Research Report: The History of Artificial Intelligence – From Expert Systems to the Era of Generative Models (GenAI)

1. Introduction

Understanding the fundamental mechanisms of artificial intelligence has gone from being an optional career path to an absolute requirement for any software engineer entering the market in 2025. From the perspective of a future programmer, knowing the historical evolution of these algorithms allows one to fully understand why modern programming environments work the way they do, what hidden architectural limitations they have, and how to optimize code that works with powerful language models.

Artificial intelligence (AI), in its simplest, working definition, is a field of computer science concerned with the design and development of computational systems capable of performing tasks that would traditionally require human cognition, such as reasoning, pattern recognition, or language understanding. The vast majority of current commercial solutions are classified as narrow AI, which specializes in excelling at solving a single, precisely defined problem. These models do not mimic human consciousness but utilize extremely advanced probabilistic mapping, multivariate statistics, and artificial neural networks to generate the most probable answer based on the analysis of massive sets of input data.

For a young programming aspirant entering the job market after 2025, this context is both exceptionally pragmatic and revolutionary. Contrary to pessimistic forecasts, AI isn't eliminating the programming profession, but drastically changing its nature and elevating the value of engineering itself. Pure "coding," based on memorizing syntax and manually creating repetitive structures, is becoming a fully automated activity. The modern software engineer is transforming into a systems architect and business logic verifier. This phenomenon has led to the formal emergence of a new, dominant professional category: the AI-Assisted Developer. The role of such an engineer involves defining the problem, skillfully instructing AI models, and then rigorously auditing the security, performance, and integration of the machine-generated code with the rest of the repository.

These changes are powerfully reflected in the global usage statistics for AI tools. The market is adopting these solutions at a pace unprecedented in the history of computing. GitHub's Copilot platform, one of the leaders in coding assistants, reached an impressive 20 million users in July 2025, having previously recorded 400% year-on-year growth. In corporate environments, AI assistants are no longer an experiment—by the turn of 2025, approximately 90% of Fortune 100 companies were using them.

The impact of AI on everyday software architecture is easily quantified. On average, around 46% of all new code on platforms supported by these tools is generated directly by AI. For specific object-oriented frameworks like Java, this figure rises to a staggering 61%, and in Python, it's 55%.

Productivity studies, including Stack Overflow's powerful 2025 report based on surveys of tens of thousands of developers, ruthlessly debunk the myth of slow adoption. Over 52% of developers categorically confirm the positive impact of AI assistants on their work productivity. Controlled research environments, in turn, show that developers using generative models

complete tasks 55% faster, and the time required for code reviews (pull requests) drops by nearly 75%—from 9.6 days to an average of 2.4 days. From an employer's perspective, ignoring such productivity gains is business suicide.

Programming Task Category	Percentage of developers using AI (2025)
Deployment and Monitoring (Deployment / Ops)	75.8%
Project architecture planning	69.2%
Predictive analytics	65.6%
Code Review and Approval (Commit/Review)	58.7%
Writing unit and integration tests	44.1%
Generating documentation (Documenting code)	38.5%
Debugging and error fixing	36.4%
Learning new concepts and languages	32.3%

The Polish labor market reflects these global megatrends with equal force. A report prepared by the NASK research institute indicates that approximately 817,500 positions in the Polish market (representing over 5 million jobs in total, including a large portion of technical and office roles) are highly susceptible to direct interaction with generative systems. Possessing hard programming skills (e.g., knowledge of C++ or JavaScript syntax) is becoming merely a necessary, but no longer sufficient, requirement for securing a high-paying position. For an IT technical school student, this means that to maintain a competitive advantage in 2024, they must understand the algorithmic foundations of AI, know how to effectively engineer commands (prompting), and understand the evolutionary limitations of systems to efficiently patch vulnerabilities in automatically created software.

2. Prehistory and the birth of AI

For a software engineer, examining the early history of artificial intelligence is a lesson in humility, demonstrating that the most visionary algorithmic concepts often have to wait decades for the hardware power to be realized in practice. Understanding why early algorithms relied on symbolic search reveals how software is inextricably linked to the physical constraints of the von Neumann architecture.

The ambition to create machines capable of thinking can be traced back to Greek mythology, but from a purely computer science perspective, the groundwork for these considerations was laid by the British mathematician and cryptographer Alan Turing. Just a decade after establishing the theoretical foundations of computer science (the so-called Turing Machine), in 1950 he published his groundbreaking essay "Computing Machinery and Intelligence." In it, he proposed a thought experiment that bypassed the complex philosophical debates about the nature of "consciousness" itself. Instead, Turing posed an engineering question: could a machine communicate in a way that deceived humans? The experiment, which he called "The Imitation Game" and is now commonly known as the Turing Test, involves placing a human judge in front of a text terminal. The judge conducts a conversation with two hidden entities—a machine and a human. If, based on logic and fluency alone, the judge cannot determine which interlocutor is software, the machine is deemed intelligent. Although the Turing Test defined the research horizon for over six decades, it has lost its relevance in modern times. Modern large language models (LLMs) can easily imitate humans, despite lacking consciousness or a genuine understanding of the world. This clearly demonstrates that perfect imitation is algorithmically easier to achieve than generating general artificial rationality.

The official birth of artificial intelligence as a distinct, formalized branch of science is considered to be 1956. It was then, during a summer research workshop at Dartmouth College in the United States, that the most prominent visionaries of the time, including John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon, gathered. John McCarthy officially used the term

"artificial intelligence" for the first time. Optimism that summer was enormous; it was widely assumed that a group of eminent mathematicians would program fully intelligent machines in a matter of weeks or years, relying solely on the creation of clever, logical algorithms on early digital computers.

Direct evidence fueling this optimism was the remarkable success of the first advanced computer programs. In 1955–56, Allen Newell and Herbert Simon constructed the program Logic Theorist. Unlike earlier computing machines, such as ENIAC, which were solely concerned with solving arithmetic operations for artillery, Logic Theorist's task was purely symbolic. It was designed to automatically prove complex mathematical theorems from Bertrand Russell and Alfred North Whitehead's famous *Principia Mathematica* . The algorithm relied on rigorous sets of predefined axioms and implemented complex tree searches using mechanisms such as forward chaining and backward chaining. The software's success was overwhelming. The program proved 38 different logical theorems from the book, and in one case, it found a mathematical proof that was shorter and much more elegant than the book's authors themselves had conceived. This was an unprecedented moment in which programming code was proven capable of simulating human logic within a closed, axiomatic system.

The next milestone was reached between 1964 and 1967 at MIT, where Joseph Weizenbaum wrote an early natural language processing program called ELIZA. This was the first chatbot in history that, unlike Newell's software, didn't solve mathematical problems but rather simulated conversation. Written in MAD-SLIP, the program supported various behavioral scripts, the most famous of which was called DOCTOR, which pretended to be a human psychotherapist in the Rogerian style. In reality, ELIZA was a masterful programming deception. It had absolutely no word understanding module; its architecture relied entirely on pattern matching and rigid parsing. The program searched the terminal for keywords typed by the user and applied simple substitution rules. When a user typed "I feel sad," ELIZA's parser would grab the pattern "I feel X" and substitute it for a hardcoded answer: "Why do you feel X?" If the system didn't find a dictionary match, it would return a generic answer, such as "Tell me more about that." To its creator's surprise, this assistant caused a profound sociological shock. Users began confiding in the machine incredibly intimate details of their lives, utterly convinced that the software displayed a tremendous amount of human empathy. This phenomenon has become a permanent part of computer science vocabulary as the "ELIZA Effect."

The Golden Age of optimism, however, quickly clashed with the ruthless laws of physics and discrete mathematics. As software engineers began to transcend simple, closed-loop logic environments and sought to program algorithms to address real-world problems, they encountered the so-called combinatorial explosion. In the real world, the number of possible decision states grows faster than exponential, while hardware of the time had minimal RAM and pitifully slow processors. In 1973, the devastating Lighthill Report was published in the UK, brutally concluding that the analytical techniques discovered in artificial intelligence were incapable of scaling beyond the safety of laboratory conditions, heralding the complete failure of earlier promises. Disappointed government and private funders cut off funding for the laboratories, ushering in a decade-long period of market stagnation, colloquially known as the First AI Winter.

3. The era of expert and rule-based systems

For a young developer, familiarizing themselves with the concept of expert systems is crucial, as it provides an excellent case study of so-called rule-based systems. This allows them to understand where the optimal utility of classic if-else coding ends and where the implementation of self-learning mathematical networks becomes necessary in software architecture.

In the 1980s, the engineering community learned a painful lesson from the funding freeze. The

dream of building a "machine that thinks like a human" was abandoned, redirecting 100% of its resources to specialized commercial tools. The category of expert system (Expert System) was born—a complex computer program with a single purpose: to emulate the decision-making competence of a human expert, but only in a very narrow field, such as materials engineering or taxation.

The secret to the power and subsequent downfall of expert systems lay in their architecture. Every such system was based on a drastic separation of data from logic, dividing itself into two foundations:

1. **Knowledge Base** : This was a kind of passive, massive memory container. It contained thousands of facts, laws of physics, domain relationships, and heuristics previously extracted through months of interviews with experts. This knowledge was manually recorded by programmers in the form of strict logical conditions, primarily in the form of instructions: "IF condition A and condition B are met, THEN perform action C."
2. **Inference Engine** : This was the processing module. As the user entered their current data (e.g., chemical parameters of a sample), the inference engine searched the structured knowledge base, dynamically selecting chains of rules step by step to reach a final logical conclusion. The advantage of this separation was that the programmer could update the rules in the knowledge base without breaking the engine's logic loop itself.

A prime pioneering example of this architecture in the 1960s was DENDRAL. Designed by Stanford researchers (including geneticist Joshua Lederberg and programmer Edward Feigenbaum), this powerful analytical system was used in chemistry laboratories to analyze mass spectrometry results. The program could deduce and represent the structure of organic molecules with remarkable accuracy based on input spectral parameters. In this extremely narrow niche, the software achieved results that far surpassed those of novice researchers.

However, the system that has made a lasting impact on computer science education is considered to be MYCIN, created in the early 1970s, also at Stanford University. While DENDRAL analyzed chemical molecules, MYCIN was the first medical advisor for physicians. Its primary operational goal was the rapid diagnosis of dangerous bacterial bloodstream infections (e.g., meningitis) and the prescription of specific, weight-adjusted doses of very rare antibiotics. Its entire knowledge base consisted of several hundred rules structured in an expert system. A typical logical schema (IF-THEN syntax) in MYCIN code looked like this: "IF the Gram stain is negative AND the organism's morphology is rod-shaped AND the patient has a fever, THEN there is evidence (with a specified confidence factor, e.g., 0.8) that the infection class is enteric."

Technologically and statistically, the program was a stunning success. In blind tests conducted using historical documentation, MYCIN was able to predict the correct therapies more accurately than trained infectious disease specialists, not to mention interns. Paradoxically, however, despite its proven computational superiority, the system never made it to hospital wards. The barrier wasn't imperfect code or RAM errors. The problem turned out to be profound legal and ethical issues—who was responsible for a patient's death caused by an error in a mathematical algorithm? Doctors at the time had too little trust in computers to entrust them with human life in a system that refused to be flexible.

From a software architecture perspective, rule-based systems quickly hit a technological wall. Their primary limitation was their absolute inability to learn or self-optimize. Engineers had to hard-code all knowledge by hand, line by line. Whenever a law changed in a given business sector or a new disease emerged without a corresponding rule, the system would crash spectacularly (the so-called brittleness problem). The costs of hiring programmers and experts to constantly edit and maintain knowledge bases became enormous. This phenomenon, combined with the mass failure of commercial companies selling such "Lisp" machines, triggered another crisis of confidence in the early 1990s. The Second AI Winter had begun. The final realization was that attempting to manually capture the complexity of the entire world and

human experience using IF-THEN statements was an architectural absurdity.

4. The Big Data Revolution and Machine Learning

In the traditional programming paradigm, an engineer writes logical code (rules), provides it with input data, and a processor generates the final output based on it. Entering the world of machine learning (ML) means a complete reversal of this logic for programmers. Here, a human feeds the machine with both data and past correct answers, forcing a specialized statistical model to independently "discover" the hidden rules within them to predict future outcomes. Machine learning has defined the transition from predetermined commands to adaptive probability matrices.

Artificial intelligence's emergence from long stagnation and onto the highway of exponential growth was fueled by the collision of two external revolutions, over which algorithm developers had no direct control. The first was the global Big Data revolution. With the advent of widespread broadband internet, social media, and Internet of Things (IoT) devices, the global digital landscape began to produce unimaginable amounts of unstructured images, text, and video. Without this free, diverse pool of digital knowledge, machine models would have no fuel for training.

The second, much more important catalyst for the world of hardware engineering was innovation in the silicon architecture of graphics chips, commonly known as GPUs (Graphics Processing Units). Standard, powerful server CPUs are designed for the incredibly complex, linear solution of single instructions (sequentially, one after another). For decades, GPU hardware was used solely to calculate polygons in early 3D computer games, but it was of a completely different nature—these systems were composed of thousands of incredibly simple, miniaturized computing cores capable of running simultaneously. Neural network researchers realized that training a model with thousands of artificial neurons required not complex algebra but trillions of incredibly simple decimal multiplications. Offloading these calculations to highly parallel GPUs accelerated the model training process thousands of times, making the subfield known as deep learning not only interesting but also profitable and scalable for businesses.

September 30, 2012, proved to be a pivotal day for the era of deep learning. The finals of the massive academic ImageNet Challenge (ILSVRC) were held. Engineers from around the world were tasked with building a proprietary algorithm that would classify hundreds of thousands of digital photos into 1,000 different and complex visual categories (e.g., different dog species) with the highest possible accuracy. Programmers, led by Geoffrey Hinton of the University of Toronto, among others, presented a convolutional neural network (CNN) called AlexNet. This software crushed competitors using classical analytical systems, reducing the system's reading error by a colossal 9.8 percentage points compared to the second-place finisher. Importantly, the secret to AlexNet's success wasn't just a theoretical shift; it was a masterpiece of early AI engineering. This model cleverly divided the training load of the deep network model simultaneously between two powerful NVIDIA GTX 580 desktop graphics cards. AlexNet also implemented an activation function called ReLU, solving the historically powerful "vanishing gradient" problem that previously prevented the training of very deep networks composed of many complex layers. The developers presented a system in which the first layer of neurons learned to recognize boundary lines, the next learned to recognize textures and sharp shapes, while the last generated a logical final prediction (e.g., "this is a Siberian Husky"). A boom in artificial, multi-layer networks began. It was realized that with millions of sample images and powerful multi-cluster NVIDIA graphics units, no human-written structured code could overcome the mathematics created by deep learning alone.

Another major shakeup hit the industry in 2016. The ancient, abstract, and incredibly difficult board game Go was long considered the "holy grail" of artificial intelligence researchers. Due to

its unique rules, the board allows for a gigantic number of simultaneous move combinations—so vast that it exceeds the number of observable atoms in the universe. This meant that the "brute force" principle (the forceful, massive computation of the entire tree of every possible option from A to Z), which had secured the Deep Blue chess model's triumph over the great Garry Kasparov in 1997, would be rendered completely useless here. However, the engineering firm DeepMind created the AlphaGo model. The system was built based on a combination of classical probabilistic tree search (Monte Carlo) combined in parallel with large deep neural networks using reinforcement learning (RML), meaning the machine trained itself by playing Go millions of times.
AlphaGo faced the champion of humanity, Lee Sedol, in a grueling five-game duel in Seoul, ultimately defeating him 4-1. This tournament is remembered not for the computing machine itself outclassing the biological brain, but for two iconic moves on the board, symbolizing the boundaries of both civilizations. In Game 2, AlphaGo placed a stone in a maneuver called "Move 37." Commentators initially considered this a mistake, as no human expert would have played such an illogical move at this stage of the game. Only after dozens of moves did it become clear that the machine had calculated and designed long-term and remarkably intelligent control over the central axis of the board's territory. For the first time, the machine demonstrated not cold calculation but a flash of profound "creative intuition" in predicting abstract territory dozens of turns in advance. Nevertheless, Game 4 saw a human breakthrough. Disoriented by the algorithm's genius, Sedol made a desperate "Move 78," now affectionately known as the "God Touch." This unconventional and so rare a statistical error that the powerful AlphaGo model completely failed to cope with it, confounding its predictive logic to the point of spectacular failure, demonstrating the flexibility of the unstructured human mind. The historical axis of AI had crossed the Rubicon.

5. The era of generative models - GenAI

A proper understanding of the mathematical essence and topology of generative AI (GenAI) models currently separates the average developer ("code whiz") from the expert capable of generating a system, testing it, and preventing the risky phenomenon of so-called machine hallucination. This is key to the entire engineering landscape. All tools currently in use at the turn of 2025/2024, including Copilot, Cursor, and hundreds of analytical libraries, owe their unique properties to a single event and architectural change.
The milestone of the modern era, giving rise to the GenAI phenomenon, was the publication in 2017 of a groundbreaking scientific paper titled *"Attention Is All You Need* ," published by a team of researchers affiliated with Google Brain. Until 2017, engineers wishing to use machines for tasks related to the smooth generation and modeling of text and conversations had to rely on so-called recurrent neural networks (RNNs or LSTMs). The fundamental and insoluble problem of this approach was its "sequentiality." A machine based on RNN logic "read" a cluster of words slowly, letter by letter, word by word (e.g., by analyzing a large snippet of code). Unfortunately, when reaching the final lines of the system's thousandth paragraph, the engine almost literally lost statistical contextual memory of the words and logic contained in the initial parts of the file, causing glaring problems in inference, while at the same time digesting resources extremely slowly due to the difficulty of processing on the GPU.
This groundbreaking paper completely eliminated sequential processes and relegated them to the engineering archive. In its place, a revolutionary architecture was proposed called *Transformer* , with an absolutely revolutionary mathematical core called the "Self-Attention Mechanism." Conceptually, the Transformer architecture stopped looking at the structure of a single word sequentially. The network analyzed each piece of data relative to every other piece of data simultaneously, regardless of the distance or magnitude of the separation within the

document.
From a purely programming perspective, this concept can be illustrated by vector matrices of large mathematical values represented as the famous QKV triangle, i.e.:

- **Query (Q)** : This is a representation of what a given "token" is trying to find out about others around it.
- **Key (K)** : This is a set of features that inform the rest of the words in the system "this is exactly what I contain".
- **Value (Value, V)** : The true weight vector of a particular concept passed in the final process.

An analogy from the database world would be a process similar to fuzzy search: Query is a user-entered request from the client browser, Key is an indexed, mathematically hidden entity within the relational search engine itself, and Value is the actual extracted line of logic. The key difference is that the Transformer system performs this operation for absolutely every word and element of the sentence "in the same split second," assigning each a unique, fractional measure of connection and probability. Thanks to QKV's powerful architectural structure, a sentence like "The zipper on my old jeans got stuck, causing the mechanism to stop holding the bolt" became 100% indifferent to the double meaning of the term lock (building/zipper). This was due to the system's high-value association of the Key with the word "jeans," extracting the highest and most reasonable Value without error. Moreover, Transformer's matrix mathematics was perfectly suited to optimization and vectorization on thousands of parallel graphics processing units (GPUs) in server rooms, sparking a sudden renaissance of powerful, nearly limitless scaling. As of 2025, this brilliant 10-page paper had achieved mythical status in the engineering world, with some 173,000 citations, fueling a global AI boom.
The principle of scalability has underpinned large corporations like OpenAI, which have been developing so-called Large Language Models (LLMs) based on the Transformer. Between 2018 and 2020, OpenAI demonstrated with a series of Generative Pre-trained Transformer (GPT) models that the intelligence and depth of virtual intelligence increase exponentially with the mathematical parameters fed to the model before the actual training. The GPT-1 and GPT-2 models were promising, but it was the giant GPT-3 released with over 175 billion learning parameters that caused a stir. The sheer volume employed here revealed a completely new phenomenon of emergent intelligence in the form of Few-Shot Learning. Based on a single command window and a few programming examples written directly by the user (without the need for costly, in-depth training, especially fine-tuning of programming code), the machine was able to deduce completely abstract phenomena on the fly. After a drastic expansion of the database, including the addition of multimodal graphics understanding via GPT-4, a now legendary series of free and premium applications was born, led by ChatGPT.
At the turn of 2025 and 2024, the market has evolved beyond the dominance of a single corporation, giving developers a vast array of powerful API models:

- **Claude (Anthropic)** : Known primarily for its Claude 3.5 and 4.5 Sonnet releases, this family has completely dominated benchmarks that rigorously evaluate pure programming logic based on front-end development using modern frameworks. The tool excels at navigating analytical tasks without straying from the guidelines.
- **Gemini (Google)** : A system with a powerful and incredibly deep Context Window. Gemini 1.5 Pro and higher versions can swallow files weighing thousands of pages or dozens of vast code trees into the input window at once, and then ensure architectural consistency without "forgetting" individual variables and functions buried 80 subdirectories deep in the project.
- **Mistral Large 2 and LLaMA (Meta)** : Revolutionizing the realm of open source licenses and the phenomenon known as Mixture-of-Experts (MoE). The system activates only a precisely dedicated small slice of the entire giant's parameters (e.g., 41 billion of the global 675 billion parameters in the case of Mistral) in a given conversation frame and

attention vector. This drastically reduces the massive server API cost required for operation and engineering support, while maintaining the agile flexibility and accuracy of much larger and more avid competitors.

Deep transformer architectures also encompass image models based on a completely different technology called Diffusion Models, commercially supported by applications like DALL-E and the popular Midjourney program. Imagine a chef able to prepare a dish and then just as flawlessly reduce burnt cake back to fresh, unbaked flour, eggs, and sugar. Diffusion models are powerful, two-stage networks based on the mathematics of thermodynamics and the Gaussian distribution. In the first powerful training step, the algorithms destroy the image pixel by pixel, generating visual noise until they produce a "random soup of graphic noise" called Gaussian noise (a process called Forward Diffusion). Then, a development network (e.g., a predictive architecture called U-Net) is forced, using a sophisticated loss minimization algorithm, over millions of long cycles to perfectly predict the removal of noise backward (the Reverse Diffusion or Denoising process) in relation to and conditioning on a word-based text vector based on the system's matrices (Cross-Attention). This results in a powerful, nearly limitless capability and the ability to "conjure up from magic noise and particles" structures of hyperrealistic portraits completely unseen in the network simply by eliciting a text request from the user, for example, through the query "astronaut on coral."

For young programmers, 2025/2024 will ultimately be a clash of powers creating programming environments entirely based on AI models. Tools have revolutionized so-called Integrated Development Environments (IDEs), and programmers have moved from the "autocomplete" stage to a powerful and dynamic agent:

- **GitHub Copilot (Microsoft)** : The enterprise standard and most established system built by a market leader, integrated with a repository environment. Embedded directly into environments like Visual Studio, Copilot predicts large blocks of boilerplate code, solving top-down, cumbersome problems without lifting a finger.
- **Cursor AI** : Currently the king of so-called advanced editors, technically based on the VS Code engine. Its advantage is a dedicated engineering interface based on a powerful assistant capable of making dynamic software changes simultaneously to dozens of files extracted by the system, allowing advanced programmers to issue a single command via CMD+K to create a coherent, logical service with unit tests ready in the terminal in the background.
- **Windsurf** : A heavily subsidized developer favorite with a growing market and open commercial plans under the Codeium umbrella. It's fantastic for projects where the developer expects an agent to maintain a consistent, robust vector of powerful memory for hundreds of previous chat events; it provides exceptional step-by-step feedback to the operator from the machine's flow of actions to the repository database, thus "never losing track" of the engineering design and planning of the file system structure.

The transformation has forced and cemented the role of the "AI-Assisted Developer." Failure to utilize machines by companies and developers in today's world is tantamount to immediate expulsion from the market due to the disproportionate productivity dissonance. Your education lies precisely in understanding this timeline, and consequently, in understanding the dangers of blindly allowing a machine to dictate architecture and losing a healthy, critical horizon in inference when the model, due to its lack of token weight, undoubtedly delivers a flawed and potentially tragic oracle.

6. Timeline - AI Milestones

The table below provides a concise summary of the most important milestones in AI research, along with a concise breakdown of their technological significance. It serves as an excellent

reference for chronology.

Year	Event and Innovator	Importance for engineering artificial intelligence systems and algorithms
1950	Publication of the essay *Computing Machinery and Intelligence* by Alan Turing	Formal suggestion of the "Imitation Game" (Turing Test) as a rational benchmark for evaluation in artificial communication.
1955	Successful Testing of *Logic Theorist Deductive Program* (Newell, Simon)	The first clear proof in code of the principle of operation of a symbolic machine demonstrating the correct process of finding axioms from mathematical works without using a processor as a calculator of continuous phenomena.
1956	A series of summer research conferences at Dartmouth laboratories	Coining the name of the discipline of "Artificial Intelligence" and informally sparking a market-driven search for the "Holy Grail" of software engineering.
1965	A system based on strict *DENDRAL rules*	The triumph of separating the knowledge layer and the deductive engine interfaces in the study of the spectra of chemical molecule distributions in spectrometry.
1966	*ELIZA* system imitating the functions of a psychotherapist at MIT	A powerful social shock that demonstrates how blind pattern-finding tricks can create a profound false bonding effect on the unwary.
1972	The emergence *of MYCIN* in the diagnostic sphere of bacteria in the blood	Establishing a pattern for writing "If-Then" conditional software in closed-loop medicine, but facing a world of ethical barriers that prevent implementation in hospital code.
1973	Sir Lighthill's Printed Academic Report	A brutal deconstruction of programmers' dreams, supported by mathematical physics, revealing the impossibility of entering the powerful, complex real world, leading to the phenomenon of cold budget cuts for decades

Year	Event and Innovator	Importance for engineering artificial intelligence systems and algorithms
		(the First AI Winter).
1997	*Deep Blue* program and server wins the sharp chess competition (against Kasparov)	The triumph of the classical, algorithmic, and pure force paradigm with highly computationally loaded computer hardware over the structured tactics of a human grandmaster.
2012	*AlexNet* model dominates ImageNet analytical challenge with powerful efficiency	Awakening the discipline of neural network phenomena. Convolutional triumph with parallel support for two early NVIDIA GPU cores in a revolutionary and scalable image processing environment.
2016	*AlphaGo* statistical game triumphs over board game great Lee Sedol	Unprecedented use of neural networks at the level of abstraction and unpredictability (Move 37 vs. human untamed Move 78), opening the eyes of the world to the deep intuitive powerful phenomenon of ML.
2017	Publication of the groundbreaking article *Attention Is All You Need* (Google Brain)	The complete destruction of networks with recursive logic through the phenomenon of architecture linked as "Transformer" based on lightning-fast parallelism of attention.
2020	Massive commercial test of a 175 billion parameter environment with a GPT-3 model network	Defining the fundamental concepts of flexible analytics and intelligent conceptual prediction, referred to in software as a powerful Large Language Model (LLM), for broader access to APIs for engineers and students.
2022	Widespread Entry Window Opening with the *ChatGPT Phenomenon*	The eruption and complete takeover of mainstream media, developers, and processes in the so-called GenAI (generative) era to create infinite, fluid new textual value at an incredible rate of iteration and universal learning across billions of access points.
2024	Fierce battle of multimodal	The market is breaking away

Year	Event and Innovator	Importance for engineering artificial intelligence systems and algorithms
	analytical assistants (Anthropic, Meta, Mistral AI)	from a powerful monopolist to implement and develop open variants of giants that can handle tens of thousands of large text files in the form of a single token call.
2025/2024	Platform tools as development agents (GitHub Copilot, Windsurf engineering, Cursor IDE)	46% of code written in the industry is becoming proprietary and dependent on AI autocomplete. This de facto transformation formally positions the "AI-Assisted Developer" standard in the hierarchy of corporate and business roles and offerings.

7. Practical exercises

As a future software coding engineer, your absolute priority is to immediately empirically validate the architectural and statistical concepts described in technical documentation within a trusted, independent production environment. Solve the following questions, basing your analyses on the fundamental goal: understanding the difference between "primitive hard coding" and "probabilistic estimation of matrix model weights."

- **Exercise 1: AI Timeline and Building a Naive Rule-Based System in Jupyter.** Run a local Jupyter Notebook server process or log in to a free Google Colab compiler address using the latest Python 3 kernel. *Step 1:* Define three neat Markdown cells operating under strong headings, explaining definitively and logically what, in your engineering and logical view, separated the naivety of the first Winter Age of Algorithmic AI from the Golden Boom of Deep Neural Networks based on early multicore hardware in ImageNet research (refer to GPU vectors for this description). *Step 2:* Build a primitive and simple implementation in clean and concise Python of a program that constitutes a non-fluent parser of a classic "chatbot," ideologically drawing on the powerful script of old ELIZA. Use only structured dictionaries with keys and hard-coded characters, using a powerful library of regex expressions. Ensure that the input response "I feel scared about my job in programming" returns a conditional string with dynamic variable overlays in string f-formatting, generating a rigid "Why do you feel scared about your job in programming?" *Step 3:* Complete the task and work with a long, multi-sentence target docblock summarizing why, from a software perspective, the regex-based syntactic pattern matching used before 1970 has nothing in common with the statistical concept of deep vector semantic association (Deep Learning), despite the outward appearance of the logic to a bewildered human in chat. The assessment will be based on the sharpness of distinguishing semantics from the mechanical syntax of the system code.
- **Exercise 2: Context Engineering - Advanced Analysis with Benchmarking** Your new, powerful AI-Assisted assistant isn't a debugger with hard memory dumps, but rather an awareness of LLM vector logic from competing market players (e.g., a closed-source browser chat engine based on OpenAI models vs. Cursor software supported by Anthropic models, or vice versa, within the freely available API query limit from Codeium

on Windsurf). *Step 1:* Paste completely flawed and architecturally complex asynchronous Python code (asyncio with a deadlock, impossible to find due to being a silent bug that doesn't trigger a classic stack trace). *Step 2:* Design, precisely formulate, and implement a so-called advanced operational prompt that forces and specifies that both assistants think in slow, multi-layered snapshots about the code logic, a so-called "step-by-step" system with an explanation of the business programmer's preconceptions. *Step 3:* Create and prepare in your notes a structured, multi-column table based on the system's formatting logic with Markdown describing how Model A maintained consistency in response to the window conditions of the command, and at what point the other competitor induced absolutely artificial "machine hallucinations" by creating class packages on the fly or ignoring the key argument of thread variable locking. Finally, write down a single, decisive sentence assessing the ability to maintain the entire conversational vector before "losing a rigid thread after the system's powerful attention table was truncated due to lack of memory."

- **Exercise 3: Developer's Creative and Critical Technical Essay.** Write a highly structured, deeply authored technical essay report, based on a print-formatted (.md) file. The paper, with a maximum sharpness and conciseness of approximately 300 rigorously proofread words, must accurately address the technical question posed at the academic introduction: *"Which of the architecturally or hardware-wise powerful technological paradigmatic leaps (from hard-coded learning, model-based networks, QKV Transformers, to powerful, scalable assistive machines like Cursor and Copilot) in the vast history of artificial intelligence has, and will, from an engineering perspective, trigger the greatest decisive and objective breakthrough in your final target engineering environment? What rationale will you use to logically defend this concise conclusion?"* Maximum points are awarded to conclusions that reference the "empty conditioning of the knowledge base" as opposed to the sensational simultaneity of the matrix mechanism of powerful multi-process support. Disclaimer: Before being submitted for audit evaluation, text generated from the assistant software must receive comprehensive and verifiable insight into the critical analysis of the author's model distortions (to avoid naively copying logic from the structures of the spat-out chatbot by a programmer avoiding learning reliable code and engineering thinking when learning a new concept).

8. Glossary of terms

- **Artificial Intelligence** (AI) is a broad field of computer science research focused on programming digital operations and systems that mimic typical behaviors, rational reasoning, and complex, complex actions that were previously considered the exclusive domain of humans. In programming, this represents a shift from the rigid control of processor calculations to the flexible modeling of multi-operational goals based on sets of predictions.
- **Learning** (ML) — A groundbreaking operational class within AI itself that radically shifts away from coding "IF-THEN" statements to solve a desired problem, allowing programs to learn the true meaning of hidden probabilities from a generated experience base. It's mathematical learning based on statistical testing rather than statically imposing strict engineering paths in the initial application code.
- **Learning** — A branching system from ML environments, dominant in modern hardware, powered entirely by extremely extensive, layered layers of matrix-related decision-making structures, i.e. so-called multi-layer networks operating on the processes of recognition and abstract generalization, e.g. graphics and multi-level visual signals, without the preliminary and tiring processes of extracting parameters of manual features of the object

model for the engineer.

- **Neural Network** — A computational and numerical mathematical powerhouse that is the foundation of Deep Learning, drawing its original engineering inspiration from the topography of neural cell layouts in the biological brain. A collection of optimized nodes possessing their own nonlinear activation phenomena, optimizing entire algorithms after a drastic error through an extremely advanced multi-part, multi-vector, inverse negative backpropagation algorithm.
- **Expert System** - A rigidly architectural concept, dominant until the 1990s, of extremely narrow and focused logically closed decision intelligence, built by separating a powerful passive encyclopedic Manual Deductive Knowledge Base imposed from above from meetings, connected to an external very logically fast module called the Inference Distribution Engine (like the old MYCIN program diagnosing doses for rare bacteria from tests in the blood of test patients).
- **Big Data** - An explosion with huge powerful resources of historical image file vectors, text volumes and hardware sensor measurement logs in the world of the Internet with an unprecedented diversity, and especially the huge disk volume after the year 2000, which is the final necessary training fuel to feed the empty statistical probabilistic scale on deep multi-level stratifications in the era of disruption after 2012 on cloud systems under the threat of gigantic supplies.
- **GPU** (Graphics Processing Unit) - Once a purely specialized processor for rendering digital polygons, and after the 2012 revolution, an irreplaceable multithreaded coprocessor with enormous power of miniaturized hundreds of powerful asymmetric and arithmetic units counting thousands of point and trivial instructions simultaneously; which turned out to be the only solution to overcome the limited sequential pipeline of a powerful central computer system from the side of the classic processor architecture of a PC machine.
- **Generative Model** - An extremely developed conceptual and programming set of tools that use learned probabilities not for the classical purpose of division or correct selection of object-oriented generalization in selecting a parameter from file noise for decision-making, such as a banking algorithm, but for operations and mechanisms with the intention of ultimately massive, absolute, creative and continuous powerful creation of operational code for the target client or generating a hyper-realistic image of the file, distinguishes and provides unpredictable data.
- **Transformer** — The cult object of the era and the gigantic feat of the team from the scientific article Attention Is All You Need finally breaking the barriers of the algorithm-blocking slow and cumbersomely forgetful process of reading single characters on loops with a fraction of recursive memory to the triumphant release of processing to the simultaneous massive imposition of a grid of scoring probabilities called weights by a bilateral approach to the concept of multidirectional attention conceptually expanded as QKV.
- **LLM** (Large Language Model) - A gigantic operational service within structured server instances using a top-down, extremely optimized design based on the transformer topology phenomenon across hundreds of billions of fractional nodes to almost 100% accurately predict the locations of appropriate logical query strings for program syntax in a seamless and communication-free manner for "supported programmer engineering from GPT to LLaMĘ with Meta on API input".
- **AI Winter** - A freeze with a huge impact on the university, repeated critical discouragement of large subsidizing capitals deciding to completely and ruthlessly cut off the financing of programming laboratories and algorithmics right after the realization of the merciless burst of the gigantic wave of optimism inflated after years of powerfully illusory marketing on investment markets for the promised but ultimately impossible, due to the

lack of physics, computational calls on servers from old RAM systems with rigorous hard structures in Lisp languages.

- **Token** - An analytical measurement metric at the lowest conditional vector operational level when pouring a raw digital block of a command, word-like, often constituting a prefix and a concatenation of logic with a hyphen, transformed to build an absolutely rigorous process of capturing a machine digital zero into a virtual multi-level input system of a modeling algorithm, truncated to impose upper limits on the paid capacity of the query session server (like the gigantic limited window tokens of Claude memory or environments in the Copilot editor for a subscription fee for consumed session volumes on programmatic parsing phenomena).
- **Prompt** - A command layer; an engineering or linguistically rigorous set of boundary conditions for commands used and controlled in an input open request with the intent of controlling the implicit LLM generative call system to extract or block unwanted forms of appropriate styles in the developer window during the conversational callback process, e.g., forcing the removal of empty redundant code libraries before block and target file dump from the machine, including verification and assertion of programmatic code safety to prevent blind architectural confabulation in the IDE after being made aware of the potential rigors of the business language.
- **Pre-training** - Absolute and massive in terms of the gigantic investment from the perspective of the graphics card power, the first of the long-term, unfiltered period directly within the framework of the search engine logic, loading into the machine an unimaginably empty thicket of large old historical structures from an open library on the Internet, making the foundations of the algorithm only how to correctly generalize human logic without any specialization for narrow calling goals or a nested terminal agent.
- **Fine-tuning** (Domain Tuning) - A precise, theoretically small-scale developer's craft, designed to apply to a large, learned LLM a handful of several thousandths of the perfect, proprietary behavioral logics developed by security specialists to prevent the error of "eating the logic on the client's window from the company's API knowledge in the local environment or forcing the server to write back in a supported language strictly, e.g., without comments and docstring on the fly, before pushing the commit to the powerful distributed work environment of the target group" without interfering with the global mechanism and the central matrix of the transformer, but only overwriting a lightweight dedicated profile and adjusting the weight of the final powerful conditional reasoning processes of the developer's behavior model for the assistant client on a dedicated branch, locked behind paid logic security.

Works Cited

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. Why AI Is A Massive Job-Creation Technology, Despite What You Think, https://joshbersin.com/2024/03/why-ai-is-a-massive-job-creation-technology-despite-what-you-think/ 5. GitHub Copilot Statistics And User Trends In 2024 - Companies History, https://www.companieshistory.com/github-copilot-statistics/ 6. GitHub Copilot Statistics 2024 - Users, Revenue & Adoption - Panto AI, https://www.getpanto.ai/blog/github-copilot-statistics 7. GitHub Copilot Statistics 2024 - Quantumrun, https://www.quantumrun.com/consulting/github-copilot-statistics/ 8. 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 9. AI | 2025 Stack Overflow Developer Survey,
https://survey.stackoverflow.co/2025/ai 10. Github Copilot Usage Data Statistics For 2024,
https://www.wearetenet.com/blog/github-copilot-usage-data-statistics 11. Generative artificial intelligence and the Polish labor market - 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 13. The History of Artificial Intelligence | IBM, https://www.ibm.com/think/topics/history-of-artificial-intelligence 14. What is the history of artificial intelligence (AI)? - Tableau,
https://www.tableau.com/data-insights/ai/history 15. Attention Is All You Need - Wikipedia,
https://en.wikipedia.org/wiki/Attention_Is_All_You_Need 16. Logic Theorist – Knowledge and References - Taylor & Francis,
https://taylorandfrancis.com/knowledge/Engineering_and_technology/Artificial_intelligence/Logic_Theorist/ 17. Logic Theorist: The program that reverse the foundations of mathematics - Big Think, https://bigthink.com/books/logic-theorist/ 18. Principia Mathematica - Wikipedia,
https://en.wikipedia.org/wiki/Principia_Mathematica 19. 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-8274ea56247e 20. 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 22. 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/ 23.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/ 24. Expert Systems: What are they, how do they work and where are they used? - PW,
https://pawelwoloszyn.pl/slownik/systemy-ekspertowe-co-to-jest/ 25. Expert System In Artificial Intelligence - Scaler Topics,
https://www.scaler.com/topics/artificial-intelligence-tutorial/expert-system-in-ai/ 24. Expert system in Artificial Intelligence - OER Commons,
https://oercommons.org/courseware/lesson/130473/student/?section=2 27. Rule-Based System in AI - GeeksforGeeks,
https://www.geeksforgeeks.org/artificial-intelligence/rule-based-system-in-ai/ 28. Expert System in AI - AlmaBetter,
https://www.almabetter.com/bytes/tutorials/artificial-intelligence/expert-system-in-ai 29. 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 30. AlexNet and ImageNet Explained - YouTube,
https://www.youtube.com/watch?v=c_u4AHNjOpk 31. Understanding AlexNet: The 2012 Breakthrough That Redefined AI - Medium,
https://medium.com/@igquinteroch/understanding-alexnet-the-2012-breakthrough-that-redefined-ai-d0e247e2470a 32. AlexNet and ImageNet: The Birth of Deep Learning - Pinecone,
https://www.pinecone.io/learn/series/image-search/imagenet/ 33. AlexNet: The First CNN to win Image Net - Great Learning,
https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/ 34. (PDF) ImageNet Classification with Deep Convolutional Neural Networks - ResearchGate,

https://www.researchgate.net/publication/319770183_Imagenet_classification_with_deep_convolutional_neural_networks 35. AlphaGo versus Lee Sedol - Wikipedia, https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol 36. The significance of move 37 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 37. Lee Sedol and AlphaGo: The Legacy of a Historic Fight! - Go Magic, https://gomagic.org/alphago-and-lee-sedol/ 38. 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/ 39. Attention is All you Need - NIPS, https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf 40. Transformers Explained Simply: From QKV to Multi-Head Magic - Medium, https://medium.com/@rajputshubham219/intuition-is-all-you-need-4920f6ad7b18 41. Attention is all you need (Transformer) - Model explanation (including math), Inference and Training - YouTube, https://www.youtube.com/watch?v=bCz4OMemCcA 42. 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 43. Attention is all you need explained - YouTube, https://www.youtube.com/watch?v=sznZ78HquPc 44. 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/ 46. 10+ Large Language Model Examples & Benchmark - AIMultiple, https://aimultiple.com/large-language-models-examples 47. Top 50+ Large Language Models (LLMs) in 2024 - Exploding Topics, https://explodingtopics.com/blog/list-of-llms 48. LLM Leaderboard - Comparison of over 100 AI models from OpenAI, Google, DeepSeek & others - Artificial Analysis, https://artificialanalysis.ai/leaderboards/models 49. AI Diffusion Models Explained (Midjourney, DALL-E, Runway, Sora): How AI Image Generation Works - YouTube, https://www.youtube.com/watch?v=VyNATiMJO_k 50. 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 51. 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 52. Cursor vs Windsurf vs GitHub Copilot - Builder.io, https://www.builder.io/blog/cursor-vs-windsurf-vs-github-copilot 53. Cursor vs. Windsurf vs. GitHub Copilot - Educative.io, https://www.educative.io/blog/cursor-vs-windsurf-vs-copilot 54. AI Coding Tools Compared (2024): Cursor vs Claude Code vs Copilot - Benchmarks & Pricing | TLDL, https://www.tldl.io/resources/ai-coding-tools-2024 55. 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 56. 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/ 57. Artificial Intelligence Timeline (1950–2025) – AI History and Milestones | AIny, https://ainy.no/en/artificial-intelligence-timeline-1950-2025/