AI-assisted Development

Instructors **Battista Biggio**, **Angelo Sotgiu** and **Leonardo Regano**

M.Sc. in Computer Engineering, Cybersecurity and Artificial Intelligence
University of Cagliari, Italy

Thanks to Luca Scionis for helping prepare this material.

Disclaimer

Before to start, just a few thoughts:

- we are in a rapidly evolving, unpredictable scenario

Disclaimer

Before to start, just a few thoughts:

- we are in a rapidly evolving, unpredictable scenario
- on one side, the model providers market is growing (someone is also warning of an economic bubble...)

Disclaimer

Before to start, just a few thoughts:

- we are in a rapidly evolving, unpredictable scenario
- on one side, the model providers market is growing (someone is also warning of an economic bubble...)
- on the other side, companies are quickly integrating AI into their processes, also due to the fear of falling behind their competitors

Disclaimer

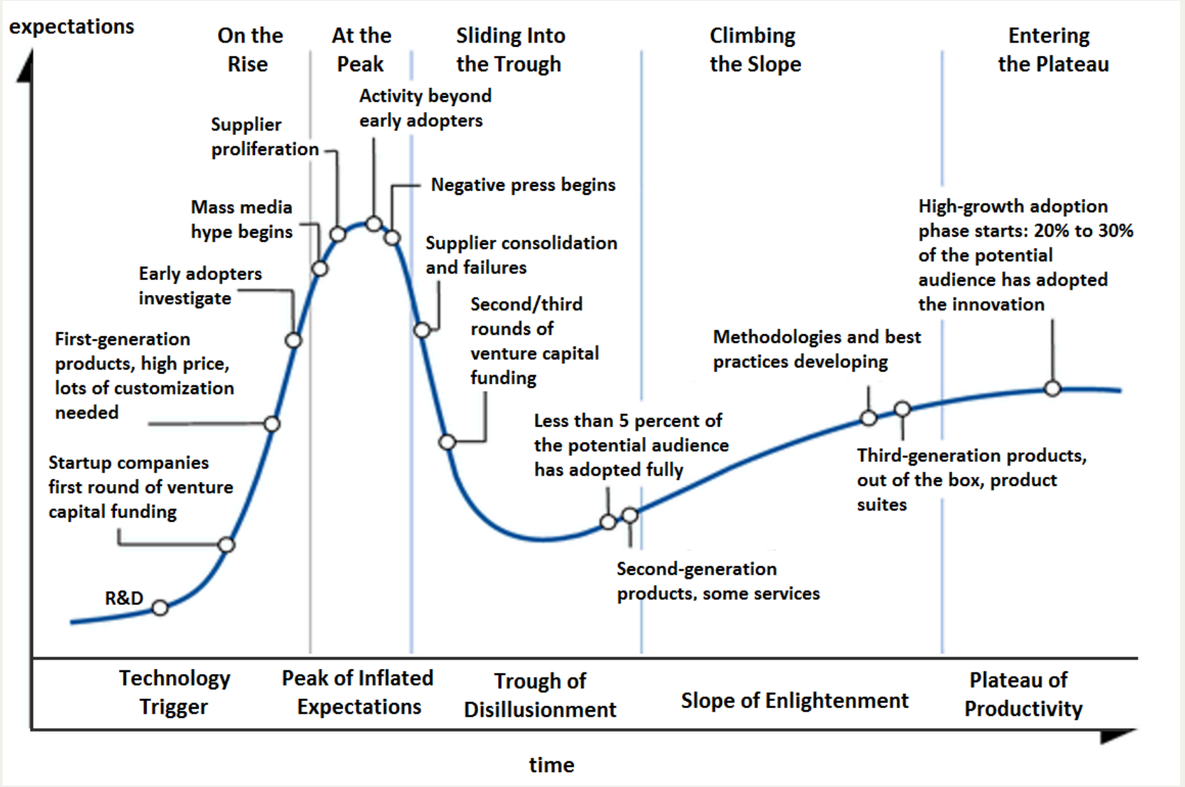
Before to start, just a few thoughts:

- we are in a rapidly evolving, unpredictable scenario
- on one side, the model providers market is growing (someone is also warning of an economic bubble...)
- on the other side, companies are quickly integrating AI into their processes, also due to the fear of falling behind their competitors
- AI is real and works... but we still don't know which technologies will consolidate and which will be abandoned

Disclaimer

Before to start, just a few thoughts:

- we are in a rapidly evolving, unpredictable scenario
- on one side, the model providers market is growing (someone is also warning of an economic bubble...)
- on the other side, companies are quickly integrating AI into their processes, also due to the fear of falling behind their competitors
- AI is real and works... but we still don't know which technologies will consolidate and which will be abandoned



Where are we? Here (complete data)

LLMs for coding

General-purpose LLMs are mostly trained on large text corpora from the web, which typically also include code (e.g., from GitHub, StackOverflow, etc.).

As for natural language, they gain deep understanding of programming languages and learn to generate new code.

LLMs for coding

General-purpose LLMs are mostly trained on large text corpora from the web, which typically also include code (e.g., from GitHub, StackOverflow, etc.).

As for natural language, they gain deep understanding of programming languages and learn to generate new code.

To improve the coding ability of LLMs, the training sets can be better curated to include more structured and comprehensive data related to developing tasks.

LLMs for coding

General-purpose LLMs are mostly trained on large text corpora from the web, which typically also include code (e.g., from GitHub, StackOverflow, etc.).

As for natural language, they gain deep understanding of programming languages and learn to generate new code.

To improve the coding ability of LLMs, the training sets can be better curated to include more structured and comprehensive data related to developing tasks.

Moreover, code-specific LLMs are often fine-tuned on tasks like code completion, bug fixing, refactoring, etc.

LLMs for coding

LLMs can be used in almost all the software development phases, potentially accelerating all those processes:

- project management
- requirements collection
- design
- implementation
- documentation
- code inspection (e.g., for bugs and security issues)
- optimization
- tests
- debugging and bug fix
- refactoring

LLMs for coding

Some concrete examples:

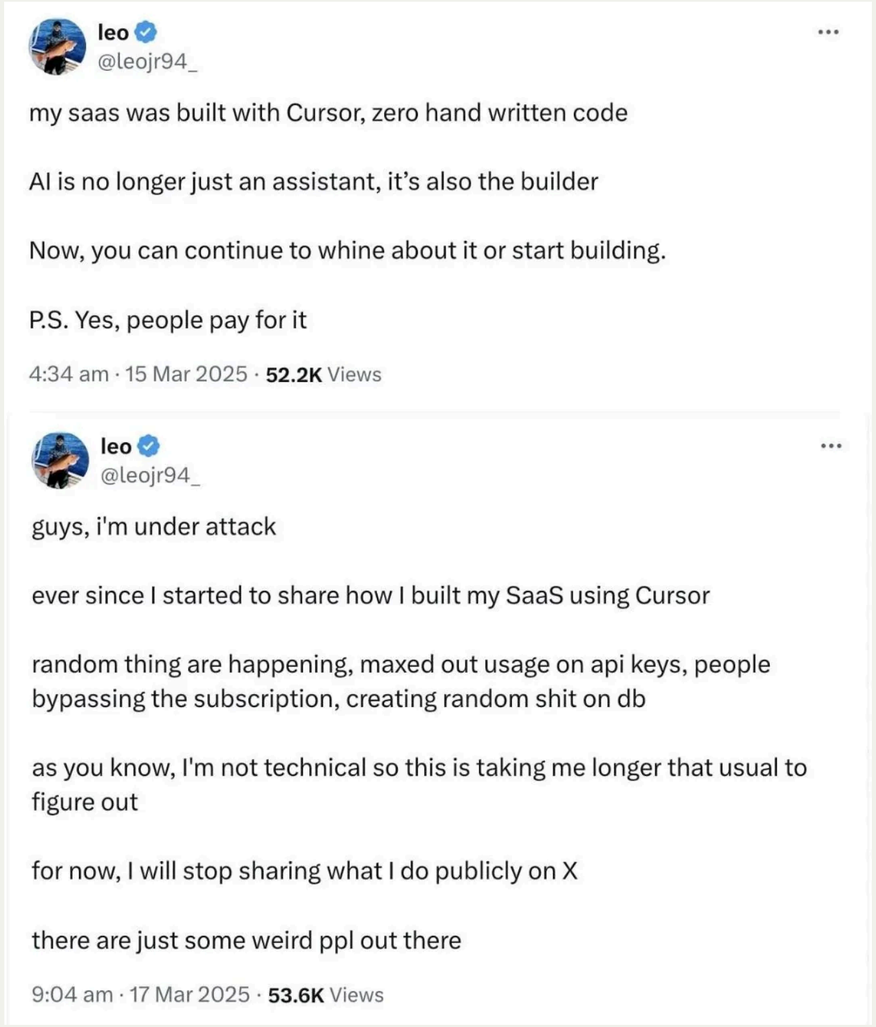
- autocompletion
- boilerplate code (sections of code that are repeated in multiple places with little to no variation)
- scaffolding (automated building of the fundamental software structure)
- refining ideas
- writing documentation and tests
- ...

LLMs for coding

Some concrete examples:

- autocompletion
- boilerplate code (sections of code that are repeated in multiple places with little to no variation)
- scaffolding (automated building of the fundamental software structure)
- refining ideas
- writing documentation and tests
- ...
- *vibe coding*: creating software without writing (and even reviewing) a single line of code, but only expressing in natural language all the requirements
    - is it the future? not yet...

Are we ready for vibe coding?



leo
@leojr94_

my saas was built with Cursor, zero hand written code

AI is no longer just an assistant, it's also the builder

Now, you can continue to whine about it or start building.

P.S. Yes, people pay for it

4:34 am · 15 Mar 2025 · **52.2K** Views

leo
@leojr94_

guys, i'm under attack

ever since I started to share how I built my SaaS using Cursor

random thing are happening, maxed out usage on api keys, people bypassing the subscription, creating random shit on db

as you know, I'm not technical so this is taking me longer that usual to figure out

for now, I will stop sharing what I do publicly on X

there are just some weird ppl out there

9:04 am · 17 Mar 2025 · **53.6K** Views

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled
- Models' knowledge is "freezed" after their training (but this can be mitigated, as we'll see later)

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled
- Models' knowledge is "freezed" after their training (but this can be mitigated, as we'll see later)
- The "context" that can be processed by LLMs is limited (although it's contantly increasing), and with large contexts models seem to lose the global overview of the problem

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled
- Models' knowledge is "freezed" after their training (but this can be mitigated, as we'll see later)
- The "context" that can be processed by LLMs is limited (although it's contantly increasing), and with large contexts models seem to lose the global overview of the problem
- Generated code *works* (not always), but usually does not take into account all the non-functional requirements (especially concerning security). Unless you carefully instruct them to do so (but even in this case, this is not guaranteed)

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled
- Models' knowledge is "freezed" after their training (but this can be mitigated, as we'll see later)
- The "context" that can be processed by LLMs is limited (although it's contantly increasing), and with large contexts models seem to lose the global overview of the problem
- Generated code *works* (not always), but usually does not take into account all the non-functional requirements (especially concerning security). Unless you carefully instruct them to do so (but even in this case, this is not guaranteed)

But wait, are human developers infallible?

Answer: No

- Models suffer from hallucinations (after all, they still are probabilistic next-token predictors!)
- The quality of the code with which the model are trained cannot be controlled
- Models' knowledge is "freezed" after their training (but this can be mitigated, as we'll see later)
- The "context" that can be processed by LLMs is limited (although it's contantly increasing), and with large contexts models seem to lose the global overview of the problem
- Generated code *works* (not always), but usually does not take into account all the non-functional requirements (especially concerning security). Unless you carefully instruct them to do so (but even in this case, this is not guaranteed)

But wait, are human developers infallible?

We will now review the current technologies for AI-assisted development, and then better focus on their issues and limitations, and the best practices that should be followed.

Retrieval-Augmented Generation

The LLMs applications we have seen since now only rely on the models "knowledge" acquired with their training and the content processed within their context window (the prompt).

Retrieval-Augmented Generation

The LLMs applications we have seen since now only rely on the models "knowledge" acquired with their training and the content processed within their context window (the prompt).

Retrieval-Augmented Generation (RAG) enhances them by providing access to external data sources, by means of a pretrained neral retriever.

Retrieval-Augmented Generation

The LLMs applications we have seen since now only rely on the models "knowledge" acquired with their training and the content processed within their context window (the prompt).

Retrieval-Augmented Generation (RAG) enhances them by providing access to external data sources, by means of a pretrained neral retriever.

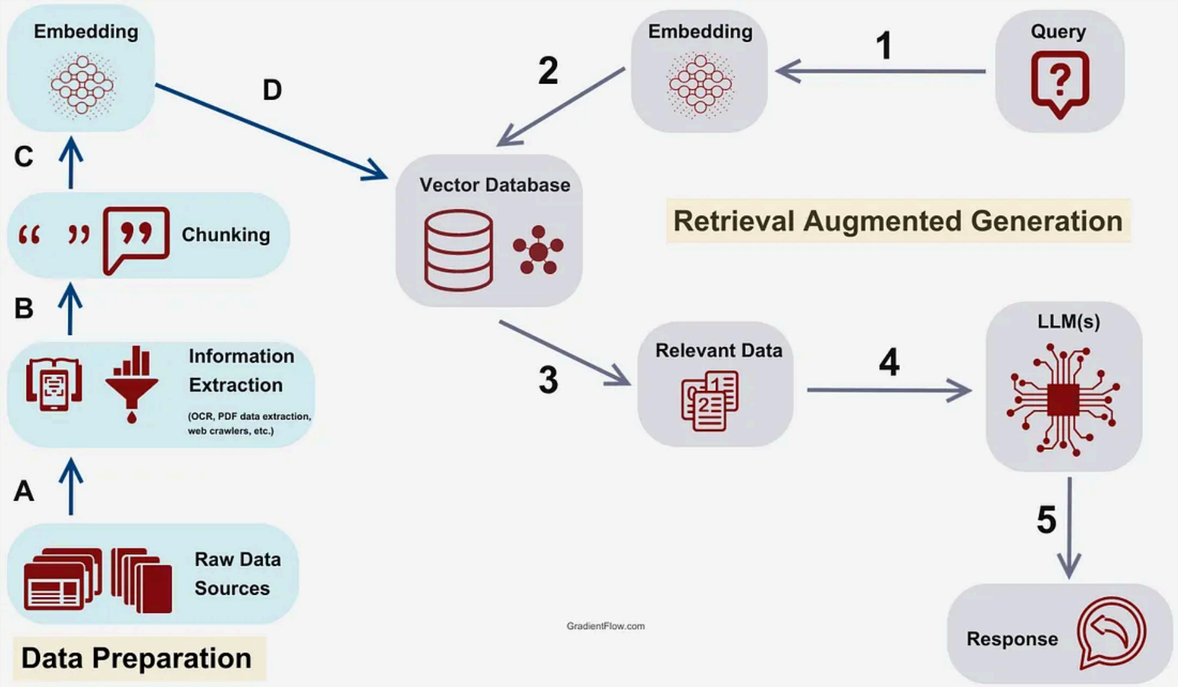


Image source

Retrieval-Augmented Generation

The LLMs applications we have seen since now only rely on the models "knowledge" acquired with their training and the content processed within their context window (the prompt).

Retrieval-Augmented Generation (RAG) enhances them by providing access to external data sources, by means of a pretrained neral retriever.

This mitigates some of the previosuly discussed limitations: makes the answers more factual, update the model knowledge without retraining it, and allows customizing the AI-tools for specific tasks (e.g., by linking LLMs to companies data assets)

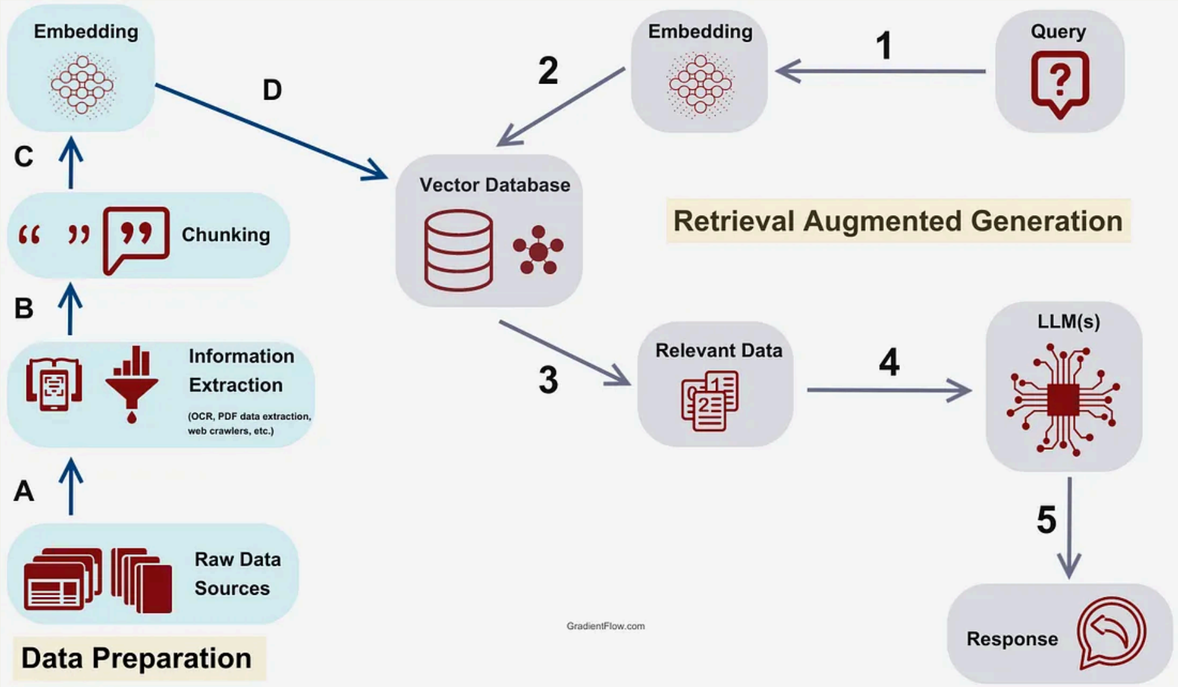


Image source

Chatbots

Commonly used AI Chat applications: ChatGPT, Claude, Microsoft Copilot, Gemini, etc. (with both general-purpose and code-specific models)

Chatbots

Commonly used AI Chat applications: ChatGPT, Claude, Microsoft Copilot, Gemini, etc. (with both general-purpose and code-specific models)

Pros:

- they provide a friendly-like environment, where users can query the model and receive answers on the task
- very good for learning and finding new solutions

Chatbots

Commonly used AI Chat applications: ChatGPT, Claude, Microsoft Copilot, Gemini, etc. (with both general-purpose and code-specific models)

Pros:

- they provide a friendly-like environment, where users can query the model and receive answers on the task
- very good for learning and finding new solutions

Cons:

- limited functionalities and processed context
- low customization
- non-trivial tasks can be hard to accomplish, e.g., need to copy-paste the specific code snippets back and forth from the prompt area to other applications, defining the request afterwards (e.g., “check bugs in this code”, or “help me sanitizing the input”, etc.)

In-app Assistants

Local or remote LLMs are natively integrated in the applications (e.g., an IDE), and might also directly control some of its functionalities.

Alternatively, full AI-based IDEs are available, such as Cursor.

In-app Assistants

Local or remote LLMs are natively integrated in the applications (e.g., an IDE), and might also directly control some of its functionalities.

Alternatively, full AI-based IDEs are available, such as Cursor.

Pros:

- they better capture the general task context (e.g., by accessing all the required project files)
- seamless usage

In-app Assistants

Local or remote LLMs are natively integrated in the applications (e.g., an IDE), and might also directly control some of its functionalities.

Alternatively, full AI-based IDEs are available, such as Cursor.

Pros:

- they better capture the general task context (e.g., by accessing all the required project files)
- seamless usage

Cons:

- effective only through proper app integration
- integration might be complex and not always available (or limited to specific functionalities)

CLI Tools

Almost all the LLMs providers and third-party alternatives currently supports a CLI (Command Line Interface) application to use their tools through the command line. (e.g., Codex CLI, Gemini CLI, Claude Code).

CLI Tools

Almost all the LLMs providers and third-party alternatives currently supports a CLI (Command Line Interface) application to use their tools through the command line. (e.g., Codex CLI, Gemini CLI, Claude Code).

Pros:

- easy to be integrated into existing development processes and combined with other tools
- lightweight and fast, scripting available

CLI Tools

Almost all the LLMs providers and third-party alternatives currently supports a CLI (Command Line Interface) application to use their tools through the command line. (e.g., Codex CLI, Gemini CLI, Claude Code).

Pros:

- easy to be integrated into existing development processes and combined with other tools
- lightweight and fast, scripting available

Cons:

- some complex tasks may require multiple commands
- less usable in certain scenarios, e.g., when interactions are needed, or when the output should be further processed

Agents

*"An **agent** is just something that acts (agent comes from the Latin agere, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals. A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome."*

Artificial Intelligence: A Modern Approach", 1995

Agents

> *"An **agent** is just something that acts (agent comes from the Latin agere, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals. A **rational agent** is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome."*
>
> *Artificial Intelligence: A Modern Approach", 1995*

So what is an AI agent?

LLM + tools calling + RAG = autonomy + reasoning and planning + tool use and integration + learning and memory

Agents

Thought-Action-Observation Cycle:

Agents

Thought-Action-Observation Cycle:

- Thought: the LLM part of the Agent decides what the next step should be

Agents

Thought-Action-Observation Cycle:

- Thought: the LLM part of the Agent decides what the next step should be
- Action: the agent takes an action by calling the tools with the associated arguments (through json messages, code execution or function calling)

Agents

Thought-Action-Observation Cycle:

- Thought: the LLM part of the Agent decides what the next step should be
- Action: the agent takes an action by calling the tools with the associated arguments (through json messages, code execution or function calling)
- Observation: the model reflects on the response from the tool

Agents

Thought-Action-Observation Cycle:

- Thought: the LLM part of the Agent decides what the next step should be
- Action: the agent takes an action by calling the tools with the associated arguments (through json messages, code execution or function calling)
- Observation: the model reflects on the response from the tool

Pros:

- complete automation of entire workflows
- interaction and orchestration of external systems (e.g., version control, CI/CD, deployment)

Agents

Thought-Action-Observation Cycle:

- Thought: the LLM part of the Agent decides what the next step should be
- Action: the agent takes an action by calling the tools with the associated arguments (through json messages, code execution or function calling)
- Observation: the model reflects on the response from the tool

Pros:

- complete automation of entire workflows
- interaction and orchestration of external systems (e.g., version control, CI/CD, deployment)

Cons:

- high responsibilities and power must be delegated
- complex to build and manage

Agents Frameworks

There are several frameworks to build LLM-powered agents:

- LangChain/LangGraph
- LlamaIndex
- smolagents

AGENTS.md

`Agents.md` (https://agents.md/) is an open format to provide context and instructions to help AI coding agents work on your project

- it's similar to the `README.md`, but for agents

AGENTS.md

`Agents.md` (https://agents.md/) is an open format to provide context and instructions to help AI coding agents work on your project

- it's similar to the `README.md`, but for agents

It usually contains:

- project overview
- build and test commands
- code style guidelines
- testing instructions
- security considerations
- guidelines for versioning systems (e.g., git)

Model Context Protocol

Model Context Protocol (MCP) is an open-source standard for connecting AI applications to external systems, such as data sources, tools and workflows.

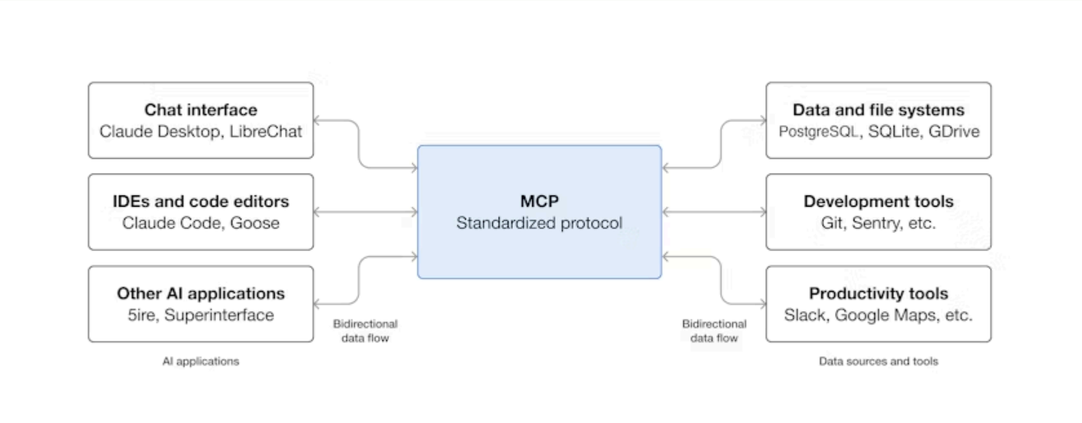


Image source

Model Context Protocol

Model Context Protocol (MCP) is an open-source standard for connecting AI applications to external systems, such as data sources, tools and workflows.

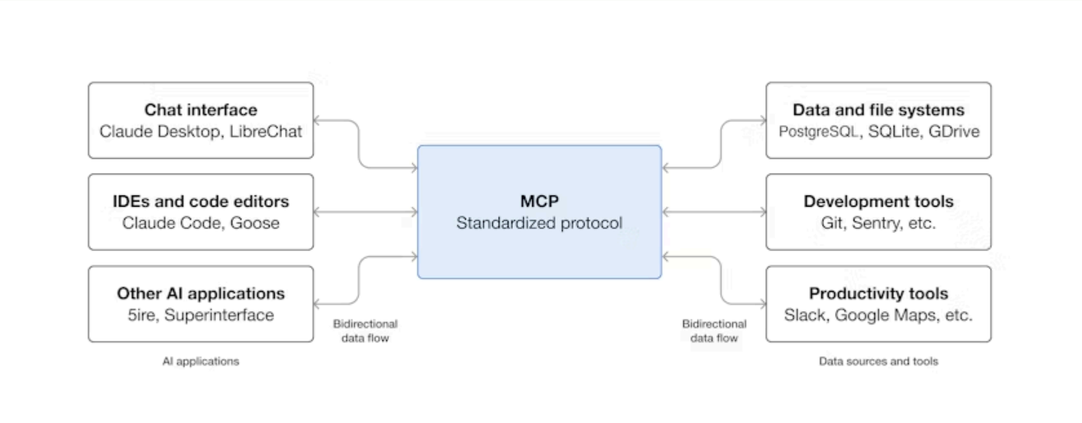


Image source

It reduces development time and complexity when building/integrating AI-based applications, providing a standardized way to access a wide range of external systems and increasing scalability.

Model Context Protocol

MCP follows a stateful client-server architecture where an **MCP host** - an AI application - establishes connections to one or more local or remote **MCP servers**. The MCP host accomplishes this by creating one **MCP client** for each MCP server. Each MCP client maintains a dedicated one-to-one connection with its corresponding MCP server.

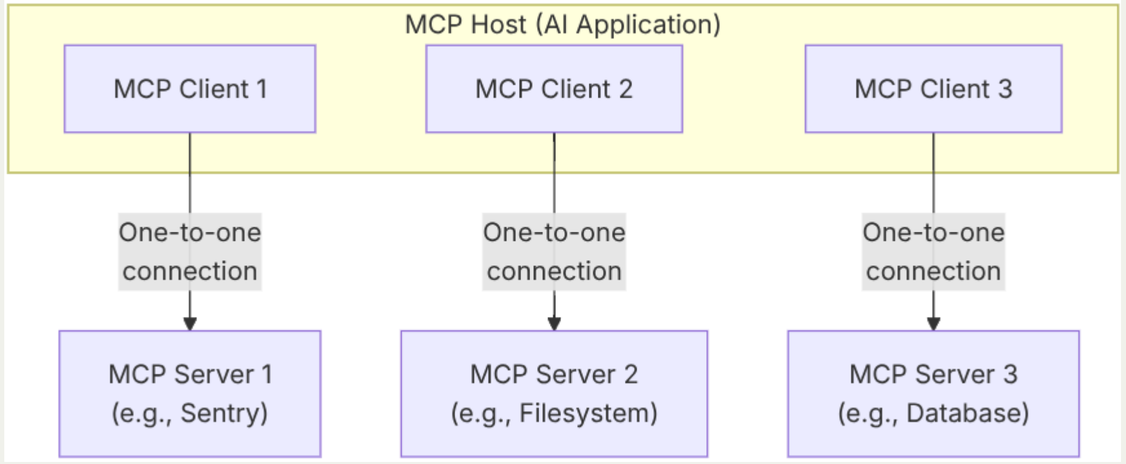


Image source

Model Context Protocol

MCP follows a stateful client-server architecture where an **MCP host** - an AI application - establishes connections to one or more local or remote **MCP servers**. The MCP host accomplishes this by creating one **MCP client** for each MCP server. Each MCP client maintains a dedicated one-to-one connection with its corresponding MCP server.

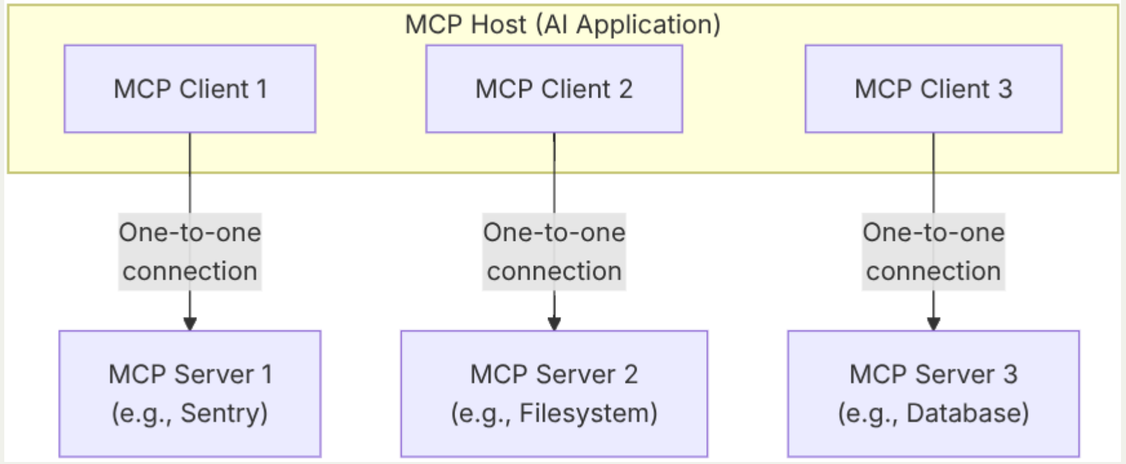


Image source

The exchanged data are JSON-RPC messages that allows to discover the available tools and actions, invoke them and receive the outputs.

Model Context Protocol

MCP servers expose tools, resources and (reusable) prompts.

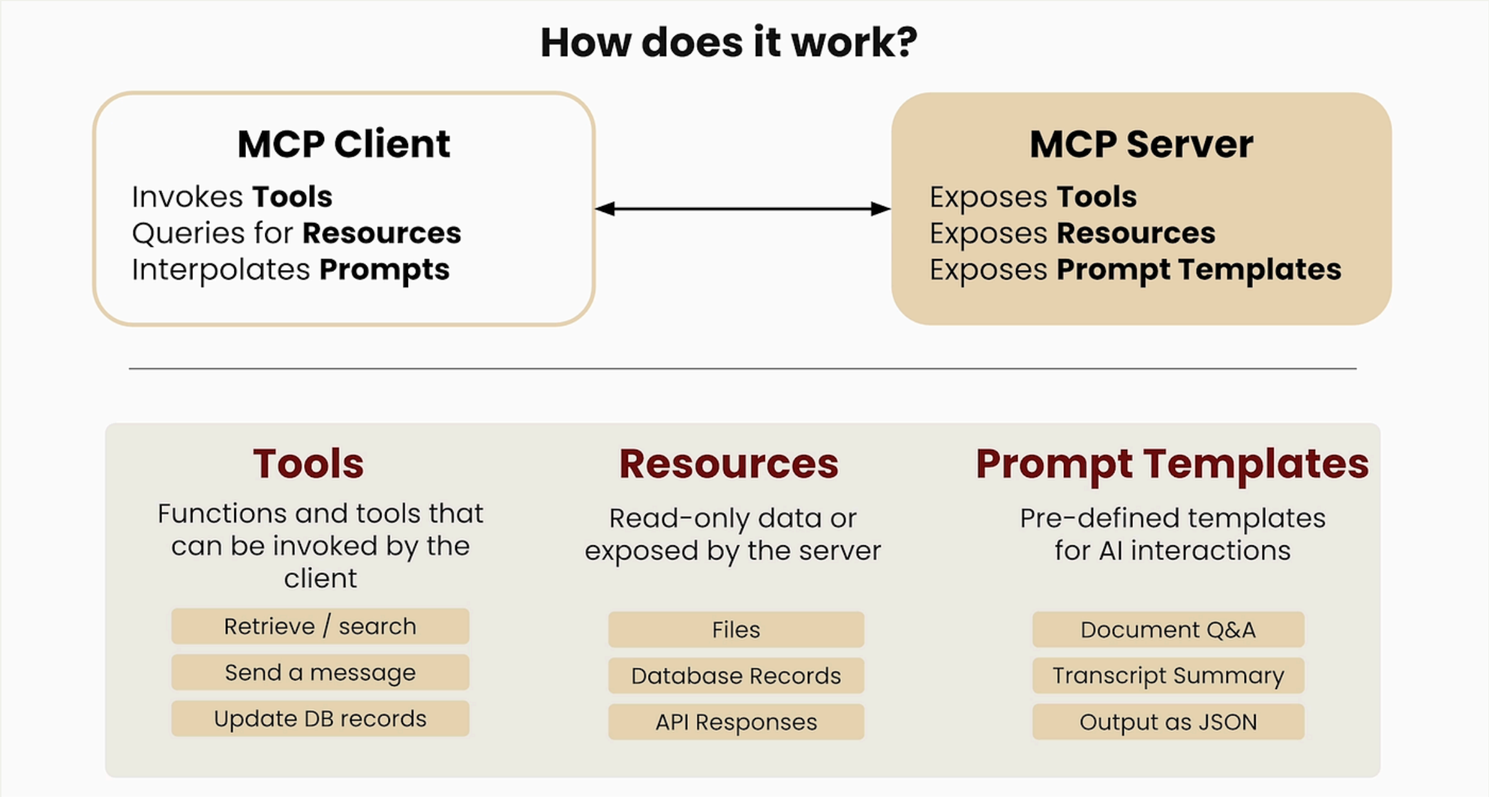


Image source

Security Issues

OWAPS Top-10 for LLM Applications

LLMs are an attack surface: real vulnerabilities allow attackers to execute arbitrary commands on the users machine, read and write arbitrary files on the disk, all through LLMs (especially agents) running on their device.

OWAPS Top-10 for LLM Applications

LLMs are an attack surface: real vulnerabilities allow attackers to execute arbitrary commands on the users machine, read and write arbitrary files on the disk, all through LLMs (especially agents) running on their device.

OWASP (Open Worldwide Application Security Project) collects every year the most concerning security issues for LLMs. Let's quickly see the 2025 top-10.

OWAPS Top-10 for LLM Applications

LLMs are an attack surface: real vulnerabilities allow attackers to execute arbitrary commands on the users machine, read and write arbitrary files on the disk, all through LLMs (especially agents) running on their device.

OWASP (Open Worldwide Application Security Project) collects every year the most concerning security issues for LLMs. Let's quickly see the 2025 top-10.

1. **Prompt Injection.** The LLMs behavior, which depends on the input prompt, might be altered in unintended ways. Malicious actors can potentially exploit that to achieve disruptive results. Often the prompt injections are imperceptible to humans.
    - Mitigations: constrain model behavior with the system prompt; define, validate and filter input/output; enforce privilege control and least privilege access; require human approval for high-risk actions; separate user prompts from external content.

2. **Sensitive Information Disclosure.** During their training, LLMs incorporate sensitive data (countermeasures are often not sufficient to filter this content). It might happen that this data is then exposed in their responses, either unintentionally or after malicious prompts.
    - Mitigations: data sanitization; input validation; access controls; privacy-preserving learning techniques; user education and transparency; secure system configuration.

2. **Sensitive Information Disclosure.** During their training, LLMs incorporate sensitive data (countermeasures are often not sufficient to filter this content). It might happen that this data is then exposed in their responses, either unintentionally or after malicious prompts.
    - Mitigations: data sanitization; input validation; access controls; privacy-preserving learning techniques; user education and transparency; secure system configuration.
3. **Supply Chain.** The adoption of outsourced LLMs and related frameworks might introduce vulnerable or tampered components (e.g., libraries, the LLMs themselves).

2. **Sensitive Information Disclosure.** During their training, LLMs incorporate sensitive data (countermeasures are often not sufficient to filter this content). It might happen that this data is then exposed in their responses, either unintentionally or after malicious prompts.
    - Mitigations: data sanitization; input validation; access controls; privacy-preserving learning techniques; user education and transparency; secure system configuration.
3. **Supply Chain.** The adoption of outsourced LLMs and related frameworks might introduce vulnerable or tampered components (e.g., libraries, the LLMs themselves).
4. **Data and Model Poisoning.** The data used to train LLMs can be manipulated to alter the model behavior (e.g., introduce vulnerabilities, backdoor or biases).

5. **Improper Output Handling.** Insufficient validation, sanitization, and handling of the outputs generated by large language models before they are passed downstream to other components and systems.

5. **Improper Output Handling.** Insufficient validation, sanitization, and handling of the outputs generated by large language models before they are passed downstream to other components and systems.
6. **Excessive Agency.** Enables damaging actions to be performed in response to unexpected, ambiguous or manipulated outputs from an LLM.

5. **Improper Output Handling.** Insufficient validation, sanitization, and handling of the outputs generated by large language models before they are passed downstream to other components and systems.
6. **Excessive Agency.** Enables damaging actions to be performed in response to unexpected, ambiguous or manipulated outputs from an LLM.
7. **System Prompt Leakage.** It is a concern if the prompt contains security-sensitive information.

5. **Improper Output Handling.** Insufficient validation, sanitization, and handling of the outputs generated by large language models before they are passed downstream to other components and systems.
6. **Excessive Agency.** Enables damaging actions to be performed in response to unexpected, ambiguous or manipulated outputs from an LLM.
7. **System Prompt Leakage.** It is a concern if the prompt contains security-sensitive information.

To mitigate almost all of them it is necessary to follow a proactive approach, conducting *adversarial* testing and attack simulations.

5. **Improper Output Handling.** Insufficient validation, sanitization, and handling of the outputs generated by large language models before they are passed downstream to other components and systems.
6. **Excessive Agency.** Enables damaging actions to be performed in response to unexpected, ambiguous or manipulated outputs from an LLM.
7. **System Prompt Leakage.** It is a concern if the prompt contains security-sensitive information.

To mitigate almost all of them it is necessary to follow a proactive approach, conducting *adversarial* testing and attack simulations.

OWASP also collects the available tools to improve the LLMs security: https://genai.owasp.org/ai-security-solutions-landscape/

Other Issues

Recall three concepts:

- LLMs are probabilistic next-token predictors
- they are trained on large text and source code corpora scraped from the web
- generative models learn the training data distribution, and try to produce content from the same distribution

Other Issues

Recall three concepts:

- LLMs are probabilistic next-token predictors
- they are trained on large text and source code corpora scraped from the web
- generative models learn the training data distribution, and try to produce content from the same distribution

This in turn means that the generated code might be *inspired* by untrusted, unverified, potentially buggy and vulnerable source code!

Other Issues

Recall three concepts:

- LLMs are probabilistic next-token predictors
- they are trained on large text and source code corpora scraped from the web
- generative models learn the training data distribution, and try to produce content from the same distribution

This in turn means that the generated code might be *inspired* by untrusted, unverified, potentially buggy and vulnerable source code!

Additionally (although we will not address that), the produced code may be protected by copyright...

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection
- hardcoded API keys and other secrets

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection
- hardcoded API keys and other secrets
- authentication logic implemented entirely on the client side

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection
- hardcoded API keys and other secrets
- authentication logic implemented entirely on the client side
- logging errors (e.g., insufficient filtering or complete absence of logs)

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection
- hardcoded API keys and other secrets
- authentication logic implemented entirely on the client side
- logging errors (e.g., insufficient filtering or complete absence of logs)
- use of dangerous functions

Introduction of Software Vulnerabilities

By default, LLMs do not take into account all the non-functional requirements. The generated code often presents:

- lack of input validation/sanitization and other trivial errors exposing to well-known vulnerabilities like XSS and SQL injection
- hardcoded API keys and other secrets
- authentication logic implemented entirely on the client side
- logging errors (e.g., insufficient filtering or complete absence of logs)
- use of dangerous functions
- outdated or non-existent dependencies

Introduction of Software Vulnerabilities

In addition to the classical strategies (e.g., code review, testing tools), some mitigations can be applied by carefully building the prompts in order to reduce insecure code generation:

- **General Security-Oriented System Prompts:** include in the prompts security-related context to encourage the LLM writing secure code

Introduction of Software Vulnerabilities

In addition to the classical strategies (e.g., code review, testing tools), some mitigations can be applied by carefully building the prompts in order to reduce insecure code generation:

- **General Security-Oriented System Prompts:** include in the prompts security-related context to encourage the LLM writing secure code
- **Language or Application-Specific Prompts:** when possible, use language-specific or application-specific security prompts, targeting known vulnerabilities or common pitfalls. Example: https://github.com/wiz-sec-public/secure-rules-files

Introduction of Software Vulnerabilities

In addition to the classical strategies (e.g., code review, testing tools), some mitigations can be applied by carefully building the prompts in order to reduce insecure code generation:

- **General Security-Oriented System Prompts:** include in the prompts security-related context to encourage the LLM writing secure code
- **Language or Application-Specific Prompts:** when possible, use language-specific or application-specific security prompts, targeting known vulnerabilities or common pitfalls. Example: https://github.com/wiz-sec-public/secure-rules-files
- **Self-Reflection for Security Review:** add a self-reflective review step immediately after code generation, to explicitly identify and address security vulnerabilities

Introduction of Software Vulnerabilities

In addition to the classical strategies (e.g., code review, testing tools), some mitigations can be applied by carefully building the prompts in order to reduce insecure code generation:

- **General Security-Oriented System Prompts:** include in the prompts security-related context to encourage the LLM writing secure code
- **Language or Application-Specific Prompts:** when possible, use language-specific or application-specific security prompts, targeting known vulnerabilities or common pitfalls. Example: https://github.com/wiz-sec-public/secure-rules-files
- **Self-Reflection for Security Review:** add a self-reflective review step immediately after code generation, to explicitly identify and address security vulnerabilities
- **Integration of Security Tools:** agents can be integrated (e.g., through MCP) with external tools to enhance the development and review process

Can LLMs detect never-seen-before vulnerabilities?

Our research experience:

Explainability-based debugging of machine learning for vulnerability discovery

Evaluating Line-level Localization Ability of Learning-based Code Vulnerability Detection Models

...other works have shown that also more recent models are not yet reliable (although agents seem promising)

LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks

MCP security

Another security concern is related to the MCP protocol, whose design initially did not consider security aspects.

- there have already been successful attacks that exploited MCP

MCP security

Another security concern is related to the MCP protocol, whose design initially did not consider security aspects.

- there have already been successful attacks that exploited MCP

For MCP-based AI applications, new threats emerged, requiring specific prevention and mitigation strategies

- but also "conventional" threats still stand (and can even be amplified in these scenarios)
- interesting readings: MCP Security Best Practices, Model Context Protocol (MCP) Security

(some) MCP Attacks and Mitigations

- **Identity Spoofing:** weak or misconfigured authentication in MCP deployments could allow attackers to impersonate legitimate clients or the agents acting on their behalf, corrupting audit trails or gaining unauthorized access to server resources
  - Mitigation: secure identity, authentication and authorization for all the involved actors (users, agents, servers, etc.); secure delegation and access control

(some) MCP Attacks and Mitigations

- **Identity Spoofing:** weak or misconfigured authentication in MCP deployments could allow attackers to impersonate legitimate clients or the agents acting on their behalf, corrupting audit trails or gaining unauthorized access to server resources
  - Mitigation: secure identity, authentication and authorization for all the involved actors (users, agents, servers, etc.); secure delegation and access control
- **Tool Poisoning:** malicious modification of tool metadata, configuration, or descriptors injected into clients via the tools/list method. This can cause AI agents or MCP components to invoke, trust, or execute compromised tools, potentially leading to data leaks or system compromise. As the MCP specification notes, 'descriptions of tool behavior such as annotations should be considered untrusted, unless obtained from a trusted server' (Key Principles), making tool poisoning a recognized risk when clients connect to unvetted servers.
  - Mitigation: input and data validation/sanitization/filtering, guardrails

- **Resource Content Poisoning:** attackers embed hidden malicious instructions within data sources (databases, documents, API responses) that MCP servers retrieve and provide to LLMs, causing the poisoned content to execute as commands when processed, effectively achieving persistent prompt injection through trusted data channels rather than direct user input.
    - Mitigation: input and data validation/sanitization/filtering, guardrails

- **Resource Content Poisoning:** attackers embed hidden malicious instructions within data sources (databases, documents, API responses) that MCP servers retrieve and provide to LLMs, causing the poisoned content to execute as commands when processed, effectively achieving persistent prompt injection through trusted data channels rather than direct user input.
    - Mitigation: input and data validation/sanitization/filtering, guardrails
- **Overreliance on the LLM:** MCP server developers may implement overly permissive tools, assuming the LLM will invoke them correctly and safely. However, model-level controls (trained refusals, safety classifiers, etc.) are not ironclad—even capable models can be manipulated through prompt injection, make errors in judgment, or be replaced with weaker models that lack equivalent safeguards.
    - Mitigation: secure tool and UX design

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)
- Iterate on the prompts, assessing their effectiveness and improving them if needed

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)
- Iterate on the prompts, assessing their effectiveness and improving them if needed
- Reset the context when needed

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)
- Iterate on the prompts, assessing their effectiveness and improving them if needed
- Reset the context when needed
- Try to isolate the software components (e.g., the security-critical ones), applying different policies on when and how use AI during their development

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)
- Iterate on the prompts, assessing their effectiveness and improving them if needed
- Reset the context when needed
- Try to isolate the software components (e.g., the security-critical ones), applying different policies on when and how use AI during their development
- Implement strong version control practices (frequent commits on small-batch modifications)

Some Practical *Dos*

- Check if the model you are using works well on the specific programming languages used in your projects
- Give as much valuable *context* as possible to the model (e.g., RAG, AGENTS.md, etc.)
- Be specific in your instructions. For instance, explicitly state the frameworks, libraries, technologies to use (e.g., for secure authentication)
- Iterate on the prompts, assessing their effectiveness and improving them if needed
- Reset the context when needed
- Try to isolate the software components (e.g., the security-critical ones), applying different policies on when and how use AI during their development
- Implement strong version control practices (frequent commits on small-batch modifications)
- Use a least-privileges approach for agents

Some Practical *Don'ts*

- Don't be lazy! AI is not a shortcut, you still need to constantly improve your knowledge and skills (from the foundations...)

Some Practical *Don'ts*

- Don't be lazy! AI is not a shortcut, you still need to constantly improve your knowledge and skills (from the foundations...)
- Never trust blindly the generated code

Some Practical *Don'ts*

- Don't be lazy! AI is not a shortcut, you still need to constantly improve your knowledge and skills (from the foundations...)
- Never trust blindly the generated code
- The generated code might be overly complex and difficult to understand, even for simple problems: it's almost never a good solution

Some Practical *Don'ts*

- Don't be lazy! AI is not a shortcut, you still need to constantly improve your knowledge and skills (from the foundations...)
- Never trust blindly the generated code
- The generated code might be overly complex and difficult to understand, even for simple problems: it's almost never a good solution
- Never forget the non-functional requirements

An Industrial Perspective

The DevOps Research and Assessment (DORA) group of Google Cloud provides very interesting insights about the adoption of AI technologies in software development companies.

An Industrial Perspective

The DevOps Research and Assessment (DORA) group of Google Cloud provides very interesting insights about the adoption of AI technologies in software development companies.

In the 2025 State of AI-assisted Software Development Report, the majority of survey respondents (90%) use AI as part of their work and believe (more than 80%) it has increased their productivity and observed code quality

- yet a notable portion (30%) currently report little to no trust in the code generated by AI, indicating a need for critical validation skills

An Industrial Perspective

The DevOps Research and Assessment (DORA) group of Google Cloud provides very interesting insights about the adoption of AI technologies in software development companies.

In the 2025 State of AI-assisted Software Development Report, the majority of survey respondents (90%) use AI as part of their work and believe (more than 80%) it has increased their productivity and observed code quality

- yet a notable portion (30%) currently report little to no trust in the code generated by AI, indicating a need for critical validation skills

Moreover, the true impact of AI on product development is still unclear:

- in certain cases, despite the positive perceived impact by developers (on productivity, code quality, team performance), AI returned a lot of promising results but also increased software delivery instability (quality and reliability of the software delivery process)
- developers didn't notice differences in terms of friction and burnout in their work

The main reason is that it is not enough to simply integrate AI into software development processes

- they must be carefully **adapted** for that... how?

The main reason is that it is not enough to simply integrate AI into software development processes

- they must be carefully **adapted** for that... how?

Let's see some concrete examples:

- **Code reviews and handoffs:** consider where AI can accelerate and clarify existing steps. For example, AI-generated first-pass reviews can reveal issues quickly and reduce time spent on routine feedback. Structuring AI input to highlight risks or summarize diffs can also make reviews easier and faster for humans.

The main reason is that it is not enough to simply integrate AI into software development processes

- they must be carefully **adapted** for that... how?

Let's see some concrete examples:

- **Code reviews and handoffs:** consider where AI can accelerate and clarify existing steps. For example, AI-generated first-pass reviews can reveal issues quickly and reduce time spent on routine feedback. Structuring AI input to highlight risks or summarize diffs can also make reviews easier and faster for humans.
- **Integration and deployment pipelines:** AI-generated code moves fast—can your systems keep up? CI/CD pipelines may need to evolve to reduce wait states and allow for higher-frequency delivery. Quality checks powered by AI can be layered in without adding manual gates, improving flow without sacrificing assurance.

- **Data infrastructure:** invest in updating data infrastructure. AI’s benefits for productivity and code quality are amplified when AI models and tools are connected to internal data (repos, work tracking tools, documentation, decision logs, etc.). Adding this valuable context improves the output of AI tools, and the benefits are larger when data is AI-accessible, accurate, and complete.

- **Data infrastructure:** invest in updating data infrastructure. AI's benefits for productivity and code quality are amplified when AI models and tools are connected to internal data (repos, work tracking tools, documentation, decision logs, etc.). Adding this valuable context improves the output of AI tools, and the benefits are larger when data is AI-accessible, accurate, and complete.
- **Security and privacy protocols:** with AI now participating in development and operations workflows, security practices must evolve. This includes ensuring secure tool usage, updating policies, and introducing AI-aware monitoring systems that maintain trust without introducing bottlenecks. Automating parts of these processes can help teams maintain speed.

- **Data infrastructure:** invest in updating data infrastructure. AI's benefits for productivity and code quality are amplified when AI models and tools are connected to internal data (repos, work tracking tools, documentation, decision logs, etc.). Adding this valuable context improves the output of AI tools, and the benefits are larger when data is AI-accessible, accurate, and complete.
- **Security and privacy protocols:** with AI now participating in development and operations workflows, security practices must evolve. This includes ensuring secure tool usage, updating policies, and introducing AI-aware monitoring systems that maintain trust without introducing bottlenecks. Automating parts of these processes can help teams maintain speed.
- **Change management and cultural alignment:** like any organizational transformation, AI adoption needs vision, support, and communication:
    - leaders should articulate the long-term goals of AI transformation and support the transition with training, shared practices, and realistic expectations
    - teams need permission to experiment, make mistakes, build fluency, and share what they learn.

  In general, companies should estabilish comprehensive internal protocols for using AI tools.

Beyond augmenting existing systems, AI offers a chance to design new workflows:

- **AI-native delivery pipelines:** AI can continuously analyze code for bugs, security vulnerabilities, and violations of team standards. It can suggest tests and even generate them dynamically.

Beyond augmenting existing systems, AI offers a chance to design new workflows:

- **AI-native delivery pipelines:** AI can continuously analyze code for bugs, security vulnerabilities, and violations of team standards. It can suggest tests and even generate them dynamically.
- **AI-native data systems:** AI can help maintain its own environment by organizing, tagging, cleaning, and analyzing data. This enables more robust insight generation and faster iteration on data-informed decisions.

Beyond augmenting existing systems, AI offers a chance to design new workflows:

- **AI-native delivery pipelines:** AI can continuously analyze code for bugs, security vulnerabilities, and violations of team standards. It can suggest tests and even generate them dynamically.
- **AI-native data systems:** AI can help maintain its own environment by organizing, tagging, cleaning, and analyzing data. This enables more robust insight generation and faster iteration on data-informed decisions.
- **AI-native security:** AI can expand the capacity of security teams by detecting threats earlier, identifying anomalous behavior, and even automating parts of the incident response process.

Beyond augmenting existing systems, AI offers a chance to design new workflows:

- **AI-native delivery pipelines:** AI can continuously analyze code for bugs, security vulnerabilities, and violations of team standards. It can suggest tests and even generate them dynamically.
- **AI-native data systems:** AI can help maintain its own environment by organizing, tagging, cleaning, and analyzing data. This enables more robust insight generation and faster iteration on data-informed decisions.
- **AI-native security:** AI can expand the capacity of security teams by detecting threats earlier, identifying anomalous behavior, and even automating parts of the incident response process.
- **AI-native collaboration models:** emerging practices like agentic workflows and swarming are beginning to reshape how humans and AI work together. Agentic workflows assign tasks to autonomous AI agents, while swarming enables teams and AI to converge dynamically on complex problems.

The future: AI model collapse?

Recent research estimated that more than 50% of web content (especially text and images) is AI-generated - after just a few years since the deployment of generative AI tools.

The future: AI model collapse?

Recent research estimated that more than 50% of web content (especially text and images) is AI-generated - after just a few years since the deployment of generative AI tools.

As those models are trained on large-scale datasets scraped from the web, researchers investigated what happens when training data is recursively AI-generated.

The future: AI model collapse?

Recent research estimated that more than 50% of web content (especially text and images) is AI-generated - after just a few years since the deployment of generative AI tools.

As those models are trained on large-scale datasets scraped from the web, researchers investigated what happens when training data is recursively AI-generated.

The results have shown that their performance gradually degraded, a phenomenon known as **model collapse**

- There is still debate on its real-world impact
- Research is already developing possible mitigation strategies

We'll see what will happen in the future...