<p>3.</p>	<p>Ja: Jakie relacje występują pomiędzy poszczególnymi klasami programu? ChatGPT: W programie występują następujące relacje pomiędzy poszczególnymi klasami:</p> <ol style="list-style-type: none"> Klasa Ornitolog a Umiejtnosc <ul style="list-style-type: none"> Relacja: Agregacja Opis: Klasa Ornitolog zawiera wiele instancji klasy Umiejtnosc. Umiejtności są dodawane do listy umiejetnosci w obiekcie Ornitolog. Klasa Obserwacja a Ornitolog <ul style="list-style-type: none"> Relacja: Asocjacja Opis: Każda instancja klasy Obserwacja jest powiązana z jedną instancją klasy Ornitolog, która przeprowadza daną obserwację. Ornitolog jest przechowywany w atrybucie obserwator. Klasa ZespolOrnitologiczny a Ornitolog <ul style="list-style-type: none"> Relacja: Asocjacja Opis: Klasa ZespolOrnitologiczny ma jednego kierownika, który jest instancją klasy Ornitolog. Zespół może również zawierać wiele innych instancji klasy Ornitolog, przechowywanych w liście czlonkowie. Klasa ZespolOrnitologiczny a Obserwacja <ul style="list-style-type: none"> Relacja: Asocjacja Opis: Klasa ZespolOrnitologiczny może zawierać wiele instancji klasy Obserwacja, które są przechowywane w liście obserwacje.
<p>4.</p>	<p>Ja: Czy w punktach 3 i 4 nie ma agregacji lub kompozycji? ChatGPT: Rzeczywiście, w punktach 3 i 4, relacje między klasami mogą być lepiej określone jako agregacje lub kompozycje. <i>Po czym następuje wypisanie relacji i ich wyjaśnienie.</i></p>
<p>5.</p>	<p>Ja: Zmodyfikuj wygenerowany program, tak aby zastosowany został mechanizm polimorfizmu i klasy abstrakcyjne. ChatGPT: Aby zastosować mechanizm polimorfizmu i klasy abstrakcyjne</p>

	<p>w programie wspierającym pracę zespołu ornitologicznego, możemy wprowadzić klasę abstrakcyjną <i>Ornitolog</i> i stworzyć konkretne podklasy dla różnych specjalizacji ornitologów. <i>Zaproponowane podklasy to OrnitologSpecjalistyczny i OrnitologAmator.</i></p>
<p>6.</p>	<p>Ja: A może hierarchia dziedziczenia mogłaby dotyczyć czegoś innego? ChatGPT: Oczywiście, możemy zastosować hierarchię dziedziczenia i polimorfizm do innego aspektu programu. Skupmy się na różnych typach obserwacji ptaków. Utworzymy klasę abstrakcyjną <i>Obserwacja</i> i konkretne podklasy dla różnych rodzajów obserwacji, takich jak <i>ObserwacjaBudowyGniazda</i> i <i>ObserwacjaZachowania</i>. <i>Pytanie zostało powtórzone jeszcze dwa razy. W odpowiedziach ChatGPT zaproponował następujące hierarchie dziedziczenia:</i></p> <ul style="list-style-type: none"> – klasa abstrakcyjna <i>Zadanie</i> i jej podklasy <i>ZadanieObserwacyjne</i> i <i>ZadanieBadawcze</i>, – klasa abstrakcyjna <i>Umiejetnosc</i> i jej podklasy <i>UmiejetnoscWspinaczki</i>, <i>UmiejetnoscKamuflazu</i>, <i>UmiejetnoscWytrzymalosci</i>. <p><i>Dla każdej propozycji został podany odpowiedni kod i jego wyjaśnienie.</i></p>
<p>7.</p>	<p>Ja: Korzystając z naszej dotychczasowej dyskusji napisz jak najkrótszy kod pasujący do opisu, który będzie zawierał dziedziczenie, a któraś z metod będzie wywoływana polimorficznie. ChatGPT: Oczywiście, oto uproszczony program, który zawiera dziedziczenie i wywołanie metody w sposób polimorficzny:</p> <pre> from abc import ABC, abstractmethod # Klasa abstrakcyjna Umiejetnosc class Umiejetnosc(ABC): def __init__(self, nazwa, opanowana=False): self.nazwa = nazwa self.opanowana = opanowana @abstractmethod def wykonaj(self): pass # Konkretnie klasy umiejętności class UmiejetnoscWspinaczki(Umiejetnosc): def wykonaj(self): print(f"Wspinanie się: {self.nazwa}, opanowana: {self.opanowana}") class UmiejetnoscKamuflazu(Umiejetnosc): def wykonaj(self): print(f"Kamuflaż: {self.nazwa}, opanowana: {self.opanowana}") # Klasa Ornitolog class Ornitolog: def __init__(self, imie, nazwisko, umiejetnosci): self.imie = imie self.nazwisko = nazwisko self.umiejetnosci = umiejetnosci def pokaz_umiejetnosci(self): for umiejetnosc in self.umiejetnosci: </pre>

<pre>umiejetnosc.wykonaj() # Przykładowe użycie programu if __name__ == "__main__": wspinaczka = UmiejetnoscWspinaczki("Wspinanie się po drzewach", True) kamuflaz = UmiejetnoscKamuflazu("Kamuflowanie się w piasku", False) bernikla = Ornitolog("Bernikla", "Szpakowata", [wspinaczka, kamuflaz]) bernikla.pokaz_umiejetnosci() Po czym następuje wyjaśnienie kodu.</pre>
--

Przejdźmy teraz do analizy rozwiązania zadania 2. Na początku konwersacji przedstawionej w tabeli 2, ChatGPT został poproszony o wygenerowanie kodu w języku Python na podstawie dość rozbudowanego opisu. Mamy w nim zespół ornitologiczny, jego członków i kierownika. Członkowie zespołu posiadają pewne umiejętności, które są niezbędne do przeprowadzania obserwacji ptaków. Obserwacje wykonywane są przez określony czas i mają określone cele badawcze. Z obserwacji sporządzane są notatki, które są codziennie przeglądane przez kierownika zespołu, a także podsumowujące raporty, których wyniki omawiane są podczas corocznych warsztatów organizowanych we współpracy z innymi zespołami.

Zadanie to zostało sformułowane na potrzeby niniejszego artykułu, jednak podobne zadania są często stosowane w nauczaniu, ponieważ umożliwiają wykorzystanie mechanizmów obiektowych, takich jak hermetyzacja, dziedziczenie, polimorfizm i abstrakcja.

Pierwszy prompt został sformułowany dość ogólnie, bez wskazania konkretnych technik programistycznych, które należy zastosować. W odpowiedzi ChatGPT wygenerował program (nie pokazany w tabeli), w którym występują elementy obiektowości, takie jak definicje czterech klas, reprezentujących umiejętność, ornitologa, obserwację i zespół ornitologiczny, oraz przykładowe działania na ich obiektach. Utworzone klasy zostały pokazane na diagramach UML (ang. *Unified Modelling Language*), o co poproszono chatbota w poleceniu 2.

W programie, występują powiązania między klasami, np. klasa Ornitolog zawiera listę z umiejętnośći (czyli odwołania do obiektów klasy Umiejetnosc). Ponieważ jednak ChatGPT nie radzi sobie dobrze (jeszcze) z graficznym przedstawieniem relacji między klasami, o ich omówienie poproszono go w osobnym poleceniu (prompt 3).

W punktach 1 i 2 omówienia chatbot rozróżnił agregację i asocjację, ale nie zrobił tego w następnych punktach. Nie jest to błąd (agregację można traktować jako szczególny przypadek asocjacji), lecz pewna niekonsekwencja w odpowiedzi. Po dopytaniu, czy w punktach 3 i 4 występuje agregacja i kompozycja, ChatGPT przyznał rację pytającemu (prompt 4).

W powyższej konwersacji ponownie potwierdzają się wcześniejsze obserwacje, że posiadanie przynajmniej podstawowej wiedzy przez użytkownika, tym razem w zakresie programowania i projektowania obiektowego, umożliwia mu świadome pogłębianie tej wiedzy podczas rozmowy z chatbotem. Natomiast brak wiedzy wstępnej może prowadzić do zagubienia się pytającego.

Wygenerowany program zawiera powiązane ze sobą klasy, ale nie pokazuje innych technik obiektowych. Dlatego też poproszono o jego modyfikację (prompt 5), aby zastosować mechanizm polimorfizmu i klasy abstrakcyjne. Wydaje się, że w odpowiedzi chatbot „poszedł na łatwiznę” proponując dziedziczenie dotyczące osób (bardzo popularne w Internecie). Zaproponowane typy pochodne `OrnitologSpecjalistyczny` i `OrnitologAmator` nie mają uzasadnienia w treści zadania. Na prośbę o inną hierarchię dziedziczenia (prompt 6), powtórzoną trzy razy, uzyskano propozycje wynikające z dostarczonego opisu. Za każdym razem generowany był coraz dłuższy kod, aż w końcu chatbot „zapomniał”, że miał on zawierać mechanizm polimorfizmu.

Dlatego też w poleceniu 7 poproszono o wygenerowanie, uwzględniając wcześniejszą dyskusję, jak najkrótszego programu, który zawierałby wskazane mechanizmy programowania obiektowego. W odpowiedzi otrzymano program o charakterze ilustracyjnym, ograniczony tylko do elementów niezbędnych do pokazania polimorfizmu – wywołania polimorficzne dotyczą metody `wykonaj`, której różne wersje (implementacje z podklas klasy `Umiejetnosc`) są uruchamiane w metodzie `pokaz_umiejetnosci` klasy `Ornitolog`.

6. Czy ChatGPT jest pomocny w nauce programowania?

W ankiecie przeprowadzonej wśród studentów pierwszego roku na Uniwersytecie Kyushu w Japonii, 100% respondentów odpowiedziało pozytywnie na powyższe pytanie [43].

Jak wspomniano w rozdziale 4, od momentu upublicznienia ChatGPT w 2022 roku, jego wpływ na edukację jest tematem wielu dyskusji i badań. W dalszej części

rozdziału pokażemy, jak ChatGPT jest postrzegany i wykorzystywany przez studentów pierwszego roku do nauki programowania. Spróbujemy również zidentyfikować główne zagrożenia związane z jego nadmiernym stosowaniem oraz zaproponować sposoby ich omijania.

Na podstawie wyników opracowań [43] i [44], można stwierdzić, że studenci dostrzegają przede wszystkim korzyści płynące z użycia ChatGPT, wynikające z jego możliwości:

- szybkiego i skutecznego odpowiadania na pytania – pozwala to zaoszczędzić czas, który inaczej byłby tracony na poszukiwanie rozwiązań w Internecie;
- pomocy w znajdowaniu błędów w kodzie – umożliwia to uniknięcie żmudnej, czasochłonnej i nieproduktywnej pracy;
- dostarczania przykładów kodu i wyjaśnień – umożliwia to otrzymanie wysokiej jakości wzorcowego programu oraz daje możliwość wielokrotnego pytania o niezrozumiałe elementy, aż do pełnego ich zrozumienia.

Część studentów uważa również, że ChatGPT:

- wspiera rozwój umiejętności myślenia;
- przyczynia się do poprawy umiejętności programistycznych;
- dodaje pewności siebie w programowaniu;
- pomaga w dostrzeganiu alternatywnych rozwiązań problemu.

Jeżeli chodzi o wady i ograniczenia stosowania, to niektórzy studenci wskazują, że ChatGPT:

- podaje odpowiedzi, które nie zawsze są poprawne, albo potrzebne;
- może spowodować nadmierne uzależnienie, co negatywnie wpływa na umiejętność samodzielnego myślenia i nauki.

Z zaletami wskazanymi przez studentów trudno się nie zgodzić. Co do wad, to pierwsza powoduje tylko pewne niedogodności w korzystaniu z generowanych odpowiedzi, natomiast druga wydaje się dużo poważniejsza.

Nauczanie na wyższej uczelni ma na celu, oprócz przekazania studentowi wiedzy, również rozwijanie jego umiejętności twórczego i krytycznego myślenia. Wydaje się, że ograniczenie myślenia tylko do formułowania dobrych promptów dla modeli językowych może zamknąć studentów w błędnym kręgu powierzchownego podejścia do napotykanym problemom.

Zapytany o to, jakie efekty powinno mieć nauczanie programowania na wyższej uczelni, ChatGPT odpowiedział:

„Nauczanie programowania na wyższej uczelni powinno rozwijać zarówno umiejętności techniczne, jak i zdolności analitycznego myślenia, aby studenci potrafili samodzielnie tworzyć, testować i utrzymywać oprogramowanie. Ważne jest, aby studenci nauczyli się pisać programy od podstaw, ponieważ to kształtuje głębokie zrozumienie algorytmów, struktur danych i logiki programowania. Choć umiejętność pisania dobrych promptów jest przydatna, samodzielne kodowanie zapewnia solidne fundamenty, niezbędne do rozwiązywania złożonych problemów i adaptacji do nowych technologii”.

Chatbot poruszył tu bardzo ważną kwestię: aby nauczyć się programować, trzeba samemu pisać, uruchamiać, testować i poprawiać kod – czytanie gotowego kodu i jego analizowanie jest korzystne, ale zapewne nie wystarczy w pracy zawodowej. Zwłaszcza, że jak mówi ChatGPT:

„W przyszłości łatwe, typowe zadania programistyczne będą prawdopodobnie wykonywane przez narzędzia AI, co pozwoli na automatyzację rutynowych i powtarzalnych czynności. Praca programistów będzie ewoluować w kierunku trudniejszych, koncepcyjnych zadań, które wymagają kreatywności, krytycznego myślenia i głębokiego zrozumienia problemów”.

W praktyce dydaktycznej można niestety obecnie dość często zaobserwować niewłaściwe podejście do uczenia się programowania. Nieformalnie, słowami uczącego się, można je opisać następująco: „Nie potrzebuję korzystać z wykładów i materiałów od nauczyciela – wstawię zadania do chatbota i będę się uczyć na podstawie wygenerowanego kodu. Jak go dobrze zrozumiem i uruchomię, to nauczę się dużo więcej, niż gdybym sam pisał program – przecież sam nie stworzę tak dobrego kodu jak ChatGPT.” A oto, co na to powiedział ChatGPT:

„Pomijanie wykładów sprawi, że przegapisz kluczowe podstawy, które są niezbędne do zrozumienia, jak i dlaczego kod działa. Tylko samodzielne pisanie kodu pozwala na prawdziwe zrozumienie programowania i rozwijanie umiejętności debugowania. ChatGPT może pomóc, ale to twoje własne błędy i doświadczenia uczą najwięcej”.

Jak w takim razie pokierować pracą studenta na pierwszym roku nauki programowania? Oczywiście, na wstępie rozważań trzeba założyć, że dla typowych zadań typu „napisz, uruchom i przetestuj program”, kod na pewno będzie generowany, a nie pisany samodzielnie „od zera”. Jest to rzeczywistość, która, zdaniem autora, już się nie zmieni i z którą nie należy walczyć. Bo, mówiąc obrazowo, mało kto chciałby codziennie przemierzać parę kilometrów do pracy na własnych nogach, gdy dostępne są szybsze i wygodniejsze środki transportu, mimo że chodzenie jest zdrowe i zalecane przez lekarzy, a jego brak skraca życie. W tej nowej rzeczywistości należy podjąć działania

wspierające studenta wkraczającego w świat programowania, aby mógł on w pełni przygotowany przejść do bardziej zaawansowanych etapów nauki. Zauważmy przy okazji, że na wyższych semestrach problem nadmiernego korzystania z ChatGPT istotnie się zmniejsza. Z doświadczeń autora wynika, że szczegółowe wymagania dla zaawansowanych tematów, nawet dotyczących popularnych dziedzin, takich jak uczenie maszynowe, można formułować w taki sposób, aby ich poprawna realizacja wymagała w znacznym stopniu samodzielnej pracy studenta, wspartej dostarczonymi materiałami dydaktycznymi.

Na początku kursu programowania warto wyjaśnić studentom, jak działają modele językowe (takie jak ChatGPT) oraz omówić korzyści wynikające z ich stosowania, a także zwrócić uwagę na ich naturalne ograniczenia. Należy udzielić wskazówek co do właściwego formułowania zapytań do chatbota oraz zwrócić uwagę na konieczność krytycznej oceny otrzymywanych odpowiedzi. Ważne jest, aby uświadomić studentom, że samo czytanie kodu może dawać pozorne poczucie jego rozumienia i wobec tego ważne jest, aby sprawdzać na bieżąco swoje prawdziwe umiejętności.

Taką bieżącą weryfikację umiejętności przez samych studentów można wspomóc przez przygotowanie krótkich, niepodlegających ocenie, zadań dla chętnych. Powinny one dotyczyć różnych aspektów kodowania i mieć urozmaiconą formę, np. uzupełnianie kodu, wskazywanie błędów, podawanie wartości zmiennych po wykonaniu sekwencji instrukcji itp. Zadania takie mogą być udostępniane na platformie edukacyjnej zintegrowanej z modelami językowymi albo wykorzystującej klasyczne narzędzia do tworzenia pytań testowych różnych typów.

Podlegające ocenie, niewielkie zadania programistyczne dotyczące poszczególnych tematów kursu mogą być rozliczane w formie prezentacji multimedialnych lub raportów, które wymuszają bardziej aktywną pracę z kodem niż tylko jego generowanie. Na przykład można poprosić studenta o nagranie procesu śledzenia działania kodu linijka po linijce lub o umieszczenie w raporcie zrzutów ekranu zawierających wyświetlone pośrednie wartości zmiennych uzyskiwane w trakcie wykonywania wskazanych instrukcji programu. Zadania tego typu powinny być wykonywane z użyciem wskazanego środowiska programistycznego.

Od samego początku nauki programowania należy przyzwyczajać studentów do pracy w zaawansowanych środowiskach programistycznych wykorzystujących sztuczną inteligencję, takich jak Visual Studio Code, IntelliJ IDEA, PyCharm.

Stosowanie profesjonalnych środowisk, umożliwiających ciekawą i szybką pracę z kodem, może zwiększyć motywację i zaangażowanie studentów. Z drugiej strony, ułatwiają one nauczycielowi monitorowanie postępów w nauce.

W nauczaniu programowania szczególnie ważna staje się teraz bezpośrednia rozmowa ze studentem na temat dostarczonego przez niego kodu. Zdaniem autora, taką formę zaliczenia należy bezwzględnie stosować w przypadku większych projektów, które integrują wiedzę zdobytą w trakcie semestru i mają największy wpływ na ocenę końcową z przedmiotu. Wskazane jest również przeprowadzanie rozmów na temat mniejszych zadań realizowanych „z lekcji na lekcję”, jednak jest to często trudne ze względu na dużą liczbę studentów i ograniczenia czasowe.

W przypadku większych projektów można poprosić studenta o sukcesywne budowanie programu, rozpoczynając od podstawowych funkcji i stopniowo rozbudowując go o bardziej zaawansowane elementy. Dla każdej wersji należy zapisać zapytania do ChatGPT, które do niej doprowadziły, udokumentować jej uruchomienie oraz samodzielne przetestowanie dla różnych wartości danych wejściowych. Warto również poprosić o graficzne przedstawienie programu, na przykład poprzez narysowanie schematów blokowych dla zaimplementowanych algorytmów oraz diagramów klas w programach zorientowanych obiektowo.

Podczas rozmowy ze studentem nie należy ograniczać się jedynie do omówienia programu (znaczenia zmiennych, użytych struktur danych, zastosowanych technik), ale warto również wprowadzić elementy aktywnej pracy z kodem, na przykład można poprosić studenta o dokonanie pewnych modyfikacji „na żywo” i sprawdzenie ich działania.

Zainteresowanym studentom na drugim semestrze nauki programowania można zaproponować wykonanie większych projektów w zespołach dwu- lub trzyosobowych. Współpraca może odbywać się na platformie GitHub, która umożliwia śledzenie postępu projektu i wkładu członków zespołu w czasie rzeczywistym, a także uczy zarządzania kodem, dokumentowania pracy i praktykowania dobrych nawyków programistycznych.

Aby efektywnie realizować proces nauczania programowania, dostosowując go do specyfiki kursu, warto rozważyć utworzenie interaktywnych chatbotów z wykorzystaniem API ChatGPT. Takie chatboty, oprócz udzielania odpowiedzi na pytania studentów, mogą również generować spersonalizowane zadania i quizy, które

stopniowo zwiększając poziom trudności w miarę postępów studentów, co sprzyja efektywnemu przyswajaniu wiedzy.

7. Podsumowanie

W ostatnich latach obserwujemy dynamiczny rozwój narzędzi wykorzystujących sztuczną inteligencję, które wspierają studentów w zdobywaniu wiedzy i umiejętności praktycznych, ale jednocześnie mogą ograniczać ich samodzielne myślenie i kreatywność. W związku z tym konieczne jest opracowanie nowych metod nauczania oraz, ewentualnie, zmodyfikowanie celów dydaktycznych, aby dostosować je do nowej rzeczywistości. Obecnie brakuje skutecznych, nadrzędnych wytycznych w tym zakresie, niewiele jest też dostępnych wyników badań w literaturze. W konsekwencji wykładowcy muszą przede wszystkim polegać na własnej intuicji i doświadczeniu.

W niniejszym artykule, po krótkim wprowadzeniu historycznym dotyczącym rozwoju komputerów i sztucznej inteligencji, omówiono wpływ publicznego udostępnienia modelu językowego sztucznej inteligencji, jakim jest ChatGPT, na nauczanie programowania. Przedstawiono korzyści płynące z używania tego narzędzia, a także wskazano na zagrożenia związane z jego nadmiernym lub niewłaściwym wykorzystaniem. Zaproponowano konkretne rozwiązania dotyczące formułowania zadań programistycznych oraz sposobów ich oceny, tak aby studenci mogli bardziej świadomie i aktywnie rozwijać swoje umiejętności w programowaniu, a nie ograniczać się tylko do biernego rozumienia kodu wygenerowanego przez ChatGPT.

Literatura

- [1] S. Russell, P. Norvig, *Artificial Intelligence. A Modern Approach*, [online] <https://dl.ebooksworld.ir/books/Artificial.Intelligence.A.Modern.Approach.4th.Edition.Peter.Norvig.%20Stuart.Russell.Pearson.9780134610993.EBooksWorld.ir.pdf>.
- [2] *Abacus*, [online] <https://www.britannica.com/technology/abacus-calculating-device>.

- [3] B. Klin, *Abakus – najstarsze urządzenie rachunkowe*, Delta 7, 2017, [online] <https://www.deltami.edu.pl/2017/07/abakus-najstarsze-urządzenie-rachunkowe/>.
- [4] S. Augarten, *BIT by BIT. An Illustrated History of Computers*, [online] <http://ds-wordpress.haverford.edu/bitbybit/bit-by-bit-contents/>.
- [5] *The History of Vacuum Tubes*, [online] <https://www.telecom-milestones.com/vacuum-tubes-history>.
- [6] *Atanasoff Berry Computer*, [online] <https://jva.cs.iastate.edu/operation.php>.
- [7] B. Miś, *Najsłynniejszy komputer w dziejach*, [online] https://historiainformatyki.pl/common/files_download.php?fid=241.
- [8] *Colossus*, [online] <https://www.tnmoc.org/colossus>.
- [9] *UNIVAC I*, [online] <https://www.pcmag.com/encyclopedia/term/univac-i>.
- [10] *IBM 701*, [online] <https://www.pcmag.com/encyclopedia/term/ibm-701>.
- [11] W. Brinkman, D. E. Haggan, W. Troutman, *A history of the invention of the transistor and where it will lead us*, IEEE Journal of Solid-State Circuits SC-32(12), 1998. [online] https://www.researchgate.net/publication/2977642_A_history_of_the_invention_of_the_transistor_and_where_it_will_lead_us.
- [12] *1953: Transistorized Computers Emerge*, [online] <https://www.computerhistory.org/siliconengine/transistorized-computers-emerge/>.
- [13] *Generations of computers*, [online] https://uomustansiriyah.edu.iq/media/lectures/6/6_2023_05_31!11_00_11_PM.pdf.
- [14] *Jack Kilby and the World's First Integrated Circuit*, [online] <https://www.al-aboutcircuits.com/news/jack-kilby-and-the-world-first-integrated-circuit/>.
- [15] *R. Noyce – twórca krzemowej elektroniki*, [online] <https://mlodytechnik.pl/technika/28878-robert-noyce-tworca-krzemowej-elektroniki>.
- [16] T.S. Perry. *How Ted Hoff Invented the First Microprocessor*, [online] <https://spectrum.ieee.org/ted-hoff>.
- [17] K. Shirriff. *The Surprising Story of the First Microprocessors*, [online] <https://spectrum.ieee.org/the-surprising-story-of-the-first-microprocessors>.
- [18] *Pong Game*, [online] <https://www.ponggame.org/>.
- [19] *Computer networking history*, [online] <https://www.computerhope.com/history/network.htm>.

- [20] P. Inger, *Email Through the Ages – From the Beginnings to its Importance Today*, [online] <https://www.mailstore.com/en/blog/history-of-email/>.
- [21] *A Short History of the Internet*, [online] <https://www.scienceandmediamuseum.org.uk/objects-and-stories/short-history-internet>.
- [22] *Faster, smaller and more-powerful PCs*, [online] <https://www.britannica.com/technology/personal-computer/Faster-smaller-and-more-powerful-PCs>.
- [23] *75,8 proc. Polaków korzysta ze smartfona, a z tabletu 12,3 proc.*, [online] <https://www.gov.pl/web/krrit/758-proc-polakow-korzysta-ze-smartfona-z-tabletu-123-proc>.
- [24] M. Morioka, R.O. Muireartaigh, “Descartes and Artificial Intelligence”. *Journal of Philosophy of Life* 13(1), 2023, [online] https://www.academia.edu/92216737/Artificial_Intelligence_Robots_and_Philosophy.
- [25] A. Turing, *Computing Machinery and Intelligence*, *Mind* LIX(236), 1950, [online] <https://web.archive.org/web/20110726153108/http://orium.homelinux.org/paper/turingai.pdf>.
- [26] R. Anyoha, *The history of Artificial intelligence*, [online] <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>
- [27] A. Toosi, A. Bottino, B. Saboury, E. Siegel, A. Rahmim, *A Brief History of AI: How to Prevent Another Winter (A Critical review)*, [online] arXiv:2109.01517v3.
- [28] *ELIZA: a very basic Rogerian psychotherapist chatbot*, [online] <https://web.njit.edu/~ronkowitz/eliza.html>
- [29] M. Lefkowitz, *Professor’s perceptron paved the way for AI – 60 years too soon*, [online] <https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon>.
- [30] P. Werbos, *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard University, Cambridge, 1974.
- [31] D.E. Rumelhart, G.E. Hinton, R.J. Williams, *Learning Representations by Back-Propagating Errors, Parallel distributed processing: explorations in the microstructure of cognition 1*, MIT Press, Cambridge, 1986.
- [32] Y. LeCun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, W. Hubbard, *Handwritten digit recognition: Applications of neural net chips and automatic learning*, IEEE Communication, 1989.

- [33] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE 86/11, 1998, [online] <https://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>.
- [34] A. Krizhevsky, I. Sutskever, G. E Hinton. *Imagenet classification with deep convolutional neural networks*. Adv. Neural Inf. Process. Syst., 25, 2012, [online] https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
- [35] *What are convolutional neural networks?*, [online] <https://www.ibm.com/topics/convolutional-neural-networks>.
- [36] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, *Generative Adversarial Networks*, 2014, arXiv:1406.2661v1.
- [37] S.M. Rayavarapu, S.P. Tammineni, S.R. Gottapu, A. Singam, *A Review of Generative Adversarial Networks for Security Applications*, Informatyka, automatyka, pomiary w gospodarce i ochronie środowiska 2, 2024, <http://doi.org/10.35784/iapgos.5778>.
- [38] *What are recurrent neural networks?*, [online] <https://www.ibm.com/topics/recurrent-neural-networks>.
- [39] J.J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences, 1982. [online] https://www.researchgate.net/publication/16246447_Neural_Networks_and_Physical_Systems_with_Emergent_Collective_Computational_Abilities.
- [40] S. Hochreiter, J. Schmidhuber, *Long short-term memory*, Neural Computation 9, 1997. [online] https://www.researchgate.net/publication/13853244_Long_Short-term_Memory.
- [41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, *Attention is All You Need*, 2017, arXiv:1706.03762v7.
- [42] B.A. Becker, B. Mac Namee, *Generative AI in Computing Education: Wrecking Ball or Holy Grail?*, [online] <https://www.ucd.ie/cs/blog/generativeaiin-computingeducationwreckingballorholylgrail/>.
- [43] B. Ma, C. Li, K. Shin'ichi, *Enhancing Programming Education with ChatGPT: A Case Study on Student Perceptions and Interactions in a Python Course*, 2024, arXiv:2403.15472v2.

- [44] R. Yilmaz, F.G.K. Yilmaz, *Augmented intelligence in programming learning: Examining student views on the use of ChatGPT for programming learning*, *Computers in Human Behavior: Artificial Humans 1*, 2024, [online] <https://www.sciencedirect.com/science/article/pii/S2949882123000051>.
-

Learning Programming with ChatGPT: A Shortcut to Success or a Way Astray?

Abstract

The article examines the benefits and potential risks of using ChatGPT in programming education. It presents a brief history of computers and artificial intelligence, along with examples of programming solutions generated by the chatbot. The article also suggests ways to encourage students to actively engage in programming, rather than just relying on AI-generated code.

Keywords: *artificial intelligence, ChatGPT, programming education*