Raport badawczy: Historia sztucznej inteligencji — od systemów eksperckich do ery modeli generatywnych (GenAI)

1. Wprowadzenie

Zrozumienie fundamentalnych mechanizmów sztucznej inteligencji przestało być opcjonalną ścieżką kariery, a stało się absolutnym wymogiem dla każdego inżyniera oprogramowania wkraczającego na rynek w latach 2025 i 2026. Z perspektywy przyszłego programisty, znajomość historycznej ewolucji tych algorytmów pozwala w pełni pojąć, dlaczego współczesne środowiska programistyczne działają w określony sposób, jakie posiadają ukryte ograniczenia architektoniczne oraz w jaki sposób optymalizować kod współpracujący z potężnymi modelami językowymi.

Sztuczna inteligencja (Artificial Intelligence, AI) w najprostszym, roboczym ujęciu to dziedzina informatyki zajmująca się projektowaniem i tworzeniem systemów obliczeniowych zdolnych do wykonywania zadań, które tradycyjnie wymagałyby ludzkiego procesu poznawczego, takich jak wnioskowanie, rozpoznawanie wzorców, czy rozumienie języka. Zdecydowana większość obecnych rozwiązań komercyjnych klasyfikowana jest jako wąska sztuczna inteligencja (Narrow AI), która specjalizuje się w wybitnym rozwiązywaniu jednego, konkretnie zdefiniowanego problemu. Modele te nie naśladują ludzkiej świadomości, lecz wykorzystują niezwykle zaawansowane mapowanie probabilistyczne, statystykę wielowymiarową oraz sztuczne sieci neuronowe, aby na podstawie analizy ogromnych zbiorów danych wejściowych wygenerować najbardziej prawdopodobną odpowiedź.

Dla młodego adepta sztuki programowania, wkraczającego na rynek pracy po 2025 roku, kontekst ten ma wymiar wybitnie pragmatyczny i zarazem rewolucyjny. Wbrew pesymistycznym prognozom, AI nie eliminuje zawodu programisty, lecz drastycznie zmienia jego naturę i podnosi wartość samej inżynierii. Czyste "pisanie kodu" (ang. coding) oparte na zapamiętywaniu składni i ręcznym tworzeniu powtarzalnych struktur staje się czynnością w pełni zautomatyzowaną. Współczesny inżynier oprogramowania transformuje się w architekta systemów i weryfikatora logiki biznesowej. Zjawisko to doprowadziło do formalnego wyłonienia się nowej, dominującej kategorii zawodowej: programisty wspomaganego przez AI (AI-Assisted Developer). Rola takiego inżyniera polega na definiowaniu problemu, umiejętnym instruowaniu modeli sztucznej inteligencji, a następnie rygorystycznym audytowaniu bezpieczeństwa, wydajności i integracji wygenerowanego przez maszynę kodu z resztą repozytorium.

Zmiany te znajdują potężne odzwierciedlenie w globalnych statystykach wykorzystania narzędzi AI. Rynek adaptuje te rozwiązania w tempie niespotykanym w historii informatyki. Platforma GitHub Copilot, będąca jednym z liderów asystentów kodowania, osiągnęła w lipcu 2025 roku imponującą barierę 20 milionów użytkowników wszech czasów, notując wcześniej przyrost na poziomie 400% rok do roku. W środowiskach korporacyjnych asystenci AI przestali być eksperymentem — na przełomie 2025 i 2026 roku korzystało z nich już około 90% firm z prestiżowej listy Fortune 100.

Wpływ sztucznej inteligencji na codzienną architekturę oprogramowania można łatwo skwantyfikować. Średnio około 46% całego nowo powstającego kodu na platformach

wspieranych przez te narzędzia jest generowane bezpośrednio przez sztuczną inteligencję. W przypadku konkretnych środowisk obiektowych, takich jak Java, wskaźnik ten wzrasta do oszałamiających 61%, a w Pythonie wynosi 55%.
Badania produktywności, w tym potężny raport Stack Overflow z 2025 roku oparty na ankietach dziesiątek tysięcy deweloperów, bezlitośnie obalają mit o powolnej adaptacji. Ponad 52% programistów kategorycznie potwierdza pozytywny wpływ asystentów AI na ich wydajność pracy. Kontrolowane środowiska badawcze dowodzą z kolei, że programiści wykorzystujący modele generatywne kończą zlecane zadania o 55% szybciej, a czas potrzebny na przegląd kodu (pull request) spada o niemal 75% — z 9,6 dnia do średnio 2,4 dnia. Z perspektywy pracodawcy ignorowanie takiego wzrostu produktywności jest biznesowym samobójstwem.

Kategoria zadania programistycznego	Odsetek programistów używających AI (2025)
Wdrażanie i monitorowanie (Deployment / Ops)	75,8%
Planowanie architektury projektów	69,2%
Analityka predykcyjna (Predictive analytics)	65,6%
Przegląd i zatwierdzanie kodu (Commit/Review)	58,7%
Pisanie testów jednostkowych i integracyjnych	44,1%
Generowanie dokumentacji (Documenting code)	38,5%
Debugowanie i naprawa błędów	36,4%
Nauka nowych koncepcji i języków	32,3%

Polski rynek pracy odzwierciedla te globalne megatrendy z równą mocą. Raport przygotowany przez instytut badawczy NASK wskazuje, że na polskim rynku około 817,5 tysiąca stanowisk (co stanowi ponad 5 milionów miejsc pracy ogółem, w tym ogromna część ról technicznych i biurowych) jest wysoce podatnych na bezpośrednią interakcję z systemami generatywnymi. Posiadanie umiejętności twardego programowania (np. znajomość składni C++ czy JavaScript) staje się zaledwie warunkiem koniecznym, ale już niewystarczającym do zdobycia wysoce płatnej posady. Dla ucznia technikum informatycznego oznacza to, że aby zachować przewagę konkurencyjną w 2026 roku, musi on rozumieć algorytmiczne podstawy AI, wiedzieć, jak skutecznie inżynierować polecenia (prompting), a także rozumieć ewolucyjne ograniczenia systemów, by sprawnie łatać luki w automatycznie tworzonym oprogramowaniu.

2. Prehistoria i narodziny AI

Dla inżyniera oprogramowania analiza wczesnej historii sztucznej inteligencji to lekcja pokory dowodząca, że najbardziej wizjonerskie koncepcje algorytmiczne muszą często poczekać dekady na odpowiednią moc sprzętową, aby zadziałać w praktyce. Zrozumienie, dlaczego wczesne algorytmy opierały się na przeszukiwaniu symbolicznym, ukazuje, jak bardzo oprogramowanie jest nierozerwalnie związane z fizycznymi ograniczeniami architektury von Neumanna.
Ambicje stworzenia maszyn potrafiących myśleć można odnaleźć już w mitologii greckiej, jednak w ujęciu czysto informatycznym grunt pod te rozważania położył brytyjski matematyk i kryptolog Alan Turing. Zaledwie dekadę po stworzeniu teoretycznych zrębów informatyki (tzw. Maszyny Turinga), w 1950 roku opublikował on epokowy esej "Computing Machinery and Intelligence". Zaproponował w nim eksperyment myślowy, który ominął zawiłe filozoficzne dysputy o naturze samej "świadomości". Zamiast tego Turing zadał inżynieryjne pytanie: czy maszyna potrafi komunikować się tak, by oszukać człowieka?. Eksperyment, nazwany przez

niego "Grą w Naśladownictwo" (The Imitation Game), a dziś znany powszechnie jako Test Turinga, polega na posadzeniu ludzkiego sędziego przed terminalem tekstowym. Sędzia prowadzi konwersację z dwoma ukrytymi bytami — maszyną oraz człowiekiem. Jeśli na podstawie samej logiki i płynności odpowiedzi sędzia nie potrafi wskazać, który rozmówca jest oprogramowaniem, maszynę uznaje się za inteligentną. Choć Test Turinga wyznaczał horyzont badań przez ponad sześć dekad, współcześnie stracił on na znaczeniu. Nowoczesne duże modele językowe (LLM) z łatwością potrafią udawać człowieka, mimo że nie dysponują świadomością ani autentycznym pojmowaniem świata. Pokazuje to dobitnie, że perfekcyjne naśladownictwo jest algorytmicznie prostsze do osiągnięcia niż wygenerowanie ogólnej sztucznej racjonalności.

Za oficjalne narodziny sztucznej inteligencji jako odrębnej, sformalizowanej gałęzi nauki uznaje się 1956 rok. To właśnie wtedy, podczas letnich warsztatów naukowych w Dartmouth College w Stanach Zjednoczonych, zebrali się najwybitniejsi ówcześni wizjonerzy, w tym John McCarthy, Marvin Minsky, Nathaniel Rochester i Claude Shannon. John McCarthy po raz pierwszy użył tam oficjalnie terminu "sztuczna inteligencja". Optymizm tamtego lata był gigantyczny; powszechnie zakładano, że grupa wybitnych matematyków zaprogramuje pełną inteligencję maszyny w przeciągu zaledwie kilku tygodni lub lat, opierając się wyłącznie na tworzeniu sprytnych, logicznych algorytmów na wczesnych komputerach cyfrowych.

Bezpośrednim dowodem podsycającym ten optymizm były rewelacyjne sukcesy pierwszych zaawansowanych programów komputerowych. W latach 1955–1956 Allen Newell i Herbert Simon skonstruowali program Logic Theorist (Teoretyk Logiczny). W przeciwieństwie do wcześniejszych maszyn liczących, takich jak ENIAC, które zajmowały się wyłącznie rozwiązywaniem operacji arytmetycznych dla artylerii, Logic Theorist miał zadanie czysto symboliczne. Zaprojektowano go do automatycznego dowodzenia skomplikowanych twierdzeń matematycznych ze słynnego zbioru *Principia Mathematica* autorstwa Bertranda Russella i Alfreda Northa Whiteheada. Algorytm opierał się na rygorystycznych zestawach wcześniej zdefiniowanych aksjomatów oraz implementował złożone przeszukiwanie drzewa logiki z użyciem mechanizmów takich jak łańcuchowanie wprzód (forward chaining) i łańcuchowanie wstecz (backward chaining). Sukces oprogramowania był druzgocący. Program udowodnił 38 różnych twierdzeń logicznych z książki, a w jednym przypadku znalazł dowód matematyczny krótszy i znacznie bardziej elegancki, niż wymyślili to sami autorzy dzieła. Był to bezprecedensowy moment, w którym udowodniono, że kod programistyczny potrafi symulować ludzką logikę w zamkniętym, aksjomatycznym systemie.

Kolejny milowy krok wykonano w latach 1964–1967 w laboratoriach MIT, gdzie Joseph Weizenbaum napisał wczesny program przetwarzania języka naturalnego o nazwie ELIZA. Był to pierwszy w historii chatbot, który w odróżnieniu od oprogramowania Newella nie rozwiązywał zadań matematycznych, lecz symulował konwersację. Program napisany w języku MAD-SLIP obsługiwał różne skrypty zachowań, z których najsłynniejszy nazywał się DOCTOR i udawał ludzkiego psychoterapeutę z nurtu rogeriańskiego. W rzeczywistości ELIZA była pod względem programistycznym mistrzowskim oszustwem. Nie posiadała absolutnie żadnego modułu rozumienia słów; jej architektura opierała się w 100% na dopasowywaniu wzorców (pattern matching) i sztywnym parsowaniu. Program wyszukiwał w terminalu słowa kluczowe wypisane przez użytkownika i stosował proste reguły substytucji. Gdy użytkownik wpisywał: "I feel sad" (Czuję się smutny), parser ELIZY łapał wzorzec I feel X i podstawiał go pod zapisaną na sztywno odpowiedź: Why do you feel X? (Dlaczego czujesz się smutny?). Jeśli system nie znalazł żadnego dopasowania słownikowego, zwracał generyczną odpowiedź, na przykład "Opowiedz mi o tym coś więcej". Ku zaskoczeniu samego twórcy, asystentka ta wywołała ogromny szok socjologiczny. Użytkownicy zaczęli zwierzać się maszynie z niezwykle intymnych

szczegółów życia, będąc absolutnie przekonanymi, że oprogramowanie wykazuje wobec nich ogromną dozę ludzkiej empatii. Fenomen ten na stałe wszedł do słownika informatyki jako "Efekt ELIZY".
Złota Era optymizmu szybko jednak zderzyła się z bezlitosnymi prawami fizyki i matematyki dyskretnej. Gdy inżynierowie oprogramowania zaczęli wykraczać poza proste, zamknięte środowiska logiczne i starali się zaprogramować algorytmy do radzenia sobie z problemami świata rzeczywistego, napotkali na tzw. eksplozję kombinatoryczną. W świecie rzeczywistym liczba możliwych stanów decyzyjnych rośnie szybciej niż wykładniczo, podczas gdy sprzęt tamtych lat dysponował śladową wręcz ilością pamięci RAM i żałośnie powolnymi procesorami. W 1973 roku w Wielkiej Brytanii opublikowano druzgocący dla branży raport Lighthilla, który brutalnie podsumował, że techniki analityczne odkryte w sztucznej inteligencji nie nadają się do skalowania poza bezpieczne warunki laboratoryjne, wieszcząc całkowite niespełnienie wcześniejszych obietnic. Rozczarowani sponsorzy rządowi i prywatni odcięli finansowanie laboratoriów, co zapoczątkowało trwający ponad dekadę okres rynkowego zastoju nazywany potocznie Pierwszą Zimą AI (AI Winter).

3. Era systemów eksperckich i regułowych

Dla młodego dewelopera zapoznanie się z koncepcją systemów eksperckich jest kluczowe, gdyż stanowi doskonałe studium przypadku tzw. systemów opartych na twardych regułach (Rule-Based Systems). Pozwala to zrozumieć, gdzie kończy się optymalna użyteczność klasycznego kodowania instrukcji warunkowych if-else, a gdzie w architekturze oprogramowania niezbędne staje się zaimplementowanie samouczących się sieci matematycznych.
W latach 80. środowisko inżynieryjne wyciągnęło bolesną lekcję z zamarznięcia funduszy. Odrzucono mrzonki o próbie zbudowania "maszyny myślącej tak jak człowiek", przekierowując 100% zasobów na wyspecjalizowane narzędzia komercyjne. Narodziła się kategoria systemu eksperckiego (Expert System) — skomplikowanego programu komputerowego, który miał jedno jedyne zadanie: emulować kompetencje decyzyjne ludzkiego eksperta, ale tylko i wyłącznie w bardzo zawężonej dziedzinie, jak inżynieria materiałowa czy podatki.
Tajemnica potęgi i późniejszego upadku systemów eksperckich kryła się w ich architekturze. Każdy tego typu system opierał się na drastycznym rozdzieleniu danych od logiki, dzieląc się na dwa fundamenty:

1. **Baza Wiedzy (Knowledge Base)**: Był to swego rodzaju pasywny, ogromny kontener pamięci. Znajdowały się tam tysiące faktów, praw fizyki, zależności dziedzinowych oraz heurystyk wydobytych wcześniej na drodze wielomiesięcznych wywiadów z ekspertami. Wiedza ta zapisywana była ręcznie przez programistów pod postacią ścisłych logicznych warunków, głównie w formacie instrukcji: "JEŚLI warunek A oraz warunek B są spełnione, TO wykonaj akcję C".
2. **Silnik Wnioskowania (Inference Engine)**: Stanowił moduł procesujący. Gdy użytkownik wprowadzał swoje bieżące dane (np. parametry chemiczne próbki), silnik wnioskowania przeszukiwał ustrukturyzowaną bazę wiedzy, dynamicznie dobierając łańcuchy reguł krok po kroku w celu dojścia do ostatecznej logicznej konkluzji. Zaletą tej separacji było to, że programista mógł aktualizować reguły w bazie wiedzy, nie psując przy tym samej pętli logicznej silnika.

Sztandarowym pionierskim przykładem tej architektury z lat 60. był DENDRAL. Zaprojektowany przez badaczy ze Stanfordu (w tym genetyka Joshuę Lederberga oraz programistę Edwarda Feigenbauma) potężny system analityczny był używany w laboratoriach chemicznych do

analizowania wyników spektrometrii mas. Program potrafił z ogromną dokładnością zdedukować i zaprezentować strukturę cząsteczek organicznych na podstawie wprowadzanych parametrów widma. W tej niezwykle wąskiej niszy oprogramowanie osiągało wyniki wielokrotnie przewyższające początkujących badaczy.
Jednak za system, który trwale zapisał się w edukacji informatycznej, uważa się MYCIN, stworzony we wczesnych latach 70., również na uniwersytecie Stanford. Podczas gdy DENDRAL analizował cząsteczki chemiczne, MYCIN był pierwszym doradcą medycznym dla lekarzy. Jego głównym celem operacyjnym była szybka diagnostyka niebezpiecznych bakteryjnych infekcji krwi (np. zapalenia opon mózgowych) oraz ordynowanie lekarzom konkretnych, dopasowanych do masy pacjenta dawek bardzo rzadkich antybiotyków. Całość jego wiedzy składała się z kilkuset reguł ustrukturyzowanych w systemie eksperckim. Typowy schemat logiczny (syntaktyka IF-THEN) wyglądał w kodzie MYCIN-a następująco: "JEŚLI barwienie bakterii metodą Grama jest ujemne ORAZ morfologia organizmu ma kształt pałeczki ORAZ u pacjenta występuje gorączka, TO istnieje dowód (z określonym czynnikiem pewności, np. 0.8), że klasa zakażenia to organizmy jelitowe".
Technologicznie i statystycznie program odniósł oszałamiający sukces. Podczas ślepych testów przeprowadzonych z użyciem dokumentacji historycznej MYCIN potrafił wytypować poprawne terapie trafniej niż wykwalifikowani specjaliści od chorób zakaźnych, nie wspominając o stażystach. Paradoksalnie jednak, mimo udowodnionej przewagi obliczeniowej, system nigdy nie trafił na szpitalne oddziały. Barierą nie był niedoskonały kod czy błędy w pamięci RAM. Problemem okazały się potężne kwestie prawne i etyczne — kto odpowiada za śmierć pacjenta spowodowaną błędem matematycznego algorytmu? Ówcześni lekarze darzyli komputery zbyt małym zaufaniem, by powierzyć im ludzkie życie w systemie, który odmawiał bycia elastycznym.
Z perspektywy architektury oprogramowania systemy regułowe szybko uderzyły w mur technologiczny. Ich głównym ograniczeniem była absolutna niezdolność do jakiejkolwiek nauki czy auto-optymalizacji. Całą wiedzę inżynierzy musieli ręcznie hardkodować linijka po linijce. Gdy w danym sektorze biznesowym zmieniało się prawo lub pojawiała nowa choroba bez odpowiedniej reguły, system ulegał spektakularnej awarii (tzw. problem kruchości — brittleness). Koszty zatrudniania programistów i ekspertów w celu permanentnego edytowania i utrzymywania baz wiedzy stawały się ogromne. Zjawisko to, w połączeniu z masowym upadkiem komercyjnych firm sprzedających tego typu maszyny typu "Lisp", wywołało na początku lat 90. kolejny kryzys zaufania. Rozpoczęła się Druga Zima AI. Uświadomiono sobie ostatecznie, że próba ręcznego zapisania złożoności całego świata i ludzkiego doświadczenia za pomocą instrukcji IF-THEN jest architektonicznym absurdem.

4. Rewolucja Big Data i Uczenie Maszynowe

W tradycyjnym paradygmacie programowania, inżynier pisze logiczny kod (reguły), dostarcza mu dane wejściowe, a procesor na ich podstawie generuje ostateczny wynik. Wkroczenie w świat uczenia maszynowego (Machine Learning, ML) oznacza dla programisty całkowite odwrócenie tej logiki. Tu człowiek wgrywa do maszyny zarówno dane, jak i poprawne odpowiedzi z przeszłości, zmuszając specjalny model statystyczny do samodzielnego "odkrycia" ukrytych pomiędzy nimi reguł w celu przewidywania przyszłych wyników. Uczenie maszynowe zdefiniowało przejście od z góry ustalonych poleceń do adaptacyjnych macierzy prawdopodobieństwa.
Wydostanie się sztucznej inteligencji z długiej stagnacji i wkroczenie na autostradę wykładniczego wzrostu zostało napędzone kolizją dwóch zewnętrznych rewolucji, na które

deweloperzy algorytmów nie mieli bezpośredniego wpływu. Pierwszym z nich była ogólnoświatowa rewolucja Big Data. Wraz z nadejściem powszechnego, szerokopasmowego internetu, mediami społecznościowymi i urządzeniami Internetu Rzeczy (IoT), globalna przestrzeń cyfrowa zaczęła produkować niewyobrażalne ilości nieustrukturyzowanych obrazów, tekstów i filmów. Bez tej darmowej, zróżnicowanej kopalni cyfrowej wiedzy modele maszynowe nie miałyby paliwa do trenowania.
Drugim, o wiele ważniejszym dla świata inżynierii sprzętowej katalizatorem były innowacje w strukturach krzemowych układów graficznych, zwanych powszechnie procesorami GPU (Graphics Processing Unit). Standardowe, potężne procesory serwerowe CPU są zaprojektowane do niezwykle skomplikowanego, liniowego rozwiązywania pojedynczych instrukcji (sekwencyjnie jedna po drugiej). Przez dekady sprzęt GPU służył jedynie do wyliczania poligonów we wczesnych grach komputerowych 3D, jednak posiadał on zgoła inną naturę — układy te zbudowane były z tysięcy niezwykle prostych, zminiaturyzowanych rdzeni obliczeniowych potrafiących działać jednocześnie. Badacze sieci neuronowych zdali sobie sprawę, że trening modelu z tysiącami sztucznych neuronów nie wymaga skomplikowanej algebry, lecz trylionów niezwykle prostych mnożeń ułamków dziesiętnych. Przerzucenie tych obliczeń na wysoce równoległe układy GPU przyspieszyło proces uczenia modelu tysiąckrotnie, czyniąc poddziedzinę znaną jako głębokie uczenie (Deep Learning) zjawiskiem nie tylko interesującym, ale opłacalnym i skalowalnym dla biznesu.
Kluczowy dla ery deep learningu okazał się dzień 30 września 2012 roku. Rozgrywany był wtedy finał potężnego akademickiego wyzwania ImageNet (ILSVRC). Przed inżynierami z całego świata postawiono zadanie zbudowania autorskiego algorytmu, który z najwyższą możliwą dokładnością sklasyfikuje setki tysięcy cyfrowych zdjęć na 1000 różnych i skomplikowanych kategorii wizualnych (np. różne gatunki psów od siebie). Programiści pod wodzą m.in. Geoffreya Hintona z Uniwersytetu w Toronto zaprezentowali konwolucyjną sieć neuronową (CNN) o nazwie AlexNet. Oprogramowanie to zmiażdżyło konkurencję używającą klasycznych systemów analitycznych, obniżając błąd odczytu systemu o kolosalne 9,8 punktu procentowego w stosunku do drugiego miejsca. Co ważne, tajemnicą sukcesu AlexNet nie była tylko zmiana teoretyczna; to był majstersztyk wczesnej inżynierii AI. Model ten sprytnie podzielił obciążenie treningowe modelu głębokiej sieci symultanicznie pomiędzy dwie potężne desktopowe karty graficzne NVIDIA GTX 580. AlexNet zastosował także funkcję aktywacji o nazwie ReLU, rozwiązując historyczny, potężny problem tzw. "zanikającego gradientu", który uniemożliwiał wcześniej trenowanie sieci bardzo głębokich i ułożonych w wiele skomplikowanych warstw. Programiści zaprezentowali system, w którym pierwsza warstwa neuronów uczyła się rozpoznawać linie brzegowe, kolejna tekstury i ostre kształty, podczas gdy ostatnia generowała logiczną predykcję ostateczną (np. "to jest Siberian Husky"). Rozpoczął się gwałtowny boom na sztuczne, wielowarstwowe sieci. Zrozumiano, że jeśli mamy miliony przykładowych obrazów i potężne wieloklastrowe jednostki graficzne NVIDIA, żaden napisany ludzką ręką kod strukturalny nie pokona matematyki ułożonej przez samo głębokie uczenie.
Kolejny potężny wstrząs przeszedł przez branżę w roku 2016. Starochińska, abstrakcyjna i niesamowicie trudna gra planszowa Go długo uchodziła za "świętego Graala" badaczy sztucznej inteligencji. Przez specyfikę zasad na planszy występuje gigantyczna ilość równoczesnych kombinacji ruchów — tak ogromna, że przewyższa liczbę obserwowalnych atomów we Wszechświecie. Oznaczało to, że zasada "brute force" (siłowego i zmasowanego przeliczania całego drzewa każdej dostępnej opcji od a do z), która to zapewniła w 1997 roku modelowi szachowemu Deep Blue tryumf nad wielkim Garrim Kasparowem, tutaj stałaby się absolutnie bezużyteczna. Firma inżynieryjna DeepMind stworzyła jednak model AlphaGo. System zbudowano w oparciu o połączenie klasycznego probabilistycznego przeszukiwania

drzew (Monte Carlo) połączonego równolegle z wielkimi głębokimi sieciami neuronowymi używającymi uczenia przez wzmacnianie (Reinforcement Learning), co znaczyło, że maszyna trenowała samą siebie, grając w Go miliony razy.
AlphaGo stanęło do morderczego 5-meczowego pojedynku w Seulu przeciwko mistrzowi ludzkości, Lee Sedolowi, deklasując go ostatecznie w stosunku 4-1. Ten turniej nie został zapamiętany jednak z powodu faktu samej maszyny obliczeniowej deklasującej mózg biologiczny, ale z dwóch ikonicznych zagrań na planszy, symbolizujących granice obu cywilizacji. W grze numer 2 AlphaGo umieściło kamień w manewrze zwanym "Ruchem 37" (Move 37). Komentatorzy uznali to początkowo za błąd, ponieważ żaden ludzki ekspert nie zagrałby w taki nielogiczny dla obecnej fazy gry sposób. Dopiero po kilkudziesięciu ruchach okazało się, że maszyna obliczyła i zaprojektowała długofalową i wybitnie inteligentną kontrolę nad centralną osią terytorium planszy. Maszyna po raz pierwszy zaprezentowała nie chłodną kalkulację, ale przebłysk głębokiej "kreatywnej intuicji" w przewidywaniu abstrakcyjnego terytorium dziesiątki tur wprzód. Niemniej w grze numer 4 doszło do ludzkiego przewrotu. Zdezorientowany geniuszem algorytmu Sedol wykonał desperacki "Ruch 78", nazywany dziś pieszczotliwie "Boskim Dotknięciem". Był to nieszablonowy i tak rzadki błąd statystyczny, że potężny model AlphaGo kompletnie sobie z nim nie poradził i pogubił logikę przewidywania aż do spektakularnej porażki, udowadniając dobitnie elastyczność nieustrukturyzowanego umysłu ludzkiego. Oś historyczna AI przekroczyła Rubikon.

5. Era modeli generatywnych — GenAI

Właściwa znajomość matematycznej istoty i topologii modeli generatywnych (Generative AI, GenAI) oddziela obecnie zwykłego dewelopera ("klepacza kodu") od eksperta potrafiącego wygenerować system, testować go oraz powstrzymać ryzykowne zjawisko tzw. halucynacji maszyny. To klucz do całego środowiska inżynieryjnego. Wszelkie narzędzia używane obecnie na przełomie lat 2025/2026, w tym Copilot, Cursor oraz setki bibliotek analitycznych, zawdzięczają swoje unikalne właściwości jednemu wydarzeniu i zmianie architektonicznej. Kamieniem milowym współczesnej ery, dającym początek fenomenowi GenAI, stała się publikacja w 2017 roku przełomowego opracowania naukowego pt. *"Attention Is All You Need"*, wydanego przez zespół badaczy zrzeszonych pod skrzydłami Google Brain. Do 2017 roku inżynierowie chcący wykorzystać maszyny do zadań związanych z płynnym generowaniem i modelowaniem tekstu i konwersacji musieli polegać na tak zwanych rekurencyjnych sieciach neuronowych (RNN lub pamięci LSTM). Fundamentalnym i nierozwiązywalnym problemem tego podejścia była jego "sekwencyjność". Maszyna oparta o logikę RNN "czytała" zbitkę słów powoli, litera po literze, słowo po słowie (np. analizując obszerny wycinek kodu). Niestety, w trakcie docierania do końcowych linii tysięcznego akapitu systemu, mechanizm niemal dosłownie tracił statystyczną pamięć kontekstową słów i logiki zawartych w początkowych partiach pliku, powodując rażące problemy we wnioskowaniu, jednocześnie trawiąc zasoby niezwykle powoli z uwagi na trudność przetwarzania na GPU.
Przełomowy artykuł całkowicie zlikwidował procesy sekwencyjne i odesłał je do inżynieryjnego archiwum. W jego miejsce zaproponowano rewolucyjną architekturę pod nazwą *Transformer*, z absolutnie rewolucyjnym matematycznym jądrem nazwanym "Mechanizmem Samouwagi" (Self-Attention mechanism). Koncepcyjnie architektura Transformer przestawała patrzeć na strukturę pojedynczego słowa sekwencyjnie. Sieć analizowała każdy kawałek danych względem każdego innego kawałka danych jednocześnie, niezależnie od faktu odległości i wielkości dystansu dzielącego je w dokumencie.

Od strony czysto programistycznej, koncepcję tę da się zobrazować poprzez wektorowe macierze wielkich wartości matematycznych reprezentowanych jako słynny trójkąt QKV, czyli:

- **Query (Zapytanie, Q)**: To reprezentacja tego, co dany "token" próbuje wyszukać względem innych wokół.
- **Key (Klucz, K)**: To zestaw cech informujących resztę słów w systemie "oto co ja dokładnie zawieram".
- **Value (Wartość, V)**: Prawdziwy wektor wagi konkretnego pojęcia przekazywanego w procesie ostatecznym.

Analogią ze świata baz danych będzie proces podobny do fuzzy-searchu: Query to wpisane przez usera żądanie z przeglądarki klienta, Key to indeksowana, matematycznie ukryta encja w samej wyszukiwarce relacyjnej, a Value to właściwy wyciągnięty wiersz logiki, z tą ogromną różnicą, że system Transformer wykonuje taką operację dla absolutnie każdego słowa i elementu zdania "w tym samym ułamku sekundy" przypisując każdemu unikalną, ułamkową miarę powiązań i szans (prawdopodobieństwa). Dzięki ogromnej strukturze architektonicznej QKV zdanie typu "Zamek zaciął się w starych dżinsach, przez co mechanizm przestał trzymać zasuwę" stało się w 100% obojętne na podwójne znaczenie terminu zamek (budynek / suwak). Wynikało to z faktu że system powiązał wysoką wartością uwagi Key na słowie "dżinsy" wyciągając najwyższe i najrozsądniejsze Value bez błędu. Co więcej, matematyka macierzowa Transformera perfekcyjnie nadawała się do optymalizacji i wektoryzacji na tysiącach równoległych superprocesorów graficznych (GPU) w serwerowniach, co wywołało nagły renesans potężnego, niemal nielimitowanego skalowania. As of 2025, ten genialny 10-stronicowy artykuł w świecie inżynierów osiągnął status mitu, posiadając około 173 tysięcy cytań, napędzając globalny boom AI.

Zasada skalowania legła u podstaw wielkich korporacji takich jak OpenAI, rozwijających na podstawie Transformera tzw. Duże Modele Językowe (Large Language Models, LLM). OpenAI udowodniło w latach 2018–2020 serią modeli Generative Pre-trained Transformer (GPT), że inteligencja i głębia wirtualnego rozumu rośnie wykładniczo z parametrami matematycznymi modelu podawanymi mu przed właściwym uczeniem. Model GPT-1 oraz GPT-2 były obiecujące, lecz dopiero wydany z ponad 175 miliardami parametrów uczenia gigant GPT-3 wywołał trzęsienie. Zastosowany tu ogrom wolumenu ujawnił w sieci całkowicie nowe zjawisko tzw. inteligencji emergentnej w formie uczenia małopróbkowego (Few-Shot Learning). Maszyna na podstawie jednego okienka polecenia i kilku przykładów programistycznych napisanych bezpośrednio przez użytkownika (bez konieczności kosztownego, dogłębnego dotrenowania zwłaszcza programistycznego kodu fine-tuning) radziła sobie z dedukowaniem zupełnie abstrakcyjnych zjawisk w locie. Po drastycznym powiększeniu bazy danych, w tym dodaniu multimodalności rozumienia grafiki przez GPT-4, narodziła się legendarna już seria darmowych oraz premium aplikacji na czele z ChatGPT.

Na przełomie 2025 i 2026 rynek wyewoluował poza dominację jednej korporacji, dając deweloperom potężny wachlarz potężnych modeli API:

- **Claude (Anthropic)**: Znany przede wszystkim z wydań Claude 3.5 i 4.5 Sonnet. Rodzina ta całkowicie zdominowała w rankingach (benchmarkach) bezwzględne ocenianie logiki czysto programistycznej opartej na programowaniu frontendowym z użyciem nowoczesnych frameworków. Narzędzie rewelacyjnie wczuwa się w pożądany kierunek zadań analitycznych, bez błądzenia wokół wytycznych.
- **Gemini (Google)**: System o potężnym i niesłychanie głębokim oknie kontekstowym (Context Window). Wersje Gemini 1.5 Pro i wyższe potrafią połknąć do okienka wprowadzania naraz pliki ważące tysiące stron lub dziesiątki rozległych drzew kodów, a następnie odpowiedzieć o spójność architektury bez "zapominania" poszczególnych

zmiennych i funkcji ukrytych głęboko na 80 podkatalogu w projekcie.

- **Mistral Large 2 i LLaMA (Meta)**: Rewolucjonizują sferę wolnych licencji oraz zjawiska znanego jako Mixture-of-Experts (MoE). System uaktywnia w danej klatce konwersacji i wektora uwagowego tylko precyzyjnie dedykowany mały wycinek parametrów całego giganta (np. 41 miliardów z globalnych 675 miliardów parametrów w wypadku Mistrala). Drastycznie zmniejsza to potężny koszt API serwerowego wymaganego do obsługi i wsparcia inżynierskiego, zachowując zręczną elastyczność i celność na poziomie znacznie większych i bardziej zachłannych potworów konkurencji.

Architektury głębokich transformatorów objęły również modele obrazu oparte jednak na zupełnie innej magii technologii zwanej systemem dyfuzyjnym (Diffusion Models), obsługiwanym komercyjnie m.in. na aplikacjach typu DALL-E czy popularnym programie Midjourney. Wyobraź sobie kucharza potrafiącego przyrządzić potrawę i równie bezbłędnie zredukować spalone ciasto z powrotem do świeżej, nieupieczonej mąki, jajek i cukru. Modele dyfuzyjne to potężne, złożone z dwóch etapów sieci bazujące na matematyce równań termodynamiki i rozkładu Gaussa. Algorytmy w pierwszym potężnym kroku uczenia niszczą obraz piksel po pikselu, generując zakłócenia wizualne aż do otrzymania "losowej zupy szumu graficznego" zwanego Gaussian noise (proces zwany Forward diffusion). Następnie sieć deweloperska (np. architektura predykcyjna zwana U-Net) używając wyrafinowanego algorytmu minimalizacji strat jest zmuszana przez miliony długotrwałych cykli, do idealnego przewidywania usunięcia szumu w tył (proces Reverse diffusion lub Denoising) w odniesieniu i warunkowaniu do słownego wektora tekstowego opartego na macierzach systemu (Cross-Attention). Skutkuje to potężną, niczym niemal nieograniczoną możliwością i zdolnością "wyczarowywania z magicznego szumu i drobinek" całkowicie niewidzianych w sieci struktur hiperrealistycznych portretów wyłącznie przez wywołanie żądania tekstowego od użytkownika podawanego np. przez zapytanie "astronauta na koralach".

Dla młodego programisty lata 2025/2026 to w ostateczności starcie mocarstw tworzących środowiska programistyczne w całości oparte na modelach AI. Narzędzia zrewolucjonizowały tzw. Integrated Development Environments (IDEs), a programiści przenieśli się z etapu "autocomplete" na potężną i dynamiczną agentowość:

- **GitHub Copilot (Microsoft)**: Standard korporacyjny i najbardziej ugruntowany system wbudowany przez rynkowego lidera, zintegrowany ze środowiskiem repozytoriów. Copilot, wpisany bezpośrednio w środowisko takie jak Visual Studio, przewiduje obszerne bloki tzw. kodu masowego (boilerplate), rozwiązując odgórne, uciążliwe problemy bez ruszania palcem.
- **Cursor AI**: Obecnie król tak zwanych zaawansowanych edytorów, bazujący technicznie jako odnoga na kodzie silnika VS Code. Jego przewagą jest dedykowany inżynieryjny interfejs obsługi na potężnym asystencie zdolnym dokonywać dynamicznych zmian oprogramowania symultanicznie na kilkudziesięciu wyciągniętych przez system plikach, pozwalając zaawansowanemu programiście wydać tylko jeden rozkaz z klawiatury CMD+K, by stworzyć spójną, logiczną usługę z testami jednostkowymi na gotowo w terminalu w tle.
- **Windsurf**: Silnie dotowany faworyt deweloperów o rosnącym potężnym rynku z otwartymi planami komercyjnymi pod egidą firmy Codeium. Rewelacyjny w projektach, w których programista oczekuje agenta zachowującego stały i żelazny wektor potężnego pamiętania setek wcześniejszych zdarzeń w czacie; wybitnie informuje operatora krok po kroku z potoku działań wykonywanych przez maszynę do bazy repozytorium przez co "nigdy nie gubi wątku" inżynieryjnego budowy i planowania struktury systemu plikowego.

Transformacja wymusiła i scementowała rolę "AI-Assisted Developera". Nie użycie maszyn

przez firmę i programistów jest równoznaczne w dzisiejszych realiach z natychmiastowym wydaleniem z rynku wobec niewspółmiernego dysonansu produktywności. Twoja edukacja leży właśnie w zrozumieniu tej osi czasowej, a co za tym idzie — pojęciu niebezpieczeństw wynikających ze ślepego przyzwolenia maszynie na dyktowanie architektury i utraty zdrowego, krytycznego horyzontu we wnioskowaniu, gdy model bez wątpliwości z powodu braku wagi tokenowej zaserwuje wadliwą i potencjalnie tragiczną wyrocznie.

6. Oś czasu — kamienie milowe AI

Poniższa tabela stanowi syntetyczne podsumowanie najważniejszych kamieni milowych w badaniach nad AI wraz ze zwięzłym wyszczególnieniem znaczenia technologicznego. Stanowi ona świetne odniesienie pod kątem chronologii wydarzeń.

Rok	Wydarzenie i Innowator	Znaczenie dla inżynierii systemów i algorytmów sztucznej inteligencji
1950	Publikacja eseju *Computing Machinery and Intelligence* przez Alana Turinga	Formalne zasugerowanie "Gry w Naśladownictwo" (Testu Turinga) jako racjonalnego benchmarku ewaluacji w sztucznym komunikowaniu.
1955	Udane testy programu dedukcyjnego *Logic Theorist* (Newell, Simon)	Pierwsze dobitne udowodnienie w kodzie zasady działania maszyny symbolicznej wykazującej poprawny proces wyszukiwania aksjomatów z dzieł matematycznych bez użycia procesora jako kalkulatora zjawisk ciągłych.
1956	Cykl letnich konferencji badawczych w laboratoriach Dartmouth	Ukucie nazwy dyscypliny "Sztucznej Inteligencji" i nieformalny, rynkowy zapłon dla poszukiwania "Świętego Graala" inżynierii oprogramowania.
1965	System oparty o twarde reguły *DENDRAL*	Tryumf oddzielenia warstwy wiedzy i interfejsów dedukcyjnych silnika przy badaniu widm rozkładów cząsteczek chemicznych w spektrometrii.
1966	Prezentacja systemu *ELIZA* imitującego funkcje psychoterapeuty na MIT	Potężny szok społeczny udowadniający, w jaki sposób sztuczki ze ślepym wyszukiwaniem wzorca potrafią wymusić głęboki efekt fałszywej więzi na nieostrożnym człowieku.

Rok	Wydarzenie i Innowator	Znaczenie dla inżynierii systemów i algorytmów sztucznej inteligencji
1972	Powstanie *MYCIN* w sferze diagnostycznej bakterii we krwi	Ustanowienie wzorca pisania oprogramowania warunkowego typu "Jeżeli-Wtedy" w zamkniętej medycynie, acz zderzenie się ze światem barier etycznych uniemożliwiających implementację w kod szpitala.
1973	Drukowany akademicki Raport sir Lighthilla	Brutalna i poparta dowodami fizyki matematycznej dekonstrukcja marzeń programistów ukazująca niemożliwość wejścia w potężny złożony świat rzeczywisty doprowadzająca do zjawiska zimnego cięcia budżetów na długie dekady (Pierwsza Zima AI).
1997	Program i serwer potężny *Deep Blue* wygrywa zawody w ostre szachy (z Kasparowem)	Tryumf paradygmatu klasycznej, algorytmicznej i czystej siły z wysoce obciążonym przeliczeniowo sprzętem komputerowym nad ustrukturyzowaną taktyką arcymistrza ludzkiego.
2012	Model *AlexNet* dominuje z potężną skutecznością wezwanie analityczne ImageNet	Wybudzenie dyscypliny na zjawiska sieci neuronowych. Konwolucyjny tryumf z równoległym wsparciem dwóch wczesnych rdzeni układów GPU NVIDIA w rewolucyjnym i skalowanym środowisku przetwarzania obrazów.
2016	Triumfalna statystyczna gra programu Google *AlphaGo* nad wielkim mistrzem planszy Lee Sedolem	Niespotykane użycie sieci neuronowych na poziomie abstrakcji i braku przewidywalności (Ruch 37 kontra ludzki nieokiełznany Ruch 78), otwierający oczy świata na głębokie intuicyjne potężne zjawisko ML.
2017	Opublikowanie epokowego wielkiego artykułu *Attention Is All You Need* (Google Brain)	Całkowity pogrom sieci o logice rekurencyjnej poprzez zjawisko architektury powiązanej jako "Transformer" bazującej na

Rok	Wydarzenie i Innowator	Znaczenie dla inżynierii systemów i algorytmów sztucznej inteligencji
		błyskawicznej równoległości uwagi.
2020	Potężny test komercyjny 175-miliardowego środowiska parametrów z siecią modelu GPT-3	Zdefiniowanie fundamentalnych pojęć elastycznej analityki i inteligentnej predykcji pojęciowej określanej w oprogramowaniu jako potężny model LLM (Large Language Model) dla szerszego dostępu inżynierów i studentów na API.
2022	Powszechne otwarcie okna wejściowego z fenomenem *ChatGPT*	Erupcja i całkowite przejęcie mainstreamu mediów, deweloperki i procesów uwarunkowanych w erze tzw. GenAI (generative) do tworzenia nieskończonej płynnej nowej tekstowej wartości na niesamowitym tempie iteracji i uczenia powszechnego na miliardach punktów dostępowych.
2024	Zażarta walka asystenckich multimodalnych układów analitycznych (Anthropic, Meta, Mistral AI)	Rynek odrywa się od potężnego monopolisty by wdrażać i rozwijać otwarte warianty gigantów potrafiące obsłużyć dziesiątki tysięcy wielkich plików tekstowych pod postacią jednego wywołania tokenowego.
2025/2026	Narzędzia platform w roli agentów deweloperskich (GitHub Copilot, inżynieria Windsurf, Cursor IDE)	46% kodu pisanego w branży staje się autorsko zależnym od autouzupełniania AI. Uczeń przekształcony de facto formalnie staje w hierarchii ról i ofert korporacyjno biznesowych w tak zwany standard "AI-Assisted Developera".

7. Ćwiczenia praktyczne

Jako przyszły inżynier kodujący oprogramowanie twoim absolutnym priorytetem jest natychmiastowe empiryczne weryfikowanie koncepcji architektonicznych i statystycznych opisywanych w dokumentacji technicznej wewnątrz zaufanego, niezależnego środowiska produkcyjnego. Rozwiąż poniższe zdania opierając swoje analizy o fundamentalny cel: zrozumienia co różni "prymitywne kodowanie twarde" od "probabilistycznej estymacji wag

modelu macierzowego".

- **Ćwiczenie 1: Oś Czasu AI i budowa naiwnego systemu regułowego w środowisku Jupyter** Uruchom lokalny proces serwerowy środowiska Jupyter Notebook lub zaloguj się pod wolnym adresem kompilatora Google Colab wykorzystując najnowszy kernel z Python 3. *Krok 1:* Zdefiniuj trzy zgrabne komórki typu Markdown operujące pod silnymi nagłówkami objaśniając definitywnie i logicznie co twoim inżynieryjnym i logicznym zdaniem oddzielało naiwność pierwszej Zimowej Ery Algorytmicznej AI wobec Złotego Boom'u Głębokich Sieci Neuronowych opartego o wczesną wielordzeniowość sprzętową w badaniach ImageNet'a (opis odwołaj do wektorów na układzie GPU). *Krok 2:* Zbuduj prymitywną i prostą implementację w czystym i zwięzłym Pythonie programu stanowiącego niepłynny parser klasycznego "chatbota", nawiązującego ideologicznie na potężnym skrypcie starej ELIZY. Użyj tylko i wyłącznie uwarunkowanych w strukturę słowników z kluczami i funkcją obsługi twardych znaków używając potężnej biblioteki wyrażeń do wzorców re (regex). Doprowadź by odpowiedź wejściowa "I feel scared about my job in programming" zwracała warunkowy string z nałożeniem dynamicznym zmiennej w string f-formatowaniu, generując sztywne "Why do you feel scared about your job in programming?". *Krok 3:* Zwieńcz zadanie i pracę wyodrębnionym długim wielozdaniowym docblockiem docelowym podsumowującym dlaczego taki regex'owy, syntaktyczny sposób dopasowywania wzorców używany przed 1970 nie ma z perspektywy oprogramowania nic wspólnego ze statystycznym pojęciem głębokiego wektorowego powiązania semantycznego (Deep Learning), pomimo zewnętrznego i identycznego wyglądu działania logiki przez oko zdezorientowanego człowieka w czacie. Oceniana będzie ostrość odróżnienia semantyki od mechanicznej składni kodu systemowego.
- **Ćwiczenie 2: Inżynieria kontekstowa — zaawansowana analiza z benchmarkowaniem** Twoim nowym, potężnym narzędziem jako asystenta "AI-Assisted" nie jest debuger z twardymi zrzutami pamięci (memory dumps), lecz świadomość logiki wektorów LLM z konkurencyjnych stajni rynkowych (np. zamknięty silnik chatu w przeglądarce opartego na modelu OpenAI vs oprogramowanie Cursor wspomagane modelami Anthropic lub odwrotnie w ramach dostępnego za darmo limitu zapytania API od Codeium na Windsurf). *Krok 1:* Wklej kompletnie błędnie i skomplikowanie ustrukturyzowany architektonicznie kod asynchroniczny z Pythona (asyncio z zakleszczeniem typu 'deadlock', niemożliwy w znalezieniu z uwagi na bycie błędem "cichym" czyli silent-bug niepowodującym klasycznego stack trace wywołanego). *Krok 2:* Zaprojektuj, precyzyjnie sformułuj i wprowadź tak zwany zaawansowany prompt operacyjny wymuszający i precyzujący by obydwa asystenty myślały powolnymi wielowarstwowymi zrzutami o logice kodu, tzw. systemem "krok-po-kroku z objaśnieniem założeń wstępnych programisty biznesowego". *Krok 3:* Stwórz i przygotuj w notatkach ustrukturyzowaną wielokolumnową tabelę na logice formatowania systemu z Markdown opisującą jak model A zachował spójność w odpowiedzi do warunków okna na polecenie, a w którym momencie drugi rywal wywołał absolutne sztuczne "halucynacje maszynowe" tworząc pakiety klas w locie lub ignorując kluczowy argument blokowania zmiennych wątkowych. Na sam koniec zapisz jedno, decydujące zdanie oceniając zdolność trzymania całego wektora konwersacyjnego przed "utratą sztywnego wątku po ucięciu potężnej tabeli uwagowej systemu z braku pamięci".
- **Ćwiczenie 3: Twórczy i krytyczny esej techniczny dewelopera** Napisz wysoce ustrukturyzowany, głęboko autorski raport esejowo techniczny i oparty w dokument jako plik sformatowany do druku jako (.md). Praca o maksymalnie ostrej i zwięzłej wadze około 300 rygorystycznie sprawdzanych słów, musi celnie odpowiedzieć na postawioną

na wstępie akademickim postawioną problematykę techniczną: *"Który z architektonicznych lub sprzętowo potężnych technologicznych skoków paradygmatycznych (od twardych reguł do uczenia, na modelach sieciach, Transformatorach QKV aż do potężnych skalowanych maszyn asystujących jak Cursor i Copilot) w obszernej historii sztucznej inteligencji ma i wywoła z inżynierskiego punktu widzenia największy decydujący oraz obiektywny moment przełamania w pracy twoim ostatecznym docelowym środowisku inżynierskim. Z jakim powodem obronisz w logicznym stopniu tę zwięzłą konkluzje?".* Maksymalnie punktowane są konkluzje odwołujące do faktu "pustego uwarunkowania bazy wiedzy" przeciwstawionego rewelacyjnej symultaniczności na macierzowym mechanizmie potężnego wielo-procesowego wsparcia. Zastrzeżenie: Tekst wygenerowany z oprogramowania asystenta musi przed przesłaniem do audytu na ocenę zyskać wyczerpujący i weryfikowalny wgląd w krytyczną analizę faktu autorskich zniekształceń modelu (aby uniknąć naiwnego kopiowania logiki ze struktur wyplutego chatbota przez programistę uciekającego od nauki rzetelnego kodu i myślenia inżynierskiego w nauce nowej koncepcji).

8. Słowniczek pojęć

- **Sztuczna inteligencja** (Artificial Intelligence) — Rozległa dziedzina badań informatycznych skupiająca się na programowaniu operacji i systemów cyfrowych naśladujących typowo zachowania, racjonalne wnioskowania i rozbudowane skomplikowane akcje, które dotychczas wydawały się wyłącznym atrybutem człowieka. W programowaniu to przesunięcie ze środowiska sztywnej kontroli kalkulacji procesora na elastyczne modelowanie celów wielooperacyjnych opartych ze zbirów predykcji.
- **Uczenie maszynowe** (Machine Learning) — Przełomowa klasa operacyjna leżąca w samej sztucznej inteligencji operująca o drastyczne oderwanie się od kodowania instrukcji "JEŚLI-TO" przy procesie rozwiązań pożądanego problemu po to by pozwolić programom uczyć samej wagi ukrytej probabilistyki z wygenerowanych zasobów bazy doświadczeń. To matematyczna nauka oparta poprzez testy statystyczne zamiast statycznego narzucania ścisłych dróg przejścia inżyniera we wstępnym kodzie aplikacji.
- **Głębokie uczenie** (Deep Learning) — Dominujący we współczesnym sprzęcie odział gałęzi ze środowisk ML napędzający się w całości niezwykle rozległymi uwarstwionymi warstwami powiązanych macierzowo struktur decyzyjnych czyli tzw. sieci wielowarstwowych operujących na procesach rozpoznania i abstrakcyjnego uogólnienia np. grafiki i sygnału wielopoziomowego wizualnego bez wstępnych i męczących inżyniera procesów ekstrakcji parametrów ręcznych cech modelu obiektu.
- **Sieć neuronowa** (Neural Network) — Obliczeniowo numeryczna moc matematyczna będąca filarem Głębokiego Uczenia czerpiąca pierwotne mocne inżynieryjne inspiracje operacji w topografii układu komórek nerwowych w biologicznym organizmie mózgu. Zbiór zoptymalizowanych węzłów posiadających własne nieliniowe zjawiska aktywacji optymalizujących całe algorytmy po drastycznym błędzie poprzez niezwykle zaawansowany wieloczęściowy tzw. algorytm odwróconej ujemnej wstecznej propagacji błędu wielowektorowego.
- **System ekspercki** (Expert System) — Sztywno architektoniczna dominująca do lat 90 koncepcja niezwykle wąskiej i ukierunkowanej logicznie zamkniętej inteligencji decyzyjnej zbudowana poprzez oddzielenie potężnej pasywnej encyklopedycznej Bazy Ręcznej

Wiedzy dedukcyjnej narzuconej odgórnie ze spotkań połączonej z zewnętrznym bardzo logicznie szybkim modułem zwanym Silnikiem Rozdziału Wnioskowania (jak stary program MYCIN diagnozujący dawki dla rzadkich bakterii z badań w krwi chorych testowych).

- **Big Data** — Eksplozja z ogromnymi potężnymi zasobami wektorów plików historycznych obrazków, wolumenami tekstowymi i logami pomiarów sensorów sprzętu w świecie Internetu o niespotykanej wcześniej różnorodności, a zwłaszcza potężnej objętości dyskowej po roku dwutysięcznym stanowiącej ostateczne niezbędne paliwo treningowe do nakarmienia pustej statystycznej wagi probabilistycznej na głębokich uwarstwieniach wielopoziomowych w epoce przełomu po roku 2012 na układach chmurowych pod rygorem gigantycznych dostaw.
- **GPU** (Graphics Processing Unit) — Niegdyś wyspecjalizowany czysto procesor renderowania poligonów cyfrowych a po rewolucji 2012 niezastąpiony wielowątkowy koprocesor o ogromnej mocy zminiaturyzowanych setek potężnych jednostek asymetrycznych i arytmetycznych liczących tysiące punktowych i banalnych instrukcji symultanicznie jednocześnie; co okazało się jedynym ratunkiem na pokonanie ograniczonego potoku sekwencyjnego, potężnego układu centralnego komputera ze strony klasycznej architektury procesorowej maszyny PC.
- **Model generatywny** (Generative Model) — Niezwykle rozwinięta koncepcyjna i programowa kasta narzędzi wykorzystujących nauczone prawdopodobieństwa w sposób nie po klasycznym celu podziału albo poprawnej selekcji uogólnienia obiektowego we wskazaniu parametru z szumu pliku do decydowania tak jak np algorytm bankowy tylko po operacjach i mechanizmach z intencją ostatecznej masowej absolutnej twórczej i ciągłej potężnej kreacji kodu operacyjnego dla klienta docelowego lub generując obraz hiperrealistyczny pliku, odróżnia i podaje dane nieprzewidywalne.
- **Transformer** — Obiekt kultu epoki i gigantyczny wyczyn zespołu od artykułu naukowego Attention Is All You Need łamiący ostatecznie bariery blokującego algorytm wolnego i uciążliwie zapominającego procesu odczytu pojedynczych znaków na pętlach o ułamku pamięci rekurencyjnej na tryumfalne uwolnienie przetwarzania do jednoczesnego masowego nałożenia siatki punktowania prawdopodobieństw zwanych wagami przez obustronne ujęcie koncepcji wielokierunkowej uwagi rozbudowanej pojęciowo jako QKV.
- **LLM** (Large Language Model) — Gigantyczna operacyjna usługa w ramach ustrukturyzowanych serwerowych instancji używająca odgórnej niesłychanie zoptymalizowanej konstrukcji bazującej po zjawisku na topologii transformatorowej w setkach miliardach węzłów ułamkowych do niemal stuprocentowo precyzyjnego przewidywania położeń odpowiednich logicznych ciągów zapytań o składnie programu w ujęciu bezszwowym i bez przerw komunikacyjnych do zjawisk "wspieranej inżynierii programisty od GPT po LLaMĘ z Metą na API wejściu".
- **Zima AI** (AI Winter) — Zmrożenie o ogromnym impakcie na uniwersytecie, wielokrotne krytyczne zniechęcenie wielkich dotujących kapitałów decydujących do całkowitego bezlitosnego odcinania z finansowania laboratoriów programistycznych i algorytmików tuż po uświadomieniu bezlitosnego pęknięcia napompowanej po latach potężnie złudnej marketingowo na rynkach inwestycyjnych gigantycznej fali optymizmu za obiecane ale ostatecznie z braku fizyki niemożliwe wezwania obliczeniowe na serwerach z dawnych układów pamięci operacyjnej z rygorystycznymi twardymi strukturami w językach Lisp.
- **Token** — Metryka miary analitycznej w najniższym warunkowym wektorowym stopniu operacyjnym przy wlewaniu surowego bloku cyfrowego polecenia, wyrazowa, często stanowiąca prefiks i zlepek logiki z myślnikiem przekształcana by zbudować

bezwzględnie rygorystyczny proces ujęcia maszynowego cyfrowego zera do wirtualnego wielopoziomowego układu wejścia algorytmu modelowania ucinana do narzucenia górnych barier płatnej pojemności serwera zapytaniowych sesji (jak gigantyczne limitowane tokeny okienkowe pamięci Claude czy środowiska w edytorze Copilot pod opłatę abonencką za zużyte wolumeny sesji na zjawiskach programistycznego parsowania).

- **Prompt** — Warstwa nakazowa; inżynieryjny lub lingwistyczny ścisły precyzyjny zbiór warunków granicznych komend używanych i sterujących w wejściowym otwartym żądaniu z intencją sterowania uwikłanego systemu wywołania LLM generatywnego do wyciągnięcia lub zablokowania niechcianych form odpowiednich stylów w oknie dewelopera podczas konwersacyjnego procesu zwrotnego, np. przymus do wycięcia pustych zbędnych bibliotek kodu przed zrzutem bloku i pliku docelowego z maszyny z włączeniem weryfikacji i asercji bezpieczeństwa programistycznego kodu by system powstrzymał ślepą konfabulację architektoniczną w IDE po uświadomieniu o potencjalnym rygorze języka biznesu.
- **Pre-training** (Uczenie wstępne) — Bezwzględny i masywny pod względem inwestycji gigantycznej z perspektywy mocy karty graficznej pierwszy z długotrwałych niefiltrowanych wprost w ramy logiki wyszukiwarki okres ładowania do maszyny niewyobrażalnie pustego gąszczu wielkich starych struktur historycznych z otwartej biblioteki w internecie uświadamiając podwaliny algorytmu wyłącznie jak poprawnie generalizować logikę ludzką bez żadnego specjalizowania się pod wąskie cele wywoławcze czy zagnieżdżonego agenta terminali.
- **Fine-tuning** (Dostrajanie dziedzinowe) — Precyzyjny o niewielkiej w teorii masie względem uczenia pierwotnego model zabieg rzemieślniczy dewelopera mający zadanie poświecić wielkiemu nauczonemu LLM zamkniętą garść kilku wielokrotnych tysięcznych doskonałych w parametrach autorskich logik zachowań wyciągniętych przez specjalistów bezpieczeństwa dla ukrócania błędu "zjadania logiki na okienku klienta z wiedzy firmowej API w środowisku lokalnym lub wymuszenia by serwer odpisywał we wspomaganym języku rygorystycznie np bez komentarzy i docstring w locie deweloperskim przed pchnięciem commit do potężnego rozproszonego środowiska pracy grupy docelowej" bez ingerencji w globalny mechanizm i macierz centralną transformera a jedynie nadpisując lekki profil dedykowany i dopasowujący wagę końcowych potężnych procesów wnioskowania warunkowego zachowań deweloperskich modela dla klienta asystenckiego na dedykowanej branży zamkniętej za płatnym zabezpieczeniem logiki.

Cytowane prace

1. History of AI: Timeline and the Future | Maryville Online, https://online.maryville.edu/blog/history-of-ai/
2. AI vs. Machine Learning vs. Deep Learning vs. Neural Networks - IBM, https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks
3. How Will AI Affect the US Labor Market?, https://www.goldmansachs.com/insights/articles/how-will-ai-affect-the-us-labor-market
4. AI Is Writing 46% of All Code: GitHub Copilot's Real Impact on 15 Million Developers | by Reliable Data Engineering | Medium, https://medium.com/@reliabledataengineering/ai-is-writing-46-of-all-code-github-copilots-real-impact-on-15-million-developers-787d789fcfdc

5. AI | 2025 Stack Overflow Developer Survey, https://survey.stackoverflow.co/2025/ai https://www.wearetenet.com/blog/github-copilot-usage-data-statistics
6. Generatywna sztuczna inteligencja a polski rynek pracy - NASK, https://www.nask.pl/media/2025/06/Generatywna-sztuczna-inteligencja-a-polski-rynek-pracy.pdf 12. Timeline of artificial intelligence - Wikipedia, https://en.wikipedia.org/wiki/Timeline_of_artificial_intelligence
7. The History of Artificial Intelligence | IBM, https://www.ibm.com/think/topics/history-of-artificial-intelligence
8. What is the history of artificial intelligence (AI)? - Tableau, https://www.tableau.com/data-insights/ai/history
9. Attention Is All You Need - Wikipedia, https://en.wikipedia.org/wiki/Attention_Is_All_You_Need
10. Logic Theorist – Knowledge and References - Taylor & Francis, https://taylorandfrancis.com/knowledge/Engineering_and_technology/Artificial_intelligence/Logic_Theorist/
11. Logic Theorist: The program that rewrote the foundations of mathematics - Big Think, https://bigthink.com/books/logic-theorist/
12. Principia Mathematica - Wikipedia, https://en.wikipedia.org/wiki/Principia_Mathematica
13. Eliza: The First Chatbot (1964) and a Lesson Beyond the Hype | by Sitraka FORLER, https://medium.com/@sitrakaforler/eliza-the-first-chatbot-1964-and-a-lesson-beyond-the-hype-8274ea56267e
14. ELIZA - Wikipedia, https://en.wikipedia.org/wiki/ELIZA 21. ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine - Stanford University, https://web.stanford.edu/class/cs124/p36-weizenabaum.pdf
15. ELIZA: The First Step in Human-Computer Interaction Through Natural Language Processing - GeeksforGeeks, https://www.geeksforgeeks.org/artificial-intelligence/eliza-the-first-step-in-human-computer-interaction-through-natural-language-processing/
16. The Story Of ELIZA: The AI That Fooled The World - London Intercultural Academy, https://liacademy.co.uk/the-story-of-eliza-the-ai-that-fooled-the-world/
17. Systemy ekspertowe: Czym są, jak działają i gdzie znajdują zastosowanie? - PW, https://pawelwoloszyn.pl/slownik/systemy-ekspertowe-co-to-jest/
18. Expert System In Artificial Intelligence - Scaler Topics, https://www.scaler.com/topics/artificial-intelligence-tutorial/expert-system-in-ai/
19. Expert system in Artificial Intelligence - OER Commons, https://oercommons.org/courseware/lesson/130473/student/?section=2
20. Rule-Based System in AI - GeeksforGeeks, https://www.geeksforgeeks.org/artificial-intelligence/rule-based-system-in-ai/
21. Expert System in AI - AlmaBetter, https://www.almabetter.com/bytes/tutorials/artificial-intelligence/expert-system-in-ai
22. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, https://people.dbmi.columbia.edu/~ehs7001/Buchanan-Shortliffe-1984/Chapter-05.pdf
23. AlexNet and ImageNet Explained - YouTube, https://www.youtube.com/watch?v=c_u4AHNjOpk
24. Understanding AlexNet: The 2012 Breakthrough That Redefined AI - Medium, https://medium.com/@igquinteroch/understanding-alexnet-the-2012-breakthrough-that-redefined-ai-d0e267e2470a

25. AlexNet and ImageNet: The Birth of Deep Learning - Pinecone, https://www.pinecone.io/learn/series/image-search/imagenet/
26. AlexNet: The First CNN to win Image Net - Great Learning, https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/
27. (PDF) ImageNet Classification with Deep Convolutional Neural Networks - ResearchGate, https://www.researchgate.net/publication/319770183_Imagenet_classification_with_deep_convolutional_neural_networks
28. AlphaGo versus Lee Sedol - Wikipedia, https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol
29. The significance of move, by AlphaGo and move 78 by Lee Sedol - Jessa Gavilla, https://jessa-gavila.medium.com/the-significance-of-move-37-by-alphago-and-move-78-by-lee-sedol-19a21cdb289b
30. Lee Sedol and AlphaGo: The Legacy of a Historic Fight! - Go Magic, https://gomagic.org/alphago-and-lee-sedol/
31. Do you think Alphago's “move 37” will end up being the most famous Go move of this century? : r/baduk - Reddit, https://www.reddit.com/r/baduk/comments/8xyq64/do_you_think_alphagos_move_37_will_end_up_being/
32. Attention is All you Need - NIPS, https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf
33. Transformers Explained Simply: From QKV to Multi-Head Magic - Medium, https://medium.com/@rajputshubham219/intuition-is-all-you-need-4920f6ad7b18
34. Attention is all you need (Transformer) - Model explanation (including math), Inference and Training - YouTube, https://www.youtube.com/watch?v=bCz4OMemCcA
35. Attention is all you need? Explaining the concept behind Generative AI models like ChatGPT and Gemini, for a Child, a College Student, an Engineering Student and a Data Scientist | by Christian Zambra | travel.therapy.ai | Medium, https://medium.com/travel-therapy-ai/attention-is-all-you-need-e324c40cd7ca
36. Attention is all you need explained - YouTube, https://www.youtube.com/watch?v=sznZ78HquPc
37. Evolution of GPT Models, GPT 1 to GPT 4 | by Vipul Koti - Medium, https://medium.com/@vipul.koti333/evolution-of-gpt-models-gpt-1-to-gpt-4-0238ee07a29b 45. I benchmarked Claude 3.5 Sonnet vs Gemini 1.5 Pro for everyday web development tasks (Speed, Context, & Agentic Coding) : r/ClaudeAI - Reddit, https://www.reddit.com/r/ClaudeAI/comments/1rcdd6r/i_benchmarked_claude_35_sonnet_vs_gemini_15_pro/
38. 10+ Large Language Model Examples & Benchmark - AIMultiple, https://aimultiple.com/large-language-models-examples
39. Top 50+ Large Language Models (LLMs),https://explodingtopics.com/blog/list-of-llms
40. LLM Leaderboard - Comparison of over 100 AI models from OpenAI, Google, DeepSeek & others - Artificial Analysis, https://artificialanalysis.ai/leaderboards/models
41. AI Diffusion Models Explained (Midjourney, DALL-E, Runway, Sora): How AI Image Generation Works - YouTube, https://www.youtube.com/watch?v=VyNATiMJO_k
42. How Do Diffusion Models Work? Simple Explanation: No Mathematical Jargon, Promised!, https://towardsai.net/p/l/how-do-diffusion-models-work-simple-explanation-no-mathematical-jargon-promised

43. Diffusion Models Demystified: Understanding the Tech Behind DALL-E and Midjourney, https://www.kdnuggets.com/diffusion-models-demystified-understanding-the-tech-behind-dall-e-and-midjourney
44. Cursor vs Windsurf vs GitHub Copilot - Builder.io, https://www.builder.io/blog/cursor-vs-windsurf-vs-github-copilot
45. Cursor vs. Windsurf vs. GitHub Copilot - Educative.io, https://www.educative.io/blog/cursor-vs-windsurf-vs-copilot
46. Choosing the Right AI IDE for Your Team: Cursor vs. Windsurf vs. Copilot | HackerNoon, https://hackernoon.com/choosing-the-right-ai-ide-for-your-team-cursor-vs-windsurf-vs-copilot
47. GitHub Copilot vs Cursor in 2025: Why I'm paying half price for the same features - Reddit, https://www.reddit.com/r/GithubCopilot/comments/1jnboan/github_copilot_vs_cursor_in_2025_why_im_paying/
48. Artificial Intelligence Timeline (1950–2025) – AI History and Milestones | AIny, https://ainy.no/en/artificial-intelligence-timeline-1950-2025/