The Evolution of AI and its Implications

Kyler Grahame
Department of Computer Science,
Montclair State University
Montclair, New Jersey
kjgbusiness8@gmail.com ,
grahamek1@montclair.edu

Abstract - This paper explores the rapidly evolving applications of Artificial Intelligence, (AI for short), in the field of Computer Science, as well as the practical programming side of real-world applications that AI brings to programmers and developers. We will explore the implications and transformation AI brings to the field from the scope of intelligent code generation to personalized user interfaces and automated testing. AI is streamlining production and deployment timelines for developers and enhancing user experiences for actors of applications, (Actors is another reference to users). The main topics we will dive into will be AI-powered development, the significance AI brings to development timelines and life cycles, and the broader implications it has for the future of the tech industry and user interactability. Finally, it will offer a personal reflection on the importance of human oversight, hybrid models, and ethical considerations in AI adoption.

I. Introduction

Artificial Intelligence (AI) has rapidly shifted over the last 20 years from a Sci-Fi Movie element and purely theoretical concept to a powerful driver in the exponential expansion of technology. In the modern-day era and tech stack of web development, cloud computing, user experience design, and automation. AI has become a necessary tool and expansion to the evolution of technology. The goal of this paper is to provide a layered comprehension to the reader about AI, its implications, and evolution. To that end, we will begin exploring the fundamentals of AI and machine learning and provide some insight into the inner workings of the models that drive its applications. As this is such a vast topic it will explore the fundamentals of AI, how it is classified in different manners, and its applications, but will leave some aspects to be further researched.

II. Understanding the Basics

First, we will dive into the classification of different types of Artificial Intelligence Models. Early iterations of AI applications were built on machine learning models. These models were reliant on human intervention to process new information and perform tasks they were not initially trained for. These models could provide a level of ‘perception’ but this was attuned to many developers providing constant evolution and groundwork to create this facade. For example, “Apple made Siri a feature of its iOS in 2011. This early version of Siri was trained to understand a set of highly specific statements and requests. Human intervention was required to expand Siri’s knowledge base and functionality.” [1]. Artificial Intelligence's big breakthrough came in 2012 when Neural Networks were developed.

A. Neural Networks

Neural Networks rely on machines that engage in success rates and reinforcement learning. This differs from the systems like

mappings for NLPs (Natural Language Processing) which would provide the ‘perception’ of awareness of conversations or inquiries. This allowed the removal of human intervention to a degree because, unlike traditional machine learning models, Neural Networks provided the possibility for models to learn how to perform new tasks or make decisions without human intervention. To further explain, Neural Networks process data through layers. These layers consist widely of three defined components, the input layer, the hidden layer, and the output layer. The input layer handles raw data and passes it to the hidden layer. The hidden layer consists of interconnected nodes (Representative of Neurons in a human brain), these nodes apply a weighted mathematical transformation on the incoming signals passed in from the input layer. They calculate a weighted sum of inputs and apply an activation function. (The Activation Function determines whether a neuron should be activated or not.) This is then passed to the output layer which produces the final result of the network. This could be a variety of things dependent on the model ranging from classifications, generated text, recognition, and predictions. This layered system mimics the human brain's neurons, to our current understanding, and effectively makes this model adept at pattern recognition in complex and highly dimensional data such as images, speech, natural language, etc…

B. Classification of AI

The classification of different types of AIs is disoriented across different sources but they often fall into three agreed upon categories with accompanied subcategories. The categories are as follows; AI based on capabilities, AI based on functionality, and AI based on Architecture.

1) AI-Based on Capabilities

There are three subcategories in the capabilities category. First is Artificial Narrow AI, often referred to as weak AI. This is the AI we use today in 2025, it is easily trained to perform a single or narrow-scoped task and is often more efficient than a human would be at completing this task. “However, it can’t perform outside of its defined task. Instead, it targets a single subset of cognitive abilities and advances in that spectrum. Siri, Amazon’s Alexa, and IBM Watson® are examples of Narrow AI. Even OpenAI’s ChatGPT is considered a form of Narrow AI because it’s limited to the single task of text-based chat.” [1]. The next type of capability AI is General AI, often referred to as Strong AI. This classification type is purely theoretical in the modern day. It would encompass a model that essentially can have full context awareness and complete any task required of it. This is where we step into the realm of sentience from machines and could even have emotional understanding, innovation, creativity, etc… be possible by the model. The last type of AI in capability classifications is Super AI. This is strictly theoretical and would encompass more than we as human beings can comprehend. It would be capable of thinking, reasoning, learning, passing judgments, and possessing cognitive abilities. “The applications possessing Super AI capabilities will have evolved beyond the point of understanding

human sentiments and experiences to feel emotions, have needs, and possess beliefs and desires of their own." [1].

2) AI-Based on Functionality

The next type of classification of AI is functionality. The first type is Reactive Machine AI. Reactive Machine AI is a system that has no memory and is designed with the intent of completing a task of narrow scope. Since they cannot collect or reference previous results or outcomes these models only work with the data they presently have available to them through input or analysis. These are often used in the analysis of vast amounts of statistical data or mathematical relations. The next type of functionality classification is Limited Memory AI. Limited Memory AI can recall past events or results and monitor changes in comparison to time. The limitation of Limited Memory AI is that it has a context window of comprehension, this means there is a limit to the data it can keep from past inquiries or events that are relevant to the current task it is attempting to compute. Limited Memory is used in tools like ChatGPT for generating text and can improve or worsen depending on the data it has access to and continuously trains on. The third type of AI in functionality is currently theoretical and is Theory of Mind AI. This AI would be more focused on inferring human emotions and motives. It would be able to understand and contextualize deep human social constructs such as emotions, innovation, inspiration, creativity, etc… The last type of AI in functionality is Self Aware AI, this is purely theoretical as well and can be classified as a type of low-level Super AI or Strong AI. Self Aware AI would essentially encompass the idea of sentience in Artificial Intelligence and could be attuned to saying the recreation of a Human being or another form of being that possesses a similar and equal level to our cognitive ability and comprehension of social constructs.

3) AI-Based on Architecture

Architecture models are based on the strategy a model uses in its systems. There are many different types of Models and this is the least identifiable model category as many have the potential to overlap. They can be classified into 6 subsections, Machine Learning Models, Deep Learning Models, Natural Language Processing (NLP) Models, Computer Vision Models, Generative AI Models, and Hybrid Models. To go into all the subtypes and overlap for all these models would be quite redundant and could be derived into its own paper. Instead, let's highlight a specific model in the Generative AI category that has driven the 'hype' of AI over the last few years. This model is the Transformer based Model. Transformer models made their rise in a 2017 paper from Google "Attention is All You Need" [2]. Transformers built upon the previous dominating model, encoder-decoder model structure, and built upon it further. "Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 29]. Here, the encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of

symbols one element at a time. At each step the model is auto-regressive [9], consuming the previously generated symbols as additional input when generating the next." [2]. To break down transformers more simply there are generally six steps. The first step is to break input down into tokens, either by letter or by word typically. The next step is to translate these tokens into vectors, this means that tokens become a numerical representation of each letter or word. For example, lowercase 'a' in the alphabet is represented by 1, and lowercase 'z' in the alphabet is represented by 26. The third step is the Self-Attention Mechanism. This simply means that it looks at and processes all the tokens at the same time and learns what words or letters have closer relations to each other. This is where the weighted sum comes into play and drives the power of context awareness for the model. The fourth step is Positional Encoding. Since the transformer reads everything in parallel, they also add information or metadata about word order using math, This is where weighted calculated sums come into play. The fifth step is the multi-layered system processes the data and their relations. It consists of multiple layers of self-attention and feed-forward networks to refine its comprehension. The final step is generating output based on everything it just processed. This could be a next-word prediction, text translation, answering queries, generating code, etc… This can scale to be more refined based on the amount of data it was trained on as well as the number of layers involved in the system.

III. Computer Science Implications (Theory Based)

The implications of AI in the field of Computer Science will mostly be in reference to students in Universities, Research Associates, and professors researching as this is the most relevant and theory-based section of Computer Science. To further build upon that the AI models we talk about here will be Generative AI, LLMs, and more specifically Transformer models, Encoder-Decoder Models, etc…

A. AI Debates in Education

The biggest debate in CS education and AI implies that AI poses a problem for human learning as it can just provide the answer for a student. Need a calculator app? Ask ChatGPT and complete it instantly. Now this is true to the extent that AI does pose a conflict for some students and the value they obtain from their education. But to inherently state that AI is posing a problem is incorrect. AI is a tool, a tool does not inherently inhibit the user's ability to struggle through a problem and learn. Can a student use AI to quickly piece together functional code and complete a project for them, yes of course. Now that student didn't inherently learn much from that assignment, but this is only relevant to projects of smaller scope. The more a student or any programmer increases the scope of a project and is reliant on AI the more redundant code and errors they will encounter as the project scales. This obstacle will force them to start learning two things. They will start to digest the information and strategies the AI uses in narrow scopes, which can be built upon to

slowly build a wider understanding of the concepts and practical use. The second thing they will learn is how to better leverage the tool of generative AI. So in this perspective, the programmer is still struggling through code as they take on more complex problems just with a personal assistant to answer all their queries. This is also an assumption that the user is straightforwardly querying the AI to complete tasks for them, which is the most negative outcome and over time the user will still learn concepts. Now think if a student uses AI to break things down into smaller components for them. They query it to explain the lines of code they don't understand or ask for references. This will do nothing but expedite the learning curve of CS education. If you take this perspective AI is less like a cheat button to automatically accomplish things and more like a magnet to attract the needle in the haystack to learn, code, complete projects, etc quickly… To finalize the debate on AI in CS education, "Working hard and struggling is actually an important way of learning. When you're given an answer, you're not struggling or learning. And when you get more of a complex problem, it's tedious to go back to the beginning of a large language model and troubleshoot it and integrate it."[3]. This perfectly summarizes the previous point stated in the paragraph that initially the students who use AI to quickly accomplish the tasks for projects of smaller scope will suffer, but as they advance and run into large-scale projects and issues start appearing left and right they will be forced to struggle through and learn the basics, as well as learning how to more efficiently leverage AI as a tool. So it is naive to say that AI in the long term will damage developers.

B. Challenges to Traditional CS Paradigms

One of the theory-based classes most CS majors will take at University will be CS Theory, or Theoretical Computations, etc… whatever name they call it. This is a class that fundamentally explores what is and isn't computable by learning about Turing Machines, deterministic/nondeterministic models, and modern computers. CS students learn in this class that computation must be logical to be computable and follows a step-by-step procedure, which is widely quite true for almost everything in the field from theory to deploying code, to how the layers of the web work, all the way to making a simple square render in Python. The traditional view we are taught is that Turing Machines (TMs) define the boundaries of what is computable. "Turing Machines (TM) play a crucial role in the Theory of Computation (TOC). They are abstract computational devices used to explore the limits of what can be computed. Turing Machines help prove that certain languages and problems have no algorithmic solution" [4]. So you learn in this course that all data and computations are traceable and verifiable in some manner. But then we encounter Neural Networks which often do not expose explicit rules for why a decision was made by the system. Even a small discrepancy (An Outlier) in a data set can skew the output quite drastically in an LLM for example. This would not occur in classical systems of computation. This is because before AI every software was

programmed and designed to do a task, whereas AI is trained to adapt its behavior based on patterns. This leads to the question is AI completing tasks? Is it computing answers? At first glance, one might think "Of course, it is, I asked ChatGPT to write me a poem about dogs and cats and it did." But this is not inherently true. Yes, AI does solve things for us, but does it necessarily compute things in the way traditional systems would? No, AI is outputting the most likely result, not computing data. This is where AI enters the realm of non-determinism. This means that output can vary from the same input unlike our classic TMs would compute something. This shift poses a major challenge when it comes to the paradigms of the CS curriculum, as all previous topics in computability were deterministic models that could be simulated, traced, predicted, and verifiable. Now we have this probabilistic model that encompasses all AI, and it often is lacking in transparency. This leads to AI often being described as black boxes, their internal decision-making processes are difficult to comprehend. This leads to AI being in direct opposition to the classical deterministic models we used in the past, you could input the same data into a Generative AI (Let's use OpenAI's ChatGPT as an example), and every single output will vary due to its randomness seed. This leads to AI models often being uninterpretable, as stated before, and this even applies to the people who created these models. This poses challenges to the field in the guise of the question of what is computable. We for so long focused on deterministic computation, now that this new model has appeared, we may need to redefine the boundaries of what we define as computable in the CS field of theory. The more AI and probabilistic systems grow and work their way into more subsets of modern computing, the more CS theory will be required to evolve to this new expansion. An expansion where answers are not always deterministic, traceable, or even explainable in a sense. This could be one of the most significant shifts in the industry — not just in how we build software or increase efficiency, but in how we fundamentally define what it means to compute in the field of Computer Science

IV. Programming Development Implications (Practical Based)

Now that we explored the theoretical implications AI brings to the Computer Science field, we can dive into some real examples of models and how they improve and hinder the developer world. We will avoid the debate of the AI-assisted code style being detrimental to developers as we discussed above and focus on the increased deployment of code, the redundant search for fixes, and how AI can be leveraged as an overall positive feature for developers in this section.

A. Intelligent Code Generation Models

First, let's dive into the generative capabilities of AI with a focus on generative AI and text-based models. Now, just to give some background on some popular models there are about five dominating models; Claude, ChatGPT, Copilot, Gemini, and DeepSeak. They are all hybrid models and

have their strengths and weaknesses, so to highlight these briefly:

- **ChatGPT:** Super versatile, it excels in chatting, creative writing, and coding assistance with decent sizing on its project scaling. Probably the most dominant as well as mainstream of all the AI in this list. Also has some API integration that can expand upon its usefulness as a tool.
- **Gemini:** Best for deep heavy searches, excels at deep diving into Google's ecosystem, this is Google's *'Flagship Model'*.
- **Claude:** Can Provide longer, more thoughtful responses, excels at writing and conversations, as well as good with code assistance. It is often praised as the safest of the models when it comes to sensitive tasks.
- **Copilot:** Its primary strength is the integration right into dev tools and is also strong with code assistance. Copilot excels when writing boilerplate and general assistance in good structure like function names and enforcing naming conventions. (It is powered by OpenAI's Codex variant)
- **DeepSeek:** DeepSeek excels when it comes to transparency as it is an open-source LLM. It can compete with GPT and Claude when it comes to performance and generative benchmarks but its primary strength is its full transparency.
- **Honorable Mentions:**
    - **Meta's Code Llama:** Developed by Meta, Code Llama is also open-source like DeepSeek, and is optimized for programming tasks. It is lightweight and excels in local development like DeepSeek, its strengths are privacy and offline capabilities.
    - **Mistral AI:** A French AI company focused on open-source LLMs, excels in high-performance problem solving, data analysis, and numerical computations.

B. Impact on Production Timelines

Now that you have some context into the main *plaers* influencing the developer field, let's break it down into the impact it has. There are three main key impact areas; Boilerplate and Autocompletion, Debugging and Refactoring, and Learning and Documenting.

1) Boilerplate and Autocompletion

The use of AI as a codebase assistant is now streamlining code generation in its entirety. One of the clearest examples is handling boilerplate code. Boilerplate code is repetitive, structural code that often slows down beginners and experienced developers alike. A beginner working in Flask or PHP would have to look up how to build the

API-first endpoint or routing setup. With tools like ChatGPT integrated into the IDE, this process can be completed in a matter of seconds, and if the user doesn't understand they only need to prompt it for an explanation until they start to comprehend the structure and functions. This is just one small example of how AI can increase personal and widespread development. To further this understanding here is a quote from a GitHub Research Article about *'Copilot's Impact on Developer Productivity and Happiness'*, developers who used GitHub Copilot completed the task significantly faster–55% faster than the developers who didn't use GitHub Copilot [5]. This is definitive proof of AI saving time for all levels of developers, but it also lowers the entry barrier for developers of all levels. My reflection of AI is that it is as if your knowledge was the road you drove on, and when you run into gaps or potholes, AI is like the bridge to cross these safely and efficiently. It is a magnet to find the needle in the haystack quickly, whether that be comprehension of a language's syntax, a library's base functionality, system admin setup commands from ufw to systemctl in Ubuntu, etc…

2) Debugging and Refactoring

AI plays a vital role in debugging and refactoring as well. It can range from easy assistance for logical errors, syntax errors, typos, etc… to more complex error correction with smart prompting. Streamlining debugging and refactors is a massive contribution to developers as these are some of the most time-consuming tasks. Think about that singular mistype function name that your IDE Might not pick up, sometimes the most minor errors cause the most tedious debug situations especially when it comes to the freshness of the session for the developer. Normally, what could range from minutes to hours dependent on the developer to find and identify, can be found by an AI model in mere seconds. This is also a vital tool to self-learn as a developer, which is one of the most vital skills needed in the field of developing code. Think about a novice web dev working on whatever project they are rewriting NAV bar after NAV bar and footer after footer, this doesn't sound time-consuming but when each page might have slight differences in functionality in the NAV for example this is how design errors could occur from copying and pastiung the code from the wrong page. Now consider a dev placed this code into an AI with a good prompt this AI has a strong chance to recommend the dev to use partials. The dev now learned about a more dynamic code practice, easily refactored it into their codebase with the AI's help, and now has more modular and clean code. This is a super simple example but these small situations alone build better developers and streamline production times.

3) Learning and Documenting

AI is also becoming a powerful personalized tutor and learning tool for developers of all levels. It can range to be an efficient teacher from the prompts of asking for a line-by-line explanation, adding comments into a script summarizing large codebase functionality, etc… tools like ChatGPT and Claude are helping devs comprehend and document their work faster

than ever before. Out with the old tedious search for a similar use case on StackOverflow and in with the magnet to find your Answer Needle in the haystack of information the Computer Science field is. This form of self-learning is even more useful because with smart prompting a user can have the generated explanations catered to their learning format. Now to touch upon the cons of this AI learning tool briefly, AI can often provide incorrect information but the same could be said from a StackOverflow post or a Reddit thread. But there is more on the negative of generative AI hallucinations in section II and section III, just a necessary disclaimer to touch upon. AI is also improving how developers document their work. With a single prompt or query, an LLM can generate inline comments in a script, to MarkDown documentation that is clear and concise for the whole codebase. This doesn't only speed up project life cycles and improve cognitive load across teams, it helps maintain clearer and more consistent communication for company teams, open source projects, and singular developers documenting codebases for their followers. As AI Senior Dev at Tesla, Andrej Kerpathy said "The hottest new programming language is English"[6]. Generative AI is allowing the unique combination of tutoring and self-documentation support with so much more as well. While it has its flaws, it is accelerating self-education, improving code readability, and making better developers with speed.

V. Conclusion

In conclusion, the integration and evolution of AI into the field of Computer Science over the last few years marks a paradigm shift in both the theoretical foundation as well as the practical applications of the field. From the possibility of redefining the meaning of what is computable in AI, to the transformation of how developers learn, debug, and code, AI is rapidly expanding and working its way to the core of a developer's toolkit and professors' curriculum topic. While this new software's ability to enhance efficiency and lower learning barriers is undeniable it is still accompanied by its flaws. The black-box nature of the LLMs and AI in general as well as the risk of overreliance on the models highlight the need for continued human oversight. The further evolution of AI in the future will hopefully expand the reach and depth of human innovation, creativity, logic, and problem-solving in Computer Science.

REFERENCES

[1] IBM Data and AI Team, “Types of Artificial Intelligence | IBM,” *www.ibm.com*, Oct. 12, 2023. https://www.ibm.com/think/topics/artificial-intelligence-types

[2] A. Vaswani *et al.*, “Attention Is All You Need | Google”, 2017. proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[3] E. Shein, “The Impact of AI on Computer Science Education – Communications of the ACM,” *Acm.org*, vol. 67, no. 9, Jul. 2024, https://cacm.acm.org/news/the-impact-of-ai-on-computer-science-education/

[4] GeeksforGeeks, “Turing Machine in TOC,” *GeeksforGeeks*, May 04, 2016. https://www.geeksforgeeks.org/turing-machine-in-toc/

[5] E. Kalliamvakou, “Research: quantifying GitHub Copilot’s impact on developer productivity and happiness,” *The GitHub Blog*, Sep. 07, 2022. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

[6] T. Bhattacharjee, “2024: The Year English Changed the Coding Game Forever | AIM,” *Analytics India Magazine*, Dec. 06, 2024. https://analyticsindiamag.com/ai-trends/2024-the-year-english-changed-the-coding-game-forever/ (accessed Jun. 16, 2025).