Scientific Research Publishing

Journal of Computer and Communications, 2026, 14(2), 153-182
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# From Simple Apps to Systems: A Practical Framework for AI-Driven Software Development

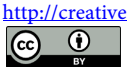
**Eliel Mathe[1], Benedict Mukanirwa[2]**

[1]Nfinic, Butembo, Democratic Republic of Congo
[2]Pikelo, Abidjan, Côte d'Ivoire
Email: elielmathe@nfinic.com, ben@pikelo.co

**How to cite this paper:** Mathe, E. and Mukanirwa, B. (2026) From Simple Apps to Systems: A Practical Framework for AI-Driven Software Development. *Journal of Computer and Communications*, **14**, 153-182.
https://doi.org/10.4236/jcc.2026.142008

**Received:** December 8, 2025
**Accepted:** February 25, 2026
**Published:** February 28, 2026







## Abstract

This article examines the integration of artificial intelligence (AI) tools into software development, with a focus on optimizing workflows for computing professionals. Amid the rapid evolution of AI technologies, we review recent innovations—notably large language models (LLMs), neural networks, and transformers—along with their practical applications. We propose a structured methodology for incorporating these tools into software projects, addressing two primary challenges: managing costs associated with token consumption and enhancing context to reduce errors, commonly referred to as "*hallucinations*". Approaches such as Retrieval-Augmented Generation (RAG) and the Model Context Protocol (MCP) are detailed for optimizing interactions with LLMs. Intended for developers, researchers, and novices alike, this work provides an overview of available tools and best practices for building robust, efficient applications, while emphasizing the importance of standardized project structures and team collaboration.

## Keywords

Artificial Intelligence, Vibe Coding, Agentic AI, Software Development

## 1. Introduction

In an era where artificial intelligence (AI) tools are evolving at an unprecedented pace, computing professionals struggle to keep pace with the growing complexity of these technologies. Yet the targeted adoption of AI tools can significantly streamline

workflows, boosting efficiency and shortening delivery timelines. However, the costs associated with these tools and the occasional inaccuracy of their outputs often deter professionals from fully transitioning to AI-integrated workflows.

This article surveys key AI innovations from the past two years and presents a practical methodology for incorporating them into software projects, from prototypes to full-scale applications. Our approach targets two major challenges: managing the operational costs of AI tools and organizing context to improve result accuracy and minimize hallucinations in software development.

Given the vast scope of software development, this article offers a concise overview while directing readers to references for deeper exploration. It is aimed primarily at software development professionals and anyone seeking to build applications leveraging AI tools. Computer scientists, educators, and beginners will also find valuable insights.

This work does not focus on creating bespoke AI tools but highlights market-available solutions for developing software applications efficiently.

## 2. Context and Literature Review

### 2.1. Machine Learning and Artificial Intelligence

Since late 2022, a transformative technology—**ChatGPT**—has permeated nearly every aspect of daily life. Although it emerged at that time, ChatGPT is the culmination of a long lineage of technological advances. This section reviews two of its closest predecessors: machine learning and artificial neural networks.

#### 2.1.1. Machine Learning

Machine learning is a set of techniques that enable machines to mimic human learning capabilities [1]. Algorithms are typically designed for classification or prediction and incorporate an error function to evaluate prediction accuracy against ground-truth examples. An iterative optimization process then minimizes the discrepancy between prediction and reality.

Unlike traditional algorithms with explicitly defined execution steps, machine learning relies on data to produce outcomes. Greater data volume generally yields better predictions.

$$y = Ax + B$$

A basic linear function in machine learning incorporates input features $x$ (often vectorial), weights $A$ (coefficients or parameters), bias $B$, and the model's prediction $y$. Andrew Ng discussed this in his 2018 lecture [2]. The survey "A Survey of Optimization Methods from a Machine Learning Perspective" by Shiliang Sun et al. provides a comprehensive overview of optimization techniques [3].

#### 2.1.2. Artificial Neural Networks

Artificial neural networks, as shown in **Figure 1**, are inspired by the human brain, forming a subclass of machine learning known as deep learning [4]. They consist of interconnected nodes (neurons) organized in layered structures analogous to

biological neural architecture[1].

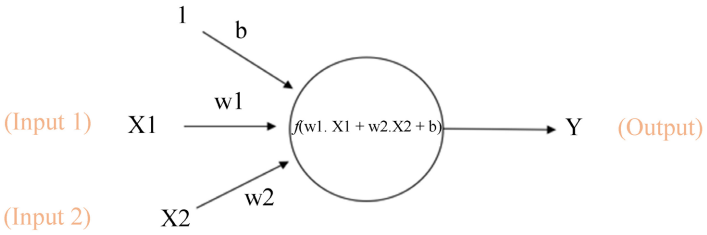



**Figure 1.** Representation of a neuron [5].

A forward pass through a neural network involves a neuron or node. It receives inputs from nodes in the previous layer (or subsequent layers, depending on the propagation method) and produces an output.

In a neural network (**Figure 2**), the more layers you have, the better the accuracy of the results produced.

Tools such as TensorFlow, developed by Google Brain, enable the creation and inference of models for diverse tasks, supporting execution from edge devices to the cloud[2].

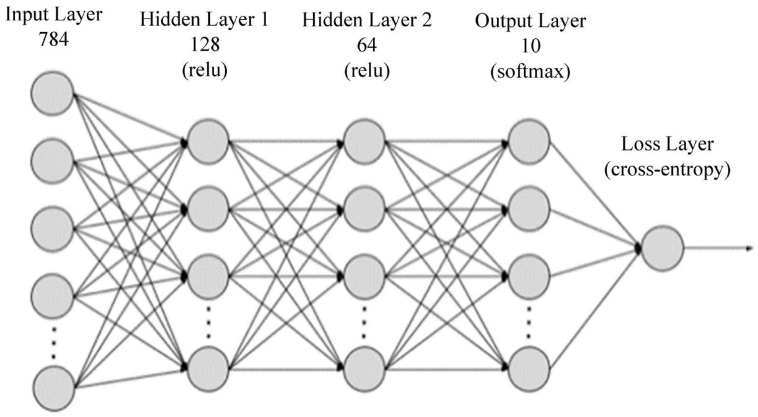



**Figure 2.** Representation of a neural network [6].

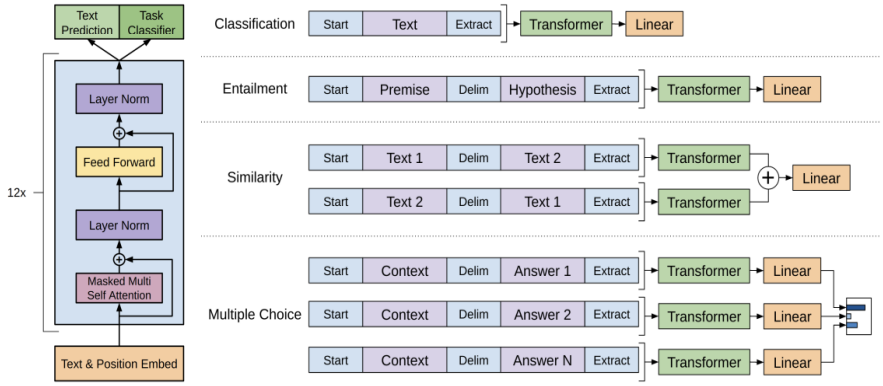
### 2.1.3. Transformers

Transformers (**Figure 3**) are a specific artificial intelligence architecture devel-

[1]A neural network is an artificial intelligence method that teaches computers to process data in a way inspired by the human brain. It is a type of machine learning process called deep learning, which uses interconnected nodes, or neurons, in a multi-layered structure similar to the human brain. It creates an adaptive system used by computers to learn from their mistakes and continuously improve. Artificial neural networks attempt to solve complex problems such as summarizing documents or recognizing faces, with greater accuracy [6].

[2]Edge: the edge of the Internet network, often devices like phones or laptops. Cloud: the other side of the Internet network, often consisting of servers.

oped by Google [7] based on the sequence-to-sequence (Seq2Seq) technique [8]. This approach is particularly effective for natural language processing because it allows models—for example, LSTMs—to assign meaning to a sequence and to remember or forget parts of it depending on their assessed importance [9]. To make Transformers more effective at understanding long-range dependencies, they are equipped with an attention mechanism [7]. GPT, for instance, is a model that leverages this technology for content generation.




**(Left)** Transformer architecture and training objectives used in this work. **(Right)** Input transformations for fine-tuning on various downstream tasks. All structured inputs are converted into token sequences so they can be processed by the pre-trained model, followed by a linear layer + softmax [10].

**Figure 3.** Transformer architecture [11].

After training, the result is a model. When performing inference with the model, it produces an output that can be text, an image, a video, or any other form of data.

### 2.1.4. Survey of a Few Models

The following classification of models was derived through interaction with ChatGPT[3]:

**1) Traditional Machine Learning**

- *Linear Regression* predicts continuous numerical values. For example, it estimates a house price based on its size: a 100 $m^2$ house might thus be valued at $250,000.
- *Logistic Regression* produces a binary decision (yes/no) from input data. For example, given 90% humidity and low atmospheric pressure, it predicts with 78% probability that it will rain today—so the answer is yes.
- *Decision Trees and Random Forests* build tree-structured rules for classification or prediction. For example, a simple rule might be: "If age > 30 years and income > $50,000, then approve the loan."
- *Support Vector Machines* (*SVMs*) identify the optimal boundary that separates two classes of data. For example, they draw a line (or hyperplane) that

[3]The key prompt was: "Classify the types of AI models by providing an example for each type of model".

perfectly distinguishes images of cats (red points) from images of dogs (blue points), with the largest possible margin.

- *The K-Means* algorithm automatically groups similar data points without prior labels (unsupervised clustering). For example, it can segment a store's customers into three groups: low spenders, average spenders, and high spenders, to tailor marketing campaigns accordingly.

**2) Artificial Neural Network Models**

- *Feedforward Networks* constitute the most basic neural networks, ideal for simple classification or prediction tasks without complex sequences. For example, they can categorize emails into labels such as "urgent", "important", "secondary", or "spam" based solely on textual content analysis.
- *Convolutional Neural Networks* (*CNNs*) excel at image processing and computer vision by automatically extracting patterns such as edges or shapes. For example, they recognize handwritten digits in an image, classifying a handdrawn "5" with high accuracy thanks to convolutional filters.
- *Recurrent Neural Networks* (*RNNs*) and LSTMs are particularly well-suited to sequential data such as text or speech, as they maintain memory of previous elements. For example, they predict the next word in a sentence like "The cat is on the…" by generating "mat", taking prior context into account.
- *Transformers* represent a powerful advance in natural language processing, efficiently handling long-range dependencies without recurrence. For example, the power tools like ChatGPT can translate an entire sentence from English to French while capturing overall meaning, such as "Hello, how are you?" into "Bonjour, comment allez-vous?".

**3) Generative Models**

- *Generative Adversarial Networks* (*GANs*) pit two antagonistic networks against each other—a generator and a discriminator—to create realistic content such as images, music, or videos, iteratively improving quality through competition. For example, models like GANSynth use GANs to compose music by independently controlling pitch and timbre, transforming a simple latent vector into a coherent instrumental melody, such as a synthesized violin solo derived from notes in a dataset like NSynth.
- *Diffusion Models*, such as Stable Diffusion or DALL·E, generate realistic images by starting from random noise and progressively refining it toward a target, guided by textual prompts for conditional synthesis. For example, given the prompt "an armchair shaped like an avocado", Stable Diffusion produces a detailed image of a surreal piece of furniture with textured green tones by inverting a diffusion process to faithfully capture the described concept.

## 2.2. Interaction with LLMs: Prompt Engineering

Large language models (LLMs) introduce a novel way of interacting with computers using natural language. Thanks to technologies such as GPT-3, computers can now understand and process text entered in everyday language.

The input text is called a "prompt[4]"; the generated response can take various forms: plain text, JSON, source code, images, videos, etc. From these interactions has emerged a new discipline: prompt engineering[5].

*Example*:

*Prompt*: "*Write a poem about the sunset in Kipriani*"

*Result*: "*In Kipriani, as the day withdraws*

*Each ray becomes promise and smile.*

*The sun falls asleep, yet the soul awakens…*"

*Excerpt from an interaction with GPT*-5

When interacting with LLMs, the main constraints are token volume (the number of characters entered as input and received as output from the model) and context (the information provided to the model). The better the context supplied, the higher the quality of the model's responses; conversely, the more text entered, the more expensive the interaction becomes, as illustrated in **Figure 4**.

Advanced prompts can be crafted to obtain more sophisticated results. Matt Shumer, for instance, shares prompts for advanced LLM usage on his website. An example prompt for creating an agent capable of growing a startup is provided in the appendix [12].

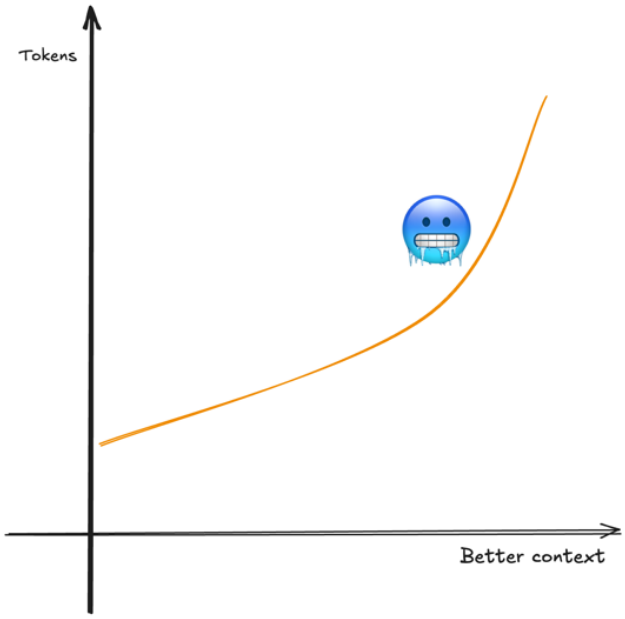



**Figure 4.** The number of tokens often grows with context improvement.

### 2.2.1. Vibe Coding

Vibe coding, a term coined by AI researcher Andrej Karpathy in early 2025, is an

[4]Prompt: giving an instruction to artificial intelligence (=a computer system or machine that possesses certain qualities of the human brain, such as the ability to interpret and produce language in a way that seems human, to recognize or create images, to solve problems and to learn from data provided to it) using natural language rather than computer language https://dictionary.cambridge.org/dictionary/english/prompt.

[5]You can go to https://platform.openai.com/docs/guides/prompt-engineering to learn more.

emerging software development technique that utilises artificial intelligence to generate functioning code from natural language prompts [13]. Vibe coding accelerates software development and makes app development accessible to people with less software development skills.

### 2.2.2. Agent Mode

Agentic AI enables the use of LLMs with far less human supervision [13]. Tools pass their results to other tools or agents, which in turn generate new outputs, iteratively, with each component having a clearly defined role.

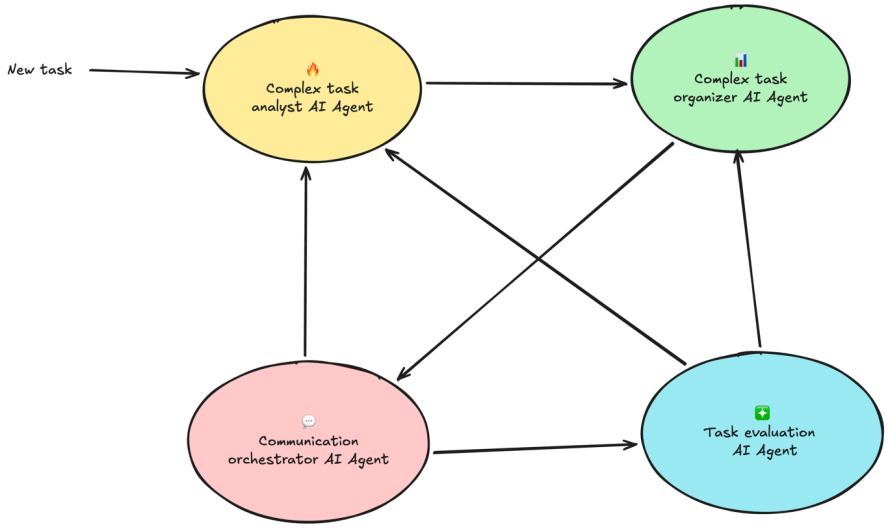



**Figure 5.** Collective work of AI.

Imagine a system composed of several specialized agents (**Figure 5**):

- one agent expert in analyzing complex problems,
- another specialized in creating structured tasks,
- a third dedicated to drafting clear and professional emails,
- and a final one responsible for evaluating the quality of contributions from all the others.

Simply submit the overall task to the group of agents, and you obtain results that are more accurate, coherent, and optimized.

However, these architectures cause token[6] consumption to explode: usage jumps from a few thousand tokens per request to several million tokens within a single interaction.

### 2.2.3. The Context

Context plays an essential role in interactions with LLMs: the richer it is, the better the results. When an AI is integrated into an application, continuous interaction with the user's data becomes essential in order to provide precise context for each exchange. Several protocols and frameworks have therefore been developed to op-

[6]The token is the unit of measurement for interactions with LLMs: it represents the text entered or the text output from an interaction with the model in most cases.

timize context management and streamline interactions: RAG, LangChain, LangGraph, Google Vertex AI, Cap'n Proto, MCP, among others [14].

In the following sections, we will review some of these frameworks.

**1) Retrieval-Augmented Generation (RAG)**

Retrieval-Augmented Generation (RAG) is an approach that combines two major strengths of artificial intelligence: information retrieval and text generation. Instead of relying solely on the internal knowledge of a language model (e.g., GPT-4), RAG actively fetches relevant information from a database, documents, or any other corpus and then uses it to produce a more accurate and up-to-date response. This mechanism overcomes the limitations of pre-trained models, which do not always have access to the most recent or domain-specific data.

LLMs possess powerful language understanding and generation capabilities, but they are not reliable sources of factual information and lack access to proprietary data or any information not included in their training set. They are also prone to "hallucinations", meaning they may fabricate answers rather than admit ignorance.

Using RAG with an LLM addresses several of these issues. By supplying the model with all the information it needs to answer a question, it can respond accurately on topics outside its original training data and significantly reduce the likelihood of hallucinations.

**2) Model Context Protocol (MCP)**

The Model Context Protocol (MCP) is a protocol that standardizes the way applications provide context to LLMs using JSON-RPC [15]. It can be thought of as the USB-C of AI: a universal standard that enables seamless connection between devices and peripherals [15].

Several projects and companies now offer an MCP interface that can be directly connected—for example, within development environments such as VSCode[7]—to enrich the context for their new APIs or tools[8].

## 3. Methodology

The use of LLMs in the software development process is evolving rapidly. Traditional stages—requirements analysis, development, testing, and deployment [16]—are now largely supported by a wide variety of AI tools capable of generating applications ranging from the simplest to the most complex. While many developers already leverage these tools for rapid prototyping, they often hesitate to apply them at scale due to concerns over loss of control or reliability.

AI agents are a game-changer, enabling structured and accelerated design of complete projects. Our methodology provides proven techniques for extracting maximum value from LLMs throughout the entire development lifecycle. The goal is to equip code owners and teams with the means to integrate AI effectively—

[7]VSCode: a software development environment available at https://code.visualstudio.com.

[8]OpenZeppelin, a company specializing in providing secure smart contracts on blockchain, now offers an MCP server: https://mcp.openzeppelin.com. Similarly, Flutter, a framework dedicated to creating mobile applications, has its own MCP server: https://dart.dev/tools/mcp-server.

even on increasingly large-scale projects—without sacrificing quality or final product control.

The classic developer stack typically consists of the following components (Figure 6):

- User interface language: e.g., HTML.
- Server-side language: e.g., Node.js.
- Database: e.g., MySQL.
- Supporting tools: Docker, GitHub, etc.

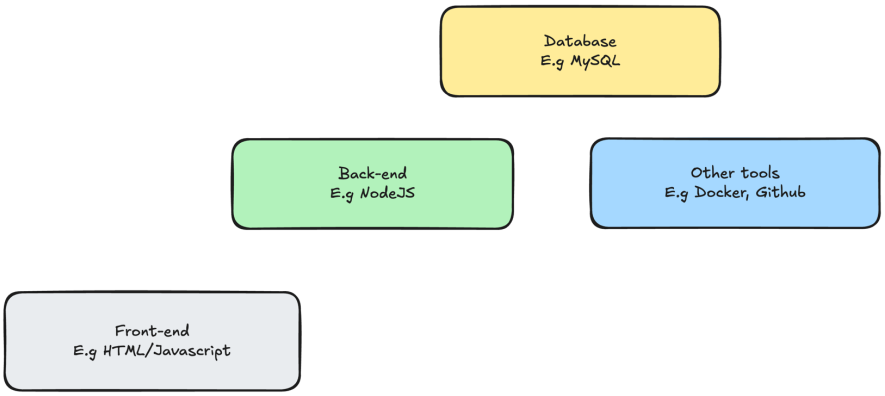



Figure 6. Classic software developer stack.

As the application grows in complexity (Figure 7), the stack expands to include:

- Advanced frontend frameworks for the user interface
- Backend distributed as microservices, each potentially written in a different language
- Integration of external APIs
- Diversified database system: central database, domain-specific databases, caches, or optimized temporary databases
- Load balancers
- Firewalls and security systems

Regardless of the system's size, all components must communicate efficiently with one another and with their environment. This relies on dedicated programming languages and standardized exchange formats [17].

AI introduces novel tools that, when applied to software development, significantly accelerate and optimize the practice of software engineering. While certain vibe coding platforms claim to construct complete systems [18] from a single prompt or a small number of prompts, this approach rapidly becomes impractical as the system grows in terms of modules, components, and overall complexity. At that scale, sustainable development requires not only adherence to established software engineering best practices, but their deliberate coupling with the AI-augmented methodologies and tools presented in this article. This combined approach—traditional engineering discipline reinforced by targeted AI capabilities—is essential to effectively manage complexity, ensure long-term maintainability, and support controlled, high-velocity scaling.

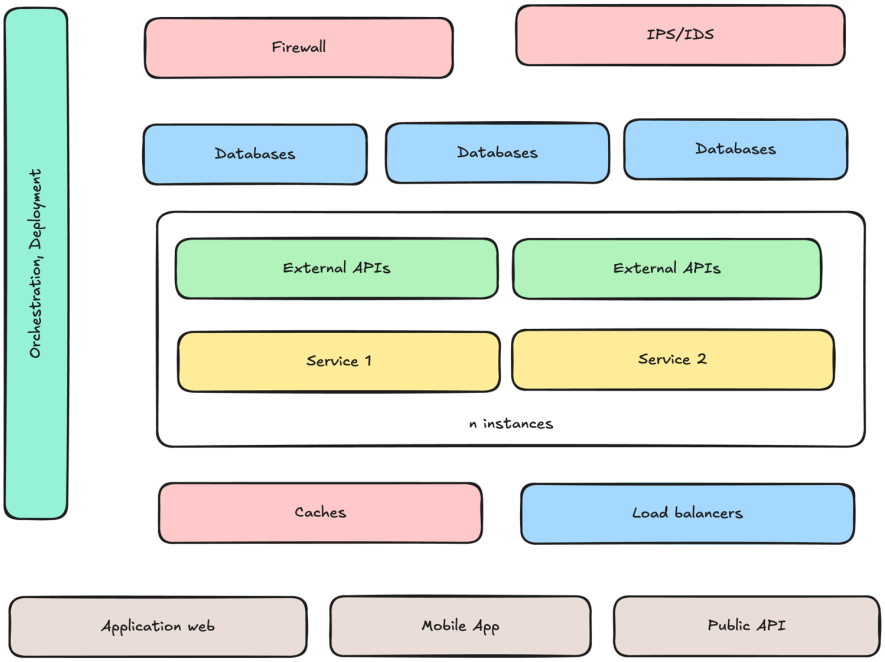



**Figure 7.** Architecture of a complex application.

In the following section, a non-exhaustive list of AI-integrated tools is presented to assist developers in adopting an updated software development methodology that fully leverages the advantages offered by LLMs. The tools discussed are positioned within a development process that remains compatible with agile methodologies and focuses primarily on three core areas: software design, software development (coding), database development and system deployment. At the end of the section, selected aspects of software security—both at the source code level and during execution—are addressed in the context of systems developed with the assistance of LLMs.

## 3.1. Overview of Tools for Software Development Using AI

### 3.1.1. Software Design

Design is a pivotal stage in the software development lifecycle, encompassing all processes carried out before actual coding begins. To ensure robust outcomes, certain tasks must remain under strict human control—particularly market research and direct client discussions.

Once these phases are complete, AI tools can dramatically accelerate the production of presentation documents, mockups, and technical specifications. Care must still be taken to minimize the risks of hallucinations or distortion of critical information.

In practice, many teams adopt agile methodologies, long recognized as highly effective, with SCRUM being the most prominent. Using AI tools does not change the fundamentals: smooth human collaboration remains essential to turn an idea into a finished product. All prompts used should therefore be centralized in a shared folder accessible to every team member, ensuring transparency and con-

sistency in interactions with the model. In Table 1, a few tools are provided for software designing.

Table 1. AI tools for software designing.

| Tool name | URL | Description |
|---|---|---|
| ChatGPT, Grok | https://chat.com https://grok.com | Conversational AI assistant for generating design ideas, textual wireframes, or functional specifications during the software planning phase. |
| KeepAPrompt | https://keepaprompt.com | Platform for storing, sharing, and discovering optimized AI prompts, enabling the creation of reusable prompts to guide software design tools. |
| Figma | https://figma.com | Collaborative UI/UX design tool with AI features that automatically generate prototypes and interfaces from textual descriptions. |
| Canva | https://canva.com | AI-assisted visual creation platform for rapidly designing mockups, diagrams, and graphical assets intended for the software prototyping phase. |

### 3.1.2. Software Development

Software development has taken a giant leap forward thanks to advances in artificial intelligence tools. We have moved from simple code completions to fully autonomous agents capable of implementing complete features in just a few interactions.

As previously emphasized, rigorous project organization is essential: it facilitates team access and collaboration while adhering to best practices to ensure extensibility and a clear structure. It is important to adopt well-established frameworks, as well as mastering code interaction through automated testing, code reviews, and continuous deployment tools. Table 2 provides non exhaustive list of software development tools.

Table 2. Software development AI tools.

| Tool name | URL | Description |
|---|---|---|
| Github | https://github.com | With GitHub Copilot + Copilot Agent: enable AI-assisted code generation, structured project organization, and automated testing, allowing robust applications to be built with minimal supervision. Tasks can be assigned directly to Copilot Agent, which autonomously creates pull requests. |
| Bolt, Lovable, Firebase Studio | https://bolt.new, https://lovable.com, https://firebase.studio | AI tools for rapid full-stack application development. These tools exhibit distinct characteristics tailored to different use cases. For instance, Firebase integrates seamlessly with the Google ecosystem. |
| n8n | https://n8n.com | Low-code/no-code workflow automation platform that integrates artificial intelligence to connect applications, orchestrate complex processes, and automate repetitive tasks within the software development lifecycle. |

**Continued**

| | | |
|---|---|---|
| VSCode, Cursor | https://vscode.com, https://cursor.com | AI-Integrated Development Environments: VS Code stands out for its exceptional extensibility and strong support for collaboration. Cursor, an advanced fork of VS Code, provides native AI assistance, delivering powerful capabilities for code autocompletion, debugging, and multi-file editing directly through natural language instructions. |

### 3.1.3. Automated and AI-Assisted Databases

Databases are a fundamental component of any information system, as they enable structured storage and organization of data. Creating one involves the following successive steps: analyzing requirements, designing the conceptual data model (CDM), then the logical data model (LDM), and finally implementing the physical data model (PDM) in the chosen DBMS[9] [19]. Although numerous methods and tools have been developed for database design, their implementation remains a true art: the more experience one gains, the better one becomes at creating performant, elegant, and perfectly optimized databases.

AI provides developers with powerful tools to design and manage databases more easily. While it is possible to discuss with ChatGPT, Claude, or Grok to create or improve a database, the most effective solution is to connect the application directly to the database using existing technologies, particularly ORMs[10] and BaaS[11]. The bad approach that is not AI friendly is to have the DBMS completely separated from the source code, as shown in Figure 8.

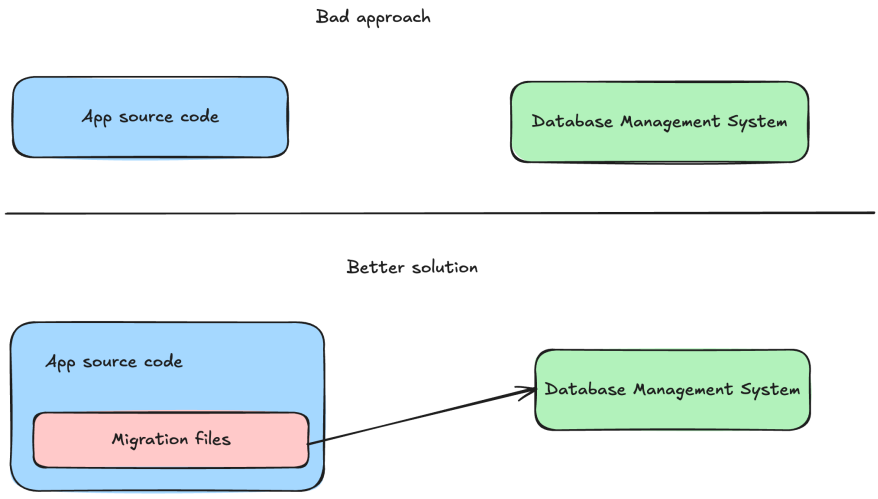



Figure 8. Use of migration files in the application source code.

When developing a backend with a framework that includes an ORM, AI agents can easily modify the database structure and automatically generate the corre-

[9]DBMS: Database Management System.
[10]ORM: Object-Relational Mapping.
[11]BaaS: Backend as a Service.

sponding migrations.

By using BaaS solutions, it becomes possible to develop and fully manage the backend directly from the frontend source code. Although this approach is fast and efficient, it can introduce significant security risks if best practices are not strictly followed.

Currently, the vast majority of backend frameworks support ORMs for SQL databases (MySQL, PostgreSQL, Oracle, etc.) as well as NoSQL databases (e.g., MongoDB). Similarly, cache databases (Redis, Memcached, etc.) are generally very easy to manipulate from application code thanks to the libraries provided by most frameworks.

One of the major points of friction in software development is writing, optimizing, and maintaining database queries (SQL, NoSQL, etc.). Integrating AI at the data-access layer offers a radical solution to streamline developers' work and democratize data access and manipulation for less technical profiles.

Today, developers no longer need to write complex data queries to interact with their database. They can simply express their request to the AI in natural language (e.g., "Find the 10 most active users in September with a satisfaction score above 90"). An MCP-compatible client forwards this request to the dedicated MCP server for the database. The MCP server translates the request into a native query (SQL or JSON, for example), executes it, and returns the formatted result to the AI, which then presents it clearly to the user.

This approach provides:

- Reduced context switching: developers can query or even modify the database directly from their development environment (IDE) without resorting to external consoles.
- Democratization: it allows non-developer team members to explore data using natural language.

Suggested BaaS (Backend-as-a-Service) platforms and databases are shown in Table 3.

Table 3. BaaS platforms and Database tools.

| Tool name | URL | Description |
|---|---|---|
| Supabase | https://supabase.com/ | An open-source BaaS platform (PostgreSQL) with community-maintained MCP servers available [22]. |
| Firebase | https://firebase.google.com/ | A non-open-source BaaS platform (NoSQL) offering seamless integration with the Google ecosystem, high maturity, and official MCP servers via Google Cloud. |
| Convex | https://www.convex.dev/ | A BaaS platform (Functions & Real-time) focused on collaborative and real-time applications, featuring a simplified API for AI integration. |
| DB-Mcp-Tool | https://github.com/wirdes/db-mcp-tool | Universal MCP server that connects existing databases to AI agents, making any backend "AI-Agent Ready" without requiring migration. |

### 3.1.4. Software Deployment

Artificial intelligence has revolutionized software deployment by making processes smoother and more automated. The field of continuous integration/continuous deployment (CI/CD), although broad and technical, benefits enormously from LLMs, which accelerate the design and optimization of deployment pipelines.

For example, tools such as Grok or ChatGPT can quickly generate Dockerfiles tailored to your application, or Ansible scripts to orchestrate multi-environment deployments. These AI assistants convert natural-language descriptions into ready-to-use configurations, thereby reducing manual errors and time-to-production. A list of tools for deployment is provided in Table 4.

Table 4. Deployment tools.

| Tool name | URL | Description |
|---|---|---|
| Grok, ChatGPT | https://grok.com, https://chat.com | Conversational AI assistant for generating CI/CD scripts, Docker configurations, or automated pipelines directly from natural-language descriptions. |
| Termius | https://termius.com | AI-integrated terminal for interacting with remote servers, especially Linux servers. |
| Dotenvx | https://dotenvx.com | Secure environment-variable management tool with built-in encryption, enabling versioning, injection, and auditing of secrets directly within deployment pipelines. |
| Harness CircleCI Github Actions | https://www.harness.io/ https://circleci.com/ https://github.com | AI-driven CI/CD platform for deployment checks, canary releases, and intelligent rollbacks, minimizing production downtime. |

### 3.1.5. Software and Source-Code Security

Security is a particularly sensitive domain in the development and operation of software products. It encompasses many aspects, ranging from program logic (e.g., flow validation) to user protection (authentication, role management), input-field hardening (against SQL injection or XSS) [20] [21], protection of sensitive data, and securing underlying infrastructure (networks, containers, etc.).

The integration of artificial intelligence (AI) tools, especially large language models (LLMs), opens unprecedented opportunities to strengthen security—for example, by automating vulnerability detection, generating code audits, or simulating attacks to test system resilience.

However, these tools, while powerful, are not infallible and introduce their own risks. We strongly advise against *fully delegating security responsibility to AI*, because even a minor error—such as a model hallucination producing vulnerable code or a biased recommendation—can lead to irreversible consequences: massive data breaches, financial losses, reputational damage, or regulatory violations[12]. Studies

[12]Regulatory violations such as GDPR or cybersecurity law in the United States.

such as the OWASP Top 10 for LLMs [22] highlight specific threats, including training-data poisoning, malicious prompt injection, and adversarial attacks that manipulate AI outputs. Moreover, code generated by LLMs can inherit hidden biases or vulnerabilities, increasing risks throughout the software supply chain.

Despite these pitfalls, AI remains a valuable ally when used in a hybrid, human-supervised manner. It can provide actionable insights, such as automated log analysis for anomaly detection or generation of tailored encryption scripts. To maximize benefits while minimizing dangers, adopt a "Secure by Design" approach that integrates AI from the earliest stages of the software lifecycle.

AI is transforming software security into a more dynamic and preventive discipline, but it demands heightened vigilance and robust human governance. By balancing innovation with caution, organizations can harness its potential without compromising the integrity of their software products.

**Table 5.** Security tools powered by AI.

| Tool name | URL | Description |
|---|---|---|
| Checkmarx | https://checkmarx.com/ | An application security platform that helps developers find and fix security vulnerabilities in source code before deployment. It offers SAST tools and seamless integration into CI/CD pipelines. |
| Aikido Security | https://www.aikido.dev/ | SAST solution with AI-powered AutoFix that automatically identifies and corrects vulnerabilities in code, prioritizing risks based on project context to speed up security reviews with minimal manual intervention. |
| CodeQL (GitHub) | https://codeql.github.com/ | GitHub's AI-powered code analysis engine that runs semantic queries on source code to detect malicious patterns and security bugs, with native repository integration for automated scans and real-time alerts. |

Although we have recommended these tools, security is a constantly evolving field. Before using any of them, ensure you conduct thorough research to confirm that the tool fully meets your own security standards (**Table 5**).

When using AI tools, we favor natural language over traditional programming languages. However, as discussed in previous chapters, two major constraints remain: context management and token volume limitation.

## 3.2. Optimizing AI Tool Usage in Large-Scale Software Projects

### 3.2.1. Token Consumption Optimization for Cost Reduction

When interacting with LLMs, text is used as input, and they generally return text as output. To measure[13] the amount of input text, cached memory, and output text, the quantity is typically calculated in tokens; nowadays, this is commonly

[13]The millions of tokens: the invisible unit of measurement shaping modern AI https://daijobu.ai/fr/2025/05/19/les-millions-de-tokens-lunite-de-mesure-invisible-qui-faconne-lia-moderne/.

measured in millions of tokens, as illustrated in Figure 9.

Model response quality is strongly correlated with the relevance and precision of the provided context, although this must be balanced against token and cost constraints. For example, when interacting with Grok[14] to modify a file, the file should either be attached to the conversation or its content copied directly into the chat interface. This allows the LLM to fully understand the document and make the necessary changes.


Figure 9. Price of the GPT-realtime model on the OpenAI website on January 13, 2026.

If the modification requires taking another file into account—particularly in the case of imports or dependencies—all relevant files must be attached, and their relative locations explicitly specified (e.g., "the file 'main.py' is located in the '/src' folder, and utils.py is in '/lib'"). This clarification ensures the model correctly interprets the relationships between modules.

Adding these files mechanically increases the token count in direct proportion to the size of the provided context.

The use of artificial intelligence agents dramatically amplifies token consumption. It therefore becomes critical to optimize their usage, especially on platforms where tokens are billed or when developing an autonomous agent. The following section reviews several effective optimization techniques.

**1) Project Discoverability**

To facilitate project discovery by artificial intelligence tools, it is recommended to adopt a structure based on templates that are widely used across the Internet. Rather than inventing a custom organization, we advocate the use of standardized project scaffolding tools that guarantee immediate recognition by LLMs.

Examples include:

- For full-stack web projects [23] with a Node.js backend, Volo[15] provides a classic, well-documented structure.
- For full-stack web projects with a Python backend, django-admin *startproject* (for Django) or *cookiecutter* https://github.com/tiangolo/full-stack (for FastAPI and Flask).
- For full-stack web projects with a PHP backend, "*composer create-project laravel/laravel*" (for Laravel) or php artisan new (within the Laravel ecosystem).

Overall, popular frameworks [18] offer far superior discoverability compared to highly customized, bespoke architectures. In the latter case, simply adding descriptive context is sufficient to guide the AI tool through navigation of the project files.

**2) Well-Structured Modules**

Project structure plays a critical role for both human collaborators and the

[14]Grok: https://grok.com/, https://fr.wikipedia.org/wiki/Grok (IA).
[15]https://github.com/volojs/volo.

artificial intelligence tools that assist during development. A clear and consistent organization greatly facilitates comprehension, maintenance, and automation.

In this regard, frameworks represent a proven solution: they enforce widely recognized structural conventions, making them particularly well-suited for interaction with LLMs. Frameworks are generally well-suited to LLM-assisted development because they enforce widely recognized structural conventions.

Beyond the choice of framework, proper naming conventions are essential. Prefer folder and file names in English, following conventions broadly adopted in the ecosystem (e.g., "src/", "components/", "services/", "utils/", "config/"). This practice ensures immediate readability for both developers and AI models.

**3) Long Files vs. Multiple Short Files**

Since many AI tools include entire files in the context, we strongly recommend favoring short files over long ones.

A short file dedicated to a single responsibility (e.g., one function or one class) fits far more efficiently into the model's context window. Conversely, a large file that bundles multiple responsibilities consumes more tokens and dilutes the relevant context, making the LLM's analysis less accurate.

Empirical experience suggests that decomposing large files into smaller, single-responsibility units improves both context efficiency and the reliability of LLM-assisted analysis. This practice optimizes both token consumption and the quality of AI-generated responses.

**4) Batching Requests**

Some AI tools charge based not on token count but on the number of requests. This is the case, for example, with GitHub Copilot Agent, which introduces "premium requests[16]": every task submitted to the agent—whether a full code-generation session or an iterative edit—counts as one such premium request.

To minimize costs, it is advisable to bundle multiple sub-tasks into a single overarching request. However, the larger the batch of questions, the less fine-grained control you have over the resulting pull request generated by Copilot. This can lead to increased code-review time or to model hallucinations, making the outputs less reliable. Figure 10 summarizes these practises.

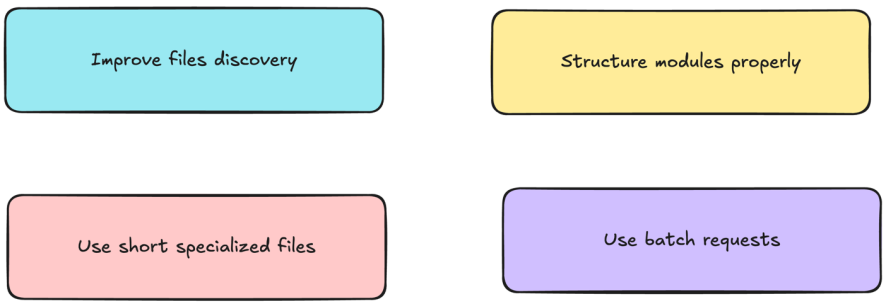



Figure 10. Token reduction summary.

[16]You can read more about "Premium request" here https://docs.github.com/en/copilot/concepts/billing/copilot-requests.

### 3.2.2. Context Enhancement and Automation

AI models are primarily trained on public datasets that are fixed in time. However, many companies and individuals possess private data that is absent from these training corpora. Consequently, at each interaction, it is necessary to supply this specific information in the prompt to enrich the context, guide responses, and prevent errors stemming from the lack of internal knowledge.

This practice significantly improves the relevance of the model's outputs by compensating for its limited access to confidential or up-to-date data.

In this section, we present methods for enriching context so that AI tools can produce the best possible results.

**1) Enhancing Context with a RAG System**

As the system scales, incorporating all available data into a single prompt to reduce LLM hallucinations becomes increasingly impractical. To achieve richer and more effective context despite large data volumes, a Retrieval-Augmented Generation (RAG) system built on a vector database provides a well-optimized and scalable solution.

To create a RAG-based application (Figure 11), the first step is to assemble a high-quality, relevant corpus of data. Next, an efficient retrieval system must be implemented that indexes the content and enables rapid retrieval of key passages [24].

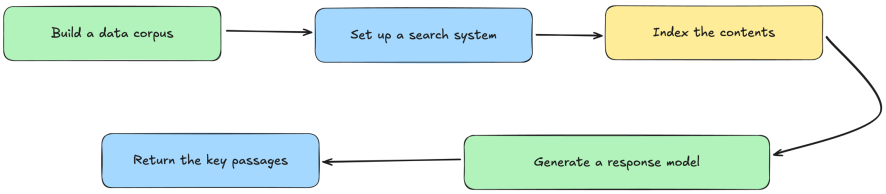



Figure 11. Steps in creating a RAG.

To improve context in your RAG-based application, follow these key steps:

**(1) *Data Collection and Preparation***

The data will serve as the foundation for the model to retrieve information and generate accurate, contextualized responses. It is therefore crucial to organize and clean this information before integrating it into the system.

This involves:

- Gathering documents (articles, reports, internal databases, FAQs, etc.).
- Cleaning the data: removing duplicates, standardizing formats, and eliminating noise.
- Structuring the data: transforming it into usable text formats (JSON, TXT, CSV, or a document database).

**(2) *Indexing and Vectorization***

Once the exploitable text is available in formats such as JSON, TXT, CSV, or a document database, the next steps are :

- Converting the documents into vector representations using semantic embedding models (e.g., OpenAI embeddings or Sentence-BERT [25]).

- Storing these vectors in a vector database such as FAISS, Pinecone, Weaviate, or Milvus.

**(3)** ***Search and Retrieval***

This is the stage where the application user interacts. When the user asks a question, that query is also converted into a vector. The RAG-powered system then performs a similarity search (using cosine similarity [26], Euclidean distance, etc.) to identify the most relevant passages from the document corpus serving as context.

**(4)** ***Augmentated Generation***

The passages retrieved in the previous step are provided as context to the language model (e.g., Claude 4.5, GPT-5, etc.), which then generates a more precise response grounded in the organization's data.

**(5)** ***Evaluation and Improvement***

It is essential to systematically evaluate the results produced by the system to ensure their accuracy and to identify necessary improvements for achieving optimal performance.

Here are some best practices:

- Regularly update the corpus so the system remains relevant.
- Manage confidentiality: encrypt sensitive data and enforce strict access controls.

**(6)** ***Limitations and Failure Modes***

While RAG significantly reduces hallucinations, it remains sensitive to retrieval quality. A common failure mode is irrelevant or weakly related document retrieval, which can bias the model toward incorrect answers while maintaining high linguistic confidence. This issue is exacerbated when embeddings poorly capture domain-specific semantics or when document chunking is misaligned with user queries. Developers can mitigate these risks by combining semantic retrieval with metadata-based filtering, enforcing minimum similarity thresholds, and periodically evaluating retrieval precision using curated test queries. Additionally, fallback mechanisms, such as explicitly allowing the model to answer "insufficient information", can reduce overconfident but incorrect outputs.

**2) Rules/Directives in the Codebase**

When developing within an IDE connected to a code repository, it is critical to clearly define what the LLM is allowed and not allowed to do. This prevents costly hallucinations that waste time and reduce reliability.

Rules address this need precisely: they are configuration files embedded in the project that contain explicit instructions about conventions, architecture, dependencies, and expected behaviors. By placing them directly in the repository, they guarantee consistent application every time code is generated or modified.

Thanks to these customized directives, the LLM produces more reliable suggestions that comply with the project's standards and align with its technical choices, patterns, and best practices. The AI thus becomes a more accurate collaborator, reducing review effort and the risk of errors.

***Note***

The specific name for these rules may vary depending on the IDE, but the goal remains identical: to guide the AI toward more relevant and project-consistent suggestions. For example, in the tool Lovable the rules are called "Knowledge," whereas in Cursor the term "rules" is retained.

This repository explains in detail how to use rules with Cursor: https://github.com/llm-paper-org/cursor-dev-companion

### 3) Automating context with MCP

The primary role of an **MCP** (Model Context Protocol) service is to provide context to LLMs in a standardized way, cleanly *separating context presentation from the actual interaction with the LLM*.

The MCP service therefore acts as an intermediate layer that simplifies and automates these interactions. It establishes a security layer that prevents the exposure of non-public information, as illustrated in **Figure 12**.

#### (1) *Main Functions of the MCP Service*

According to the official documentation [27], the MCP service fulfills several key functions:

- **Resources**: expose data (equivalent to HTTP GET endpoints) to automatically load relevant information into the LLM's context.
- **Tools**: provide executable capabilities (equivalent to HTTP POST endpoints) that allow running code or triggering concrete actions.
- **Prompts**: supply reusable prompt templates designed to structure and optimize interactions with LLMs.
- **Completions**: enable direct interaction with the model in "completion" mode for real-time generated responses.

MCP services are not limited to serving static text or reading data from a database: they provide genuine secure execution mechanisms. These tools make it possible to integrate executable code into an application's context while rigorously controlling risks, thereby ensuring reliable, secure, and fully managed interaction between the LLM and the target system. For a concrete working example, download the code from https://github.com/llm-paper-org/basic-mcp-server.

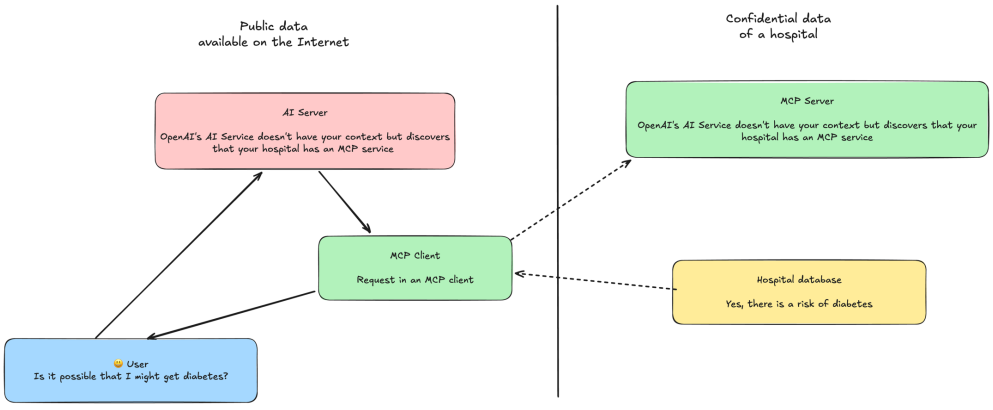



**Figure 12.** MCP service architecture[17].

[17]The execution icon comes from https://www.freepik.com/icons/execution.

(2) *Integration of the MCP Service into the Software Development Lifecycle*

As previously emphasized, providing adequate context is critical in software development workflows. Configuring an AI setup with access to contextual information from multiple components significantly reduces hallucinations in large language models (LLMs) and minimizes the need for batched requests.

MCP (Model Context Protocol) servers can be directly connected to sensitive data sources or tasked with executing operations within the system. Proper configuration enables LLMs to request and receive context through secure channels [28].

Common context sources needed while developing software that can be integrated via MCP include:

- Access to database schemas and data;
- Notes stored in tools such as Notion;
- Design assets hosted on platforms like Figma;
- Issues, workflows tracked in GitHub;
- Relevant information retrieved from the Internet to enhance task understanding;
- Files residing on local or remote servers.

Integrating targeted MCP services substantially improves the performance and reliability of AI agents. The basic integration process into an MCP-compatible client workflow is as illustrated in **Figure 13**.

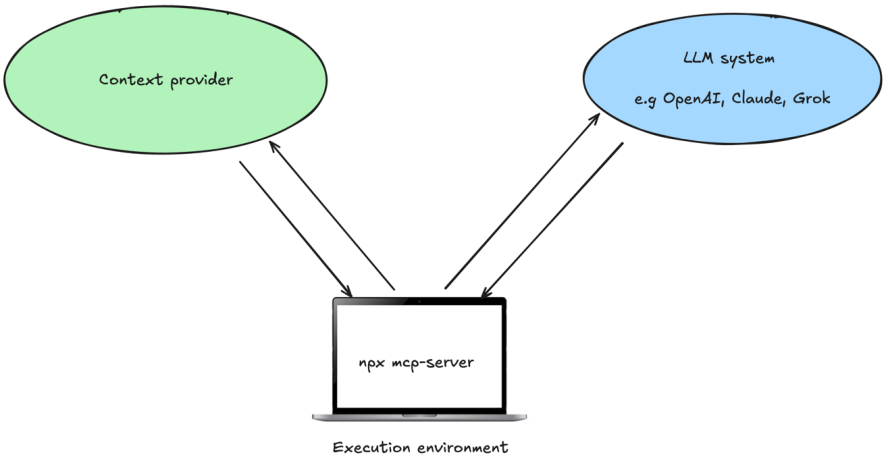



**Figure 13.** Interaction with a context provider with a local MCP Server.

MCP servers can be deployed either locally using standard input/output streams or remotely via RPC. Local deployment, often initiated through npx with a provider-specific command, ensures execution within the developer's controlled environment, thereby enhancing security. Remote MCP servers are also available as RPC endpoints.

For local integration, the MCP server executable and required parameters suffice. Many providers implement MCP servers in Node.js, allowing invocation via npx followed by the server package name.

This approach enables secure access to resources such as a MySQL database[18] using dedicated servers (e.g., mcp-server-mysql). Integration is straightforward and supported by platforms like Smithery, which offer extensions for environments including VSCode, Cursor,… For instance, the command "npx-y @smithery/cli@latest install @benborla29/mcp-server-mysql-client claude-code" (as documented by Smithery) exposes tools such as mysql_query and resources including database schema, column names, foreign keys, and relational structures to the LLM to claude-code.

The MCP MySQL server can be integrated with specific parameters that restrict access to certain database functionalities, thereby enhancing the security of the database.

```
claude mcp add mcp_server_mysql \
 -e MYSQL_HOST="127.0.0.1" \
 -e MYSQL_PORT="3306" \
 -e MYSQL_USER="root" \
 -e MYSQL_PASS="mysql_your_password" \
 -e MYSQL_DB="mysql_database_name" \
 -e MYSQL_POOL_SIZE="10" \
 -e MYSQL_QUERY_TIMEOUT="30000" \
 -e MYSQL_CACHE_TTL="60000" \
 -e MYSQL_RATE_LIMIT="100" \
 -e MYSQL_SSL="true" \
 -e ALLOW_INSERT_OPERATION="false" \
 -e ALLOW_UPDATE_OPERATION="false" \
 -e ALLOW_DELETE_OPERATION="false" \
 -e MYSQL_ENABLE_LOGGING="true" \
 -- npx @benborla29/mcp-server-mysql
```

Platforms such as MCPMarket.com and Smithery.ai serve as comprehensive marketplaces and registries for publicly available Model Context Protocol (MCP) servers. These resources enable developers to discover, evaluate, and integrate pre-built MCP servers that extend the capabilities of large language models and AI agents, thereby optimizing software development workflows through enhanced contextual access, tool integration, and reduced implementation overhead.

MCP servers support multiple interaction modes. Local access is commonly achieved via standard input/output streams. Additionally, remote access over the Internet is possible through HTTP-based transports, as demonstrated in the example implementation at https://github.com/llm-paper-org/http-mcp-server. This approach requires an MCP-compatible client and operates using a Remote Procedure Call (RPC) model over HTTP.

Agentic AI tools can seamlessly incorporate such servers into their workflows (**Figure 14**); for instance, GitHub Copilot Agent can integrate an MCP servers to provide enhanced contextual access during development tasks.

[18]https://github.com/benborla/mcp-server-mysql.

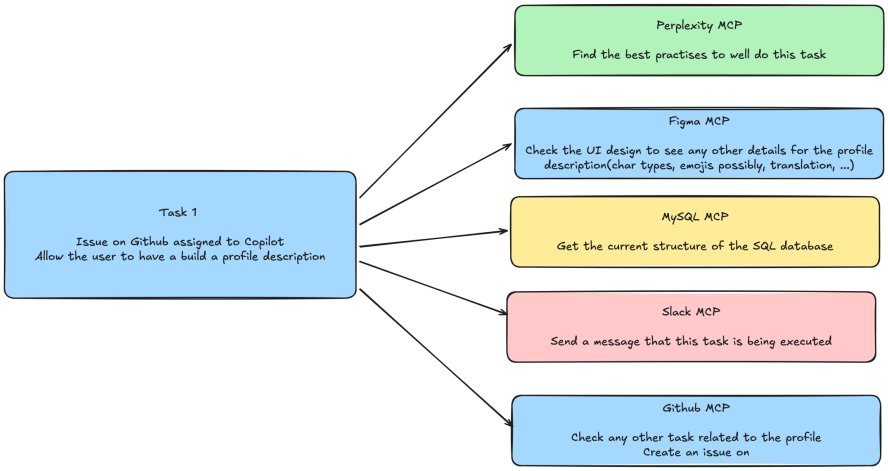



**Figure 14.** The agentic LLM has access to context from different sources from database to notes and designs thanks to MCP servers.

MCP-based architectures also present limitations. Incorrectly scoped tools or overly permissive execution rights may allow the model to invoke unintended actions, especially when prompts are ambiguous or adversarial. Moreover, MCP servers introduce an additional operational dependency: failures in the MCP layer (e.g., stale resources, inconsistent session state) can silently degrade model performance. These risks can be mitigated by applying least-privilege principles to tools, validating all tool inputs independently of the LLM, and monitoring MCP interactions through structured logging and audit trails.

## 4. Discussion on the Use of LLMs in Our Projects

Since 2023, with the emergence of models such as ChatGPT, the software development process has undergone profound change. Our initial uses of LLMs were very simple: applications that could be built in a single file, such as a Telegram bot or a smart contract to be deployed on Ethereum.

For larger-scale projects, we continued to work in a traditional manner, only occasionally integrating AI tools. The first LLM directly embedded in an IDE was GitHub Copilot, which revolutionized code autocompletion and significantly increased productivity.

The arrival of autonomous agents, such as Cursor or GitHub Copilot Agent, subsequently transformed the way we design and develop applications, moving from occasional assistance to active, structured collaboration with artificial intelligence.

### 4.1. Contribution of LLMs to Our Applications

#### 4.1.1. LLM Contribution to Pikelo—by Benedict Mukanirwa

In the development of Pikelo, LLMs have played a central role-not only in generating natural, coherent, and contextually appropriate dialogues between users and avatars, but also in guiding product design and continuous evolution.

From a technical standpoint, these models have dramatically reduced develop-

ment and debugging time. In early 2024, while building the MVP, the team sometimes spent hours hunting bugs because LLM usage was not yet fully integrated into the workflow. Today, a well-crafted prompt is often sufficient to identify and resolve an issue reported by a user.

At that time, code editors equipped with AI agents had not yet appeared. We were still in the era of "pure" LLMs, without true autonomous action capability. Agents, by contrast, can plan, execute tasks, and iterate independently with minimal human intervention.

In Table 6, we present the AI models integrated into our workflow.

Table 6. AI models used in Pikelo.

| Tool | Model | Task | Frequency | Comment |
|---|---|---|---|---|
| Cursor | Claude Sonnet (3.5, 3.7, 4, 4.5) | Code implementation | Moderate | Moderately priced cost, except for the "thinking" models. |
| | Gemini 2.5 pro thinking | Planning and creation of to-do lists. With a 1M-token context window, it navigates the files of a large project without much difficulty. | Moderate | Too expensive given the context size it provides. |
| | Gemini 2.5 flash thinking | Changing small pieces of code (or performing refactoring). Best suited for paying down technical debt. | Moderate | Free inside Cursor. It is the daily companion and helps keep costs down. It hallucinates from time to time, but can deliver solid results when combined with clearly detailed prompts (Chain of Thought: CoT). |

In the software development lifecycle, a recurring conflict often pits UI/UX designers against front-end developers. This tension arises when the implemented interface deviates from the mockups created in Figma or Adobe XD. Today, LLMs dramatically reduce this problem: by crafting a few precise prompts, it becomes possible to generate front-end code that faithfully matches the designs, thereby minimizing back-and-forth and ensuring optimal consistency between design and implementation.

#### 4.1.2. LLM Contribution to Troto—by Mathe Eliel

I have been leading the development of the Troto application for over five years at nfinic RDC SARL. The project was born before the emergence of AI tools and has evolved in step with their advances.

*Troto Version 1—a chaotic personal project*

Launched at the end of 2019, the first version relied on a frontend built with HTML, Cordova, and Framework7, and a PHP 7 backend. The current interface has been completely rebuilt in Flutter, while the backend has migrated to Type-

Script with AdonisJS as the main framework.

Initially, Troto was a personal project with no organizational or collaboration constraints. The backend used no framework at all, and onboarding new developers quickly became a major challenge, as technical communication and consistency were difficult to maintain.

*Troto Version 3—a solid team structure and team*

Version 3 was designed from the ground up as a collaborative project, with dedicated teams for front-end, back-end, file management, performance, database, translations, design, and prototyping. This organization laid the foundation for a robust, well-structured application, even before the massive integration of artificial intelligence.

In 2024, faced with the new capabilities of LLMs, we encouraged their systematic adoption to speed up development and optimize performance. In 2025, the arrival of agent-driven development changed everything: productivity increased, radically transforming the workflow on the Troto project codebase.

Table 7 provides an overview of modules utilizing AI and those that do not, along with their respective commit counts.

Table 7. AI tool usage and commit counts across modules in the Troto system.

| Module | Commits | Programming Language | Framework | AI Usage |
|---|---|---|---|---|
| Front-end | 1845 | Dart | Flutter | High |
| Back-end | 619 | TypeScript | AdonisJS | High |
| File management | 45 | JavaScript | - | Medium |
| Payments | 69 | Rust | Actix.rs | Not yet |
| Serverless functions | 5 | JavaScript | - | Medium |
| Web application | 130 | HTML | ReactJS | High |
| Admin Dashboard | 969 | PHP | - | No |
| Orchestration | 56 | | Ansible | High |

Many Troto modules make intensive use of artificial intelligence, particularly those designed with a team-oriented approach, as in Troto version 3. With GitHub Copilot Agent, we now integrate AI as a true team member. Thanks to a well-thought-out and structured architecture, the agent's suggestions are generally accurate and relevant. In contrast, in legacy parts from version 1, such as the Admin Dashboard, which was poorly designed originally, the use of LLMs often generates hallucinations, making the results less reliable.

Thanks to LLMs, issue resolution on Troto has increased from 5 to 10 per week per developer to sometimes more than 50 with the usage of GitHub Copilot Agent. This is illustrated by the increase in GitHub Copilot Actions, as shown in Figure 15. Rather than focusing on manual code writing (except in special cases), developers now concentrate on business logic, specifications, and tests.

We estimate that integrating an MCP server into our workflow would allow GitHub Copilot Agent to automatically execute tests and reduce iteration cycles. Since GitHub charges each interaction as a Premium Request, a dedicated MCP for testing would significantly reduce costs: instead of triggering a new request after each validation, the MCP would handle execution and verification within the same session.

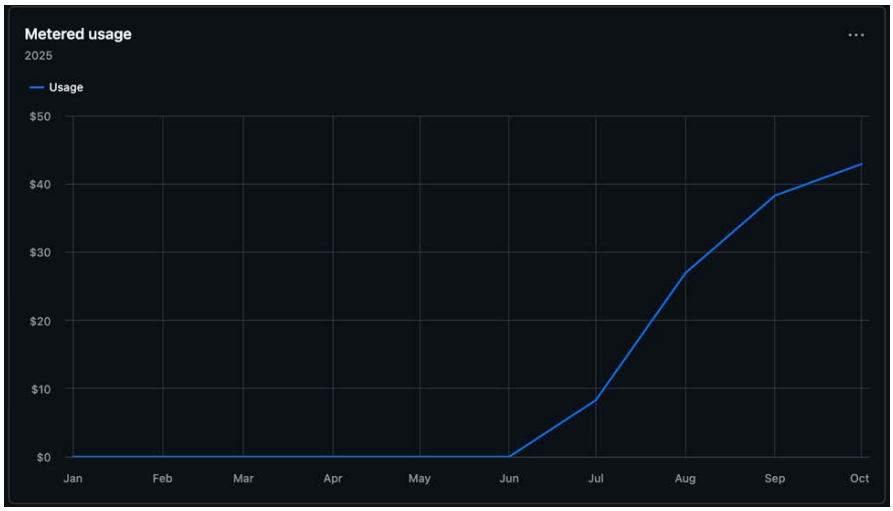



**Figure 15.** Growth in the use of GitHub Actions services.

## 4.2. Discussion on Source Code and Application Security

The use of LLMs poses numerous security risks, particularly sensitive for application developers. On one hand, platform providers can access data exchanged during interactions. Prior to adopting large language models (LLMs) in the Troto development workflow, several security vulnerabilities had already been identified. The most severe incident involved the public password-reset form on the artist dashboard. Although the form was designed to deliver one-time passwords (OTPs) via SMS in a secure manner, it lacked adequate rate-limiting and replay-attack protection. Malicious actors exploited this weakness by repeatedly triggering OTP requests, generating an extremely high volume of SMS notifications. The resulting charges rapidly depleted the online banking account used to pay for the bulk SMS service.

The progressive integration of LLMs into the development process has simultaneously introduced new classes of risk while substantially improving our ability to detect latent vulnerabilities and accelerate remediation.

The principal security concerns we have observed and addressed since adopting LLMs are the following:

1) Inadvertent leakage of sensitive infrastructure secrets during prompting

Sensitive values—server IP addresses, SSH keys, database credentials, deployment secrets, Ansible vault passwords—frequently appear in the Ansible playbooks, inventory files, group_vars, and vault-encrypted files that constitute the deployment project. Because these files are often open or referenced in the IDE

during refactoring, debugging, or feature implementation, developers occasionally include excerpts (or even fully decrypted sections) in LLM prompts to obtain more precise assistance.

To mitigate this risk, we instituted a strict organization-wide policy that prohibits the inclusion of any production or pre-production secrets in prompts sent to external LLM providers. Whenever context from deployment-related files is genuinely required, developers must now manually redact all sensitive values before copying code into the prompt, use placeholder tokens (e.g. '{{ ansible_host }}', '{{ db_password }}') combined with a description of the intended structure, or—preferably—leverage local/offline models that can ingest the complete project context when working on infrastructure code.

2) Propagation of legacy security weaknesses through code completion

Because LLMs operate predominantly in a completion paradigm, they are strongly conditioned by existing patterns found in the codebase. When older modules contained security deficiencies (insufficient input sanitization, missing CSRF tokens, unsafe use of eval/exec, etc.), these same patterns were frequently reproduced—and occasionally amplified—in newly generated sections.

To address this issue, we introduced and progressively enforced project-level security instructions (via Cursor rules, Lovable Knowledge files, and dedicated system prompts) explicitly requiring the model to adhere to current best security guidelines and language/framework-specific secure coding recommendations. In parallel, we implemented a mandatory human security-focused code review gate for every LLM-generated pull request before merging into protected branches.

3) Merge conflicts and security regressions induced by parallel agent task execution

Assigning multiple tasks in parallel to AI agents (or to a single agent operating in multi-file mode) can dramatically increase velocity, but hallucinations or divergent reasoning paths sometimes lead to concurrent modifications of the same files, resulting in conflicting or insecure merge outcomes.

To reduce these risks, we largely shifted to sequential task assignment for all security-sensitive modules and any work touching shared infrastructure or configuration files. For the remaining parallelizable tasks, we established a reinforced review checklist and now require explicit approval by at least one senior engineer before integration into the main branch.

When different system components are developed by distinct agents—often powered by different underlying models and prompt strategies—subtle semantic or architectural incompatibilities can emerge. Current agentic toolchains still offer limited support for automated end-to-end integration testing across agent boundaries.

We are actively working toward a more tightly coupled agent orchestration layer that combines standard CI/CD pipelines, dedicated MCP servers responsible for test execution and validation, centralized observability across human- and agent-generated contributions, and automated cross-component contract and integra-

tion testing (Figure 16). This integrated approach aims to preserve strong security invariants while retaining the substantial velocity gains provided by multi-agent LLM-assisted development.

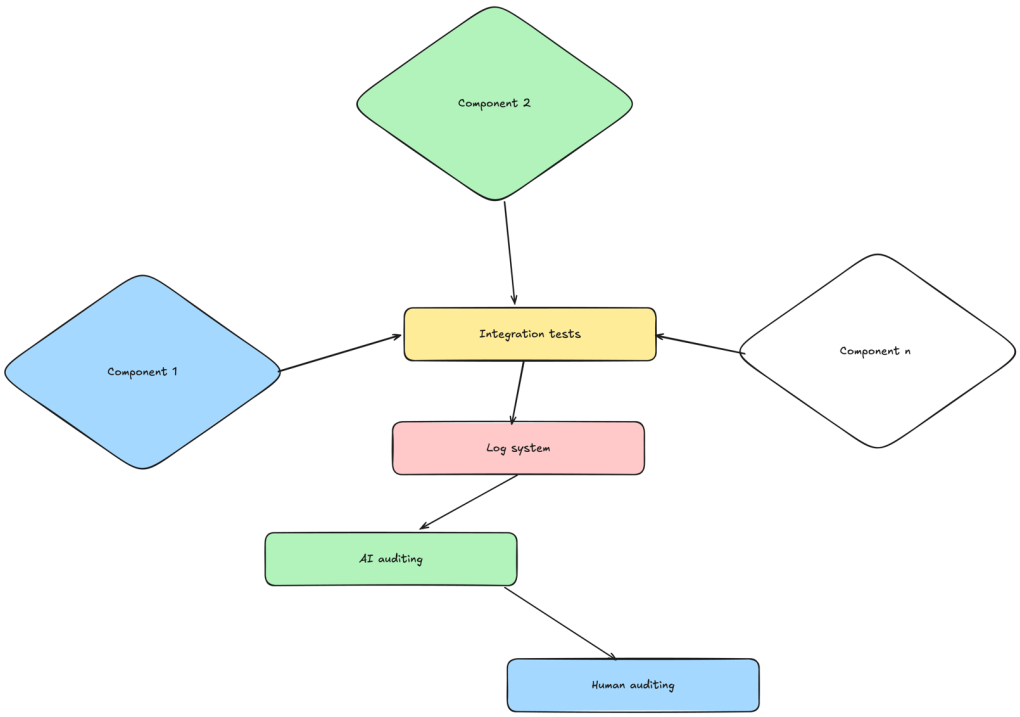



Figure 16. Global system integrated security.

## 5. Conclusions

This article explored recent advances in artificial intelligence and their integration into software development projects. These innovations profoundly transform the software creation process. By choosing the right tools, it is possible to multiply individual and collective productivity by 10, or even 100, as observed in the Troto and Pikelo projects. However, effective AI adoption relies on respecting fundamental development best practices: adopting robust frameworks, using clear communication tools, rigorous code structuring, and version control.

The use of LLMs raises two major challenges for any architect: access cost and result quality. A concrete method has been presented to reduce costs and optimize responses by enriching the context provided to the model. The MCP protocol has been detailed, along with its precursors such as RAG, and simpler approaches like rule or instruction files adapted to development environments.

Finally, the integration of LLMs comes with security risks that must be managed to protect the system and users. Particular vigilance is required, especially regarding the confidentiality of transmitted data and validation of generated outputs.

This article, of a general nature, highlights several tools for fully leveraging LLMs in creating complex systems. More specialized explorations can be conducted in areas such as frontend, backend, security, communication, or project

management, where new tools emerge daily.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References


[1] IBM (2025) What Is Machine Learning? https://www.ibm.com/think/topics/machine-learning

[2] Ng, A. (2025) Stanford CS229: Machine Learning—Linear Regression and Gradient Descent | Lecture 2 (Autumn 2018). https://www.youtube.com/watch?app=desktop&v=4b4MUYve_U8

[3] Sun, S., Cao, Z., Zhu, H. and Zhao, J. (2019) A Survey of Optimization Methods from a Machine Learning Perspective. arXiv:1906.06821.

[4] (2025) Qu'est-ce que le Deep Learning? https://aws.amazon.com/what-is/deep-learning/

[5] Model Context Protocol (2025) MCP TypeScript SDK. https://github.com/modelcontextprotocol/typescript-sdk

[6] (2025) Faiss. https://github.com/facebookresearch/faiss

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I. (2017) Attention Is All You Need. arXiv:1706.03762.

[8] Ilya, S., Oriol, V. and Quoc, V.L. (2025) Sequence to Sequence Learning with Neural Networks. https://arxiv.org/pdf/1409.3215

[9] Maximer (2019) What Is a Transformer? https://maximeallard.lu/2019/01/04/what-is-a-transformer/

[10] Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. (2018) Improving Language Understanding by Generative Pre-Training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[11] (2025) Qu'est ce qu'un réseau neuronal. https://aws.amazon.com/what-is/neural-network/

[12] Shumer, M. (2025) Startup Growth Consultant Claude 4 Prompt. https://shumerprompt.com/prompts/startup-growth-consultant-claude-4-prompt-prompt-0cee13e1-2893-4620-8dcd-ba9068d0e5ed

[13] Google Cloud (2025) What Is Vibe Coding? https://cloud.google.com/discover/what-is-vibe-coding

[14] (2025) What Is Agentic AI? https://www.ibm.com/think/topics/agentic-ai

[15] Umoren, S. (2025) 6 Model Context Protocol Alternatives to Consider in 2025. https://www.merge.dev/blog/model-context-protocol-alternatives

[16] Mode Control Protocol (2025) What Is the Model Context Protocol (MCP)? https://modelcontextprotocol.io/docs/getting-started/intro

[17] Euro Tech Conseil SARL (2025) Cycle de vie du développement logiciel: Qu'est-ce que le SDLC? https://www.eurotechconseil.com/blog/cycle-de-vie-developpement-logiciel/

[18] Replit (2026) About Replit—Empowering the Next Billion Software Creators. https://replit.com/about

[19] OWASP (2025) 2025 Top 10 Risk & Mitigations for LLMs and Gen AI Apps.

https://genai.owasp.org/llm-top-10/

[20] (2025) Qu'est-ce qu'un cadre en programmation et en ingénierie? https://aws.amazon.com/fr/what-is/framework/

[21] Patrick, L., Ethan, P., Aleksandra, P., Fabio, P., Vladimir, K., Naman, G., Heinrich, K., Mike, L., Wen-tau, Y., Tim, R., Sebastian, R. and Douwe, K. (2021) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. https://arxiv.org/pdf/2005.11401

[22] OpenAI (2025) Vector Embeddings. https://platform.openai.com/docs/guides/embeddings

[23] Grigoryan, M. (2025) Tech for Non-Tech 40: Data Exchange Formats: A Guide to JSON, XML, YAML, and More. https://medium.com/@margrig96/tech-for-non-tech-40-data-exchange-formats-a-guide-to-json-xml-yaml-and-more-aa5289093b4a

[24] Scenari-Community (2025) Conception de bases des données. https://stph.scenari-community.org/bdd/0/co/pri1c21.html

[25] Convex (2025) Convex vs. Supabase. https://www.convex.dev/compare/supabase

[26] SentinelOne (2025) Top 14 AI Security Risks in 2025. https://www.sentinelone.com/cybersecurity-101/data-and-ai/ai-security-risks/

[27] Jessica, J., Jenny, J., Maggie, W. and Rebecca, G. (2025) Cybersecurity Risks of AI-Generated Code. https://cset.georgetown.edu/publication/cybersecurity-risks-of-ai-generated-code/

[28] De Santis, M. (2024) Full stack: Définition et usages. https://www.appvizer.fr/magazine/services-informatiques/gestion-informatique/full-stack-glossaire