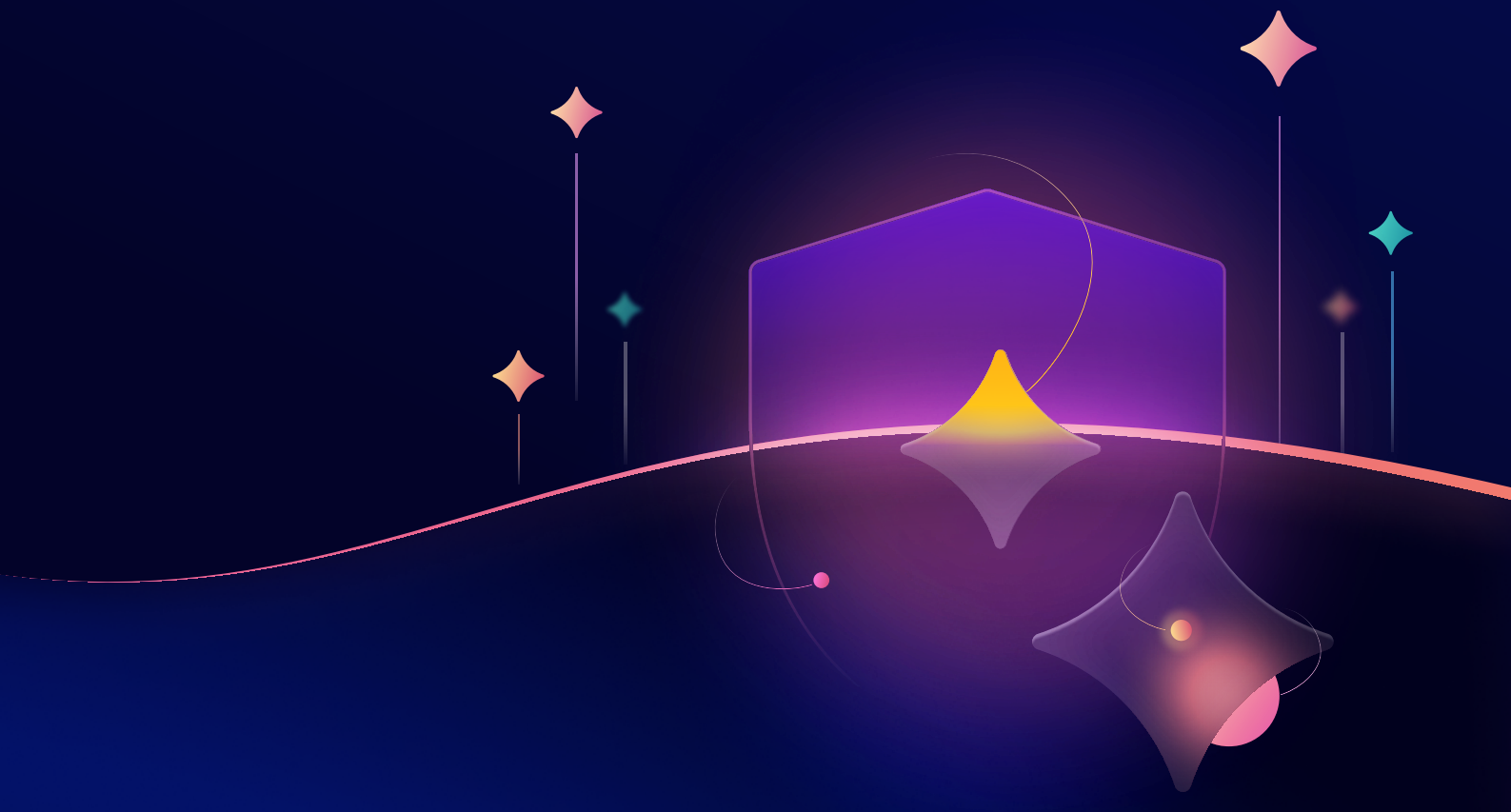
snyk

Understanding AI Models to Future-Proof your AppSec Program



Understanding AI Models to Future-Proof your AppSec Program

Today's discussions on AI are often affected by emotions like fear and awe, as well as the fact that AI is reduced to large language models (LLM) — but this isn't the entire story. So, let's step back from these fears and dig into the data explaining why AI is richer than LLMs.

AI today is really not that far off from historical descriptions

It makes sense to have some basic knowledge regarding AI. In the 1950s,[1] researchers and practitioners discussed what artificial intelligence could look like and how we would identify it. If you ask ChatGPT, it says: "Artificial Intelligence can be identified as a system or machine exhibiting capabilities to learn, understand, reason, perceive, and respond to its environment in ways that are traditionally associated with human intelligence, characterized by its ability to adapt to new situations, solve problems, make decisions, and understand complex ideas or patterns." That is actually a nice summary.

If you ask what AI is today, most of the time the answer describes an LLM — one type of deep learning neural network. But AI encompasses many more diverse and comprehensive technologies. Most of the time, using specialized AI (sometimes called "narrow AI") or a combination of methods is best.

To simplify things, AI methods roughly fall into two major buckets — a logical approach and a statistical one.[2] "Intelligent agents must be able to handle the complexity and uncertainty of the real world. Logical AI has focused mainly on the former, and statistical AI on the latter." As an example, in cybersecurity, using ML (statistical AI) in intrusion detection or logical AI in SAST is very common. For a more detailed discussion on the various approaches to AI, refer to Russell and Norvig's publication.[2]

Generative AI, which can produce assets like text, pictures, movies, or source code, and LLMs, which can use normal language as an interface (e.g. chat), are relatively new to the field. Generative AI works through a process called generative adversarial networks (GANs). The idea is to train an ML ("Discriminator") to classify an asset as valid or wrong (e.g. portraits of humans). Another ML ("Generator") tries to generate an asset (such as a picture) that is accepted by the discriminator. By training the two in conjunction — in what is called unsupervised learning — the system is capable of generating assets that are hardly, or no longer, distinguishable from a human product. An LLM extends this idea by using a Transformer. Text is then converted into a data format that is easy for ML to process ("Transformer"). Massive amounts of text are used to train the model. The model is used to generate textual answers to textual queries — often called "prompts". The answers to the same question can differ by using a factor that determines the amount of randomness the system uses — GPT calls this the "temperature". Current LLMs often have a cut off date to signify when the data used to train them was collected. However, AI developers plan to overcome this limitation by connecting LLMs to the internet so they have a current supply of information. Finally, LLMs can be specialized and trained for various subjects including source code generation (see Google's report on PaLM 2).

LLMs use a technique called deep learning which simulates the neurons of the human brain. But there are many other ways to make machines statistically learn — such as support vector machines, evolutionary algorithms, Markov chains, and more. As previously mentioned, there is also the other school of logical AI used to build expert systems with technologies like Prolog.

---

[1] It is argued that myths of ancient greeks already mentioned intelligent machines and the idea of an artificial intelligence is actually thousands of years old (see https://en.wikipedia.org/wiki/History_of_artificial_intelligence)

[2] https://homes.cs.washington.edu/~pedrod/papers/aaai06c.pdf

AI accelerates the SDLC, but with new security risks

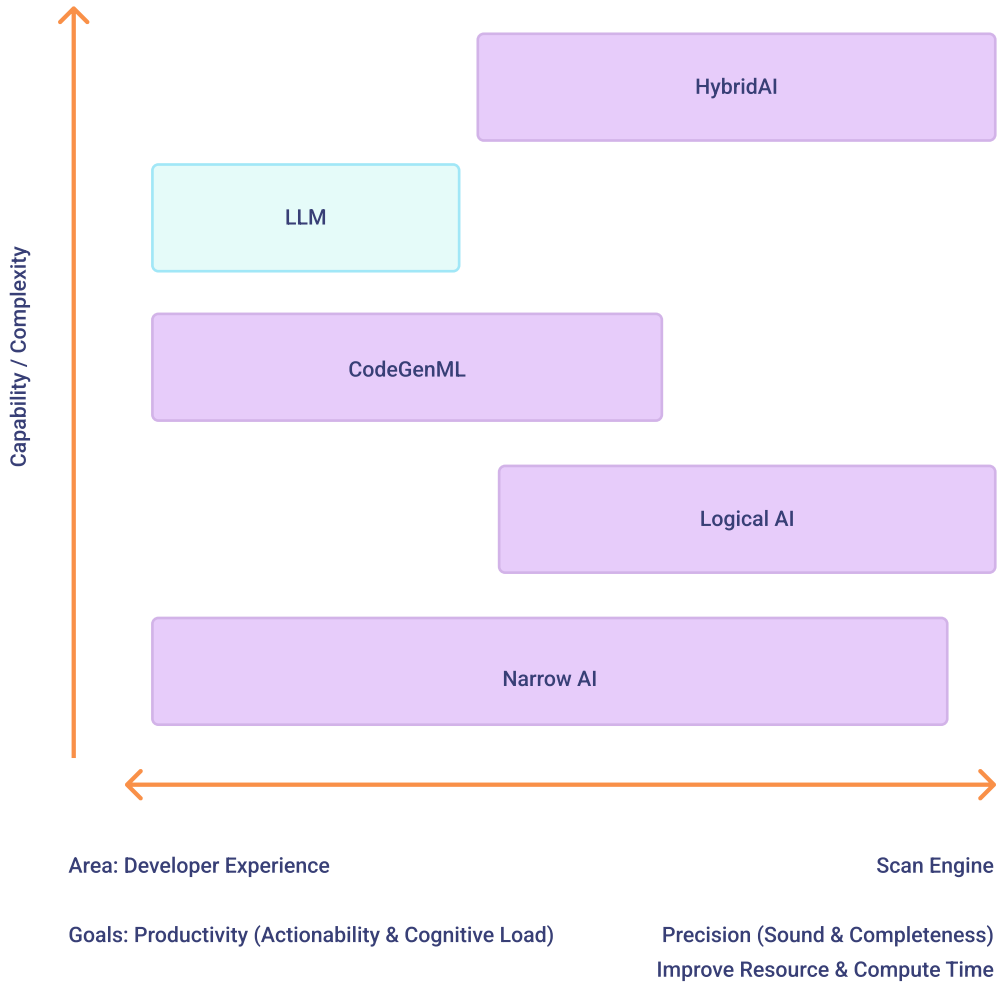
Concrete usage scenarios for AI in software development and delivery.

Usage	Location	Pros	Cons[3]
Code generation	SAST, IDE	Productivity, reduce cognitive load	Code quality, licensing, security
Code refactoring and upgrading	IDE	Productivity, reduce cognitive load	Code quality, licensing, security, bleeding information
Debugging	IDE	Reduce cognitive load	Bleeding information
Documentation	IDE, CI	Neglected task	Bleeding information
Code Understanding	IDE	Reduce cognitive load	
Find issues and generate fixes	SAST, SCA, IaC, container, IDE	Deeper and semantic analysis	False positives and runtime
Testing (Test generation, fuzzing, testing data)	IDE, DAST, fuzzing	Finding issues not thought of	Runtime, results need to be reacted upon, unrealistic test cases
Monitoring (processing data, combining data streams, detecting anomalies)	CD, Ops, SIEM	More data, find unknown patterns	

[3] A general drawback is that every usage scenario needs human oversight

Mapping out AI categories

AI technologies can be applied at every stage of software development. On the y-axis, the increasing complexity and strength of the AI tool is plotted. The x-axis documents features (e.g. elements with a direct developer experience) and capabilities (e.g. skills of the backend) of the given tools.



Note: All purple boxes show technologies where Snyk has its own solutions in place. We will use Snyk as an example in the following.

Narrow AI

Narrow AI, also called weak AI, refers to AI systems specifically designed to perform certain tasks, like providing product recommendations in an e-commerce platform or predicting user behavior in an application. Despite demonstrating intelligence within these specific tasks, these systems operate under confined parameters and don't possess the ability to understand, learn, or apply knowledge beyond their pre-programmed functions, making them ideal for targeted solutions in software development. For example, Snyk uses narrow AI in Snyk Code to find sources, sinks, and sanitizers.

Logical AI

Symbolic AI, also known as rule-based or good old-fashioned AI, is an approach that uses logic and specific symbols to represent problems and manipulate them to produce a solution — such as in expert systems where the logic for diagnosis is explicitly coded. For software developers, this could mean designing a troubleshooting program that uses predefined rules to identify and resolve software bugs, or creating a chess game where the AI uses a specific set of rules to decide the best move. Snyk Code, for example, uses a logic solver and rules compiled into a knowledgebase.

Generative AI

Generative AI is a subset of artificial intelligence that leverages machine learning techniques to produce new content — such as images, text, or music — that is similar to the training data. For software developers, this could mean creating a program that auto-generates human-like text for chatbots or automated responses, or developing an AI model that designs new, visually appealing user interfaces based on a set of existing, high-quality designs.

LLMs

LLM, short for large language model, is a type of artificial intelligence model that has been trained on a vast amount of text data, enabling it to generate human-like text based on given prompts. For software developers, this could mean leveraging an LLM to build applications like an automated customer service chatbot, or to create a code review tool that generates natural language comments explaining potential issues in the written code.

Combining AIs or Hybrid AI

Hybrid AI is an approach that combines different types of artificial intelligence methodologies — such as symbolic AI and machine learning — with elements of human intelligence to harness the strengths and offset the weaknesses of each. In software development, this could mean building a system that uses rule-based AI for initial decision-making, machine learning for pattern recognition and prediction, and incorporates human intelligence for tasks requiring creativity, empathy, or complex judgment — like final approval of generated content or intricate problem-solving. As an example, the Snyk security team uses the Snyk Code engine and ML technologies to build the Snyk Code knowledgebase. Another example is Code Fix, which uses a generative AI to build the fix suggestions and logic AI to find, and later control, the suggestions for feasibility.

A single AI model is not enough

In summary, AI is more than LLMs and is already in use in a variety of solutions. LLMs provide an amazing new interface and limitless possibilities, but they aren't the only tool at our disposal. Adding the other elements of AI in this equation expands coverage, intelligence, and opportunities of AI in application security. If you want to see how, Snyk's DeepCode AI can turbo charge your SDLC, book a demo with us or try Snyk by signing up on our website.

BOOK A DEMO