WHITE PAPER

Using Generative AI to Assist Software Development

Gorilla Logic

Agile. Unstoppable.

TABLE OF

Contents

SECTION 1

Introduction

Artificial Intelligence (AI) is not just a revolutionary field; it's already an integral part of our daily lives. AI's influence can now be seen everywhere, from personalizing our social media feeds to enhancing the accuracy of weather forecasting, and even protecting our finances through fraud detection. Recently, a new wave of innovation has arisen within this domain, marked by the introduction of Generative AI (GenAI). This subset of AI, illustrated by products like ChatGPT, is not entirely novel. However, its potential is more evident now than ever. With the continuous advancements in GenAI, we are on the cusp of experiencing its rapid expansion and transformative impact now and in the years to come.

The world started talking about GenAI more than 60 years ago, when, in the late 1960s, the Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs) were introduced. Both are probabilistic frameworks that are especially good at modeling complex, real-world phenomena in a mathematically tractable way [1].

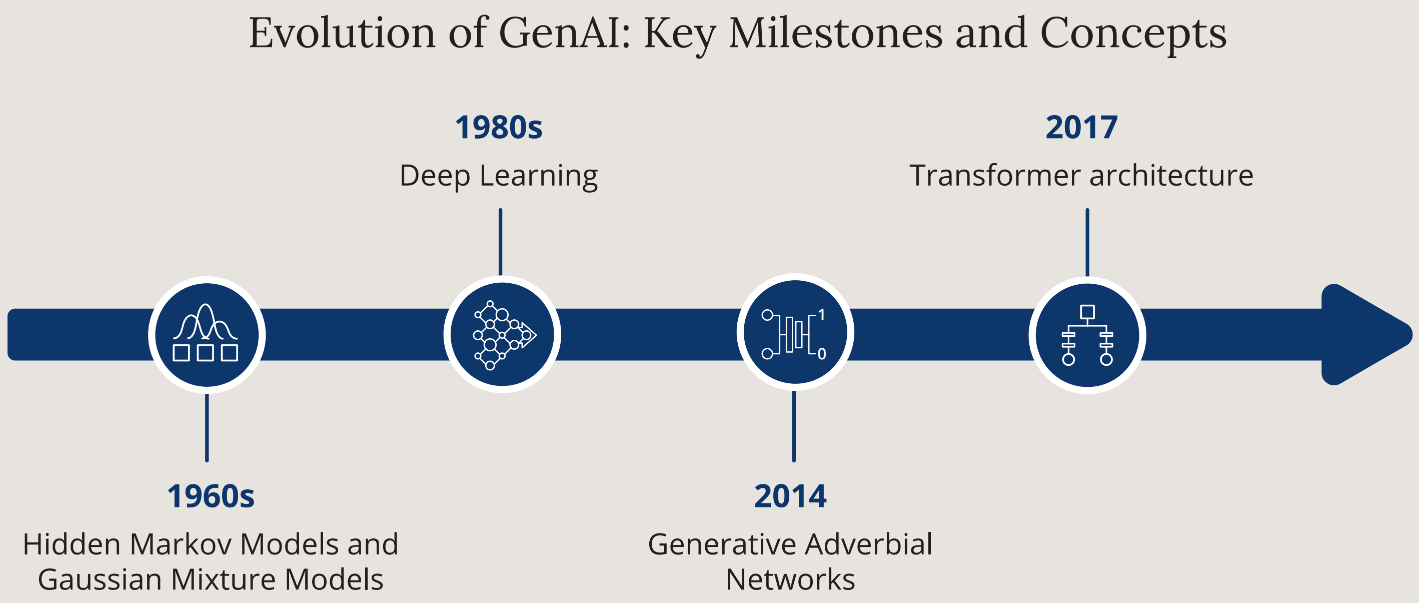

Evolution of GenAI: Key Milestones and Concepts
1980s
Deep Learning
2017
Transformer architecture
1960s
Hidden Markov Models and Gaussian Mixture Models
2014
Generative Adverbial Networks

Fig. 1. Evolution of GenAI: key milestones and concepts

Later, in the 1980s, Deep Learning (DL) emerged, and performance advanced as it became possible to build artificial neural networks that could learn and make decisions from large amounts of data, imitating how the human brain operates. With the introduction of Generative Adversarial Networks (GAN) in 2014 along with recent rapid technological advancements, as well as increased data capacity and availability, the models have seen their potential shine [1].

GenAI models are based on deep neural networks and try to learn the pattern and structure of big training data sets with the intention of generating similar new content. Based on that, the models can produce different forms of data, including text, images, sound, or source code [1].

In 2017, a new architecture for this kind of model was introduced: the Transformer architecture [2], which has become the backbone of much of the new GenAI technologies. From a simplistic view, the Transformer architecture basically proposes two elements: an encoder and a decoder. The first one gets an input and builds a representation of it. Then, the decoder analyzes the representation created by the encoder and produces an output sequence.

The following figure shows the general architecture of the Transformer model. You can find a valuable resource explaining it in more detail **here**.

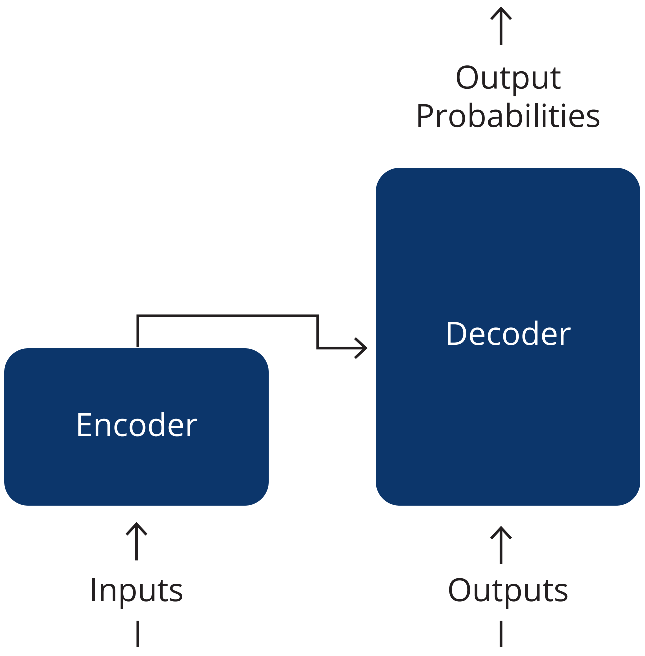


Fig. 2. General architecture of the Transformer model

Most of the Transformer models have been trained on immense amounts of raw text; that's why they are called Large Language Models (LLMs). They are outstanding in accomplishing natural language tasks such as translation, text summarization, and question answering. The data sources they have been trained on include books, articles, and websites, allowing them to generate human-like text, respond effectively to contextual prompts, and help with tasks through tools like chatbots and virtual assistants [3].

One of the LLM's real-world applications that has attracted wide attention is their use in code generation tools. Transformer models can take natural language tasks as input and generate source code solutions. At its best, using AI-based tools can improve software development efficiency by reducing time and effort, eliminating repetitive tasks, increasing code quality and readability, and blocking insecure coding patterns [4].

In 2022, OpenAI launched ChatGPT, a large language model–based chatbot that has captured the attention of the entire software development community and beyond. The model can generate code from natural language descriptions—showing the power of GenAI while changing the way society perceives AI. And it's not the only code-generating AI model available for the community: models like Codex, CodeBERT, PaLM, and AlphaCode have been trained on millions of lines of code and are now the heart of popular AI coding assistants like Github Copilot, Amazon CodeWhisperer, and Tabnine [5]. These tools are widely used for code generation, completion, translation, repair, summarization, and explanation, among other use cases.

SECTION 2

Use Cases for Generative AI in Coding

AI has enabled researchers, communities, and industries to develop powerful coding assistants. These tools help to increase developer efficiency and productivity, bring junior developers on board quickly, improve developers' career development paths, and significantly reduce the time invested in code review and rebuilding [6]. Developers can use AI tools for a variety of tasks, including defect detection, test case design, modernization, clone detection, code generation, translation, completion, refinement, or summarization [4]. We cover some of the tools' use cases below.

- **Code generation** is the process of automatically generating source code based on user input. Transformer-based models are exceptionally effective at this kind of task. The models rely on their learning from immense data sources, including publicly available code.
- **Code completion** leverages contextual representations learned from massive amounts of code, enabling the tools to suggest different ways of completing an unfinished line of code. The tools can offer autocompletion if the user asks or automatically as the developer types. Based on the model's metrics, it can save time and reduce errors in methods, variables, or any part of the code.
- **Code translation** is one of the most significant challenges researchers and companies working on AI coding assistants face. Converting code from one programming language to another is tricky due to differences in syntax, semantics, frameworks, or APIs. Some tools provide features that allow developers to translate code to another programming language. Still, most are limited to a few languages, such as Java or Python. The main goal of code translation is to migrate legacy software, but some applications also cover cloning existing features or improving performance.

- **Code refinement** occurs when LLMs use their learning from different code sources to suggest ways of fixing bugs or vulnerabilities. Through the data they have been trained in, they can analyze the code and find patterns, structures, methods, or variables that probably introduce unwanted behaviors into the system. Improving or simplifying existing code is also part of this use case.

- **Code summarization** allows developers to generate documentation quickly about the code they are building. By copying the code into the tool or allowing the tool to read the code, the developer can get a natural language description of any method, variable, or code snippet.

- **Clone detection** supports the best practice of not having duplicate code across multiple parts of a single system. AI assistants can identify identical or similar code fragments so that developers can improve how they structure their code and software. While many types of code similarity are easy to find for the current LLMs, others are trickier, so some tools will ask for the developer's intervention to double-check a suggestion's correctness.

- **Testing** encompasses a much broader range of activities than code refinement and summarization, as mentioned above. AI coding assistants also support code quality and functionality by generating test cases, identifying parts that are not adequately tested, or understanding the causes of bugs.

Benefits of Transformer-Based Models

We have covered the most common current use cases of AI coding assistants. However, their use surpasses code generation and understanding. Transformer-based models can also bring outstanding performance to other fields, such as the following [1]:

- **Secure coding practices:** With adequate management, AI-based coding assistants can generate secure code. By using solid prompts and being explicit about the desired outcome, a model's code can have fewer bugs or vulnerabilities than code created by an unseasoned developer. Additionally, developers can use the tools to produce test cases to guarantee the software's security.

- **Analysis of cybersecurity incidents:** Some Transformer-based models can automatically analyze cybersecurity incidents. Furthermore, they can generate strategic recommendations to fortify the system against future attacks.

- **Threat intelligence:** LLMs have been trained in large amounts of data that allow them to identify potential security threats, analyze their risk, and generate recommendations to solve or mitigate them.

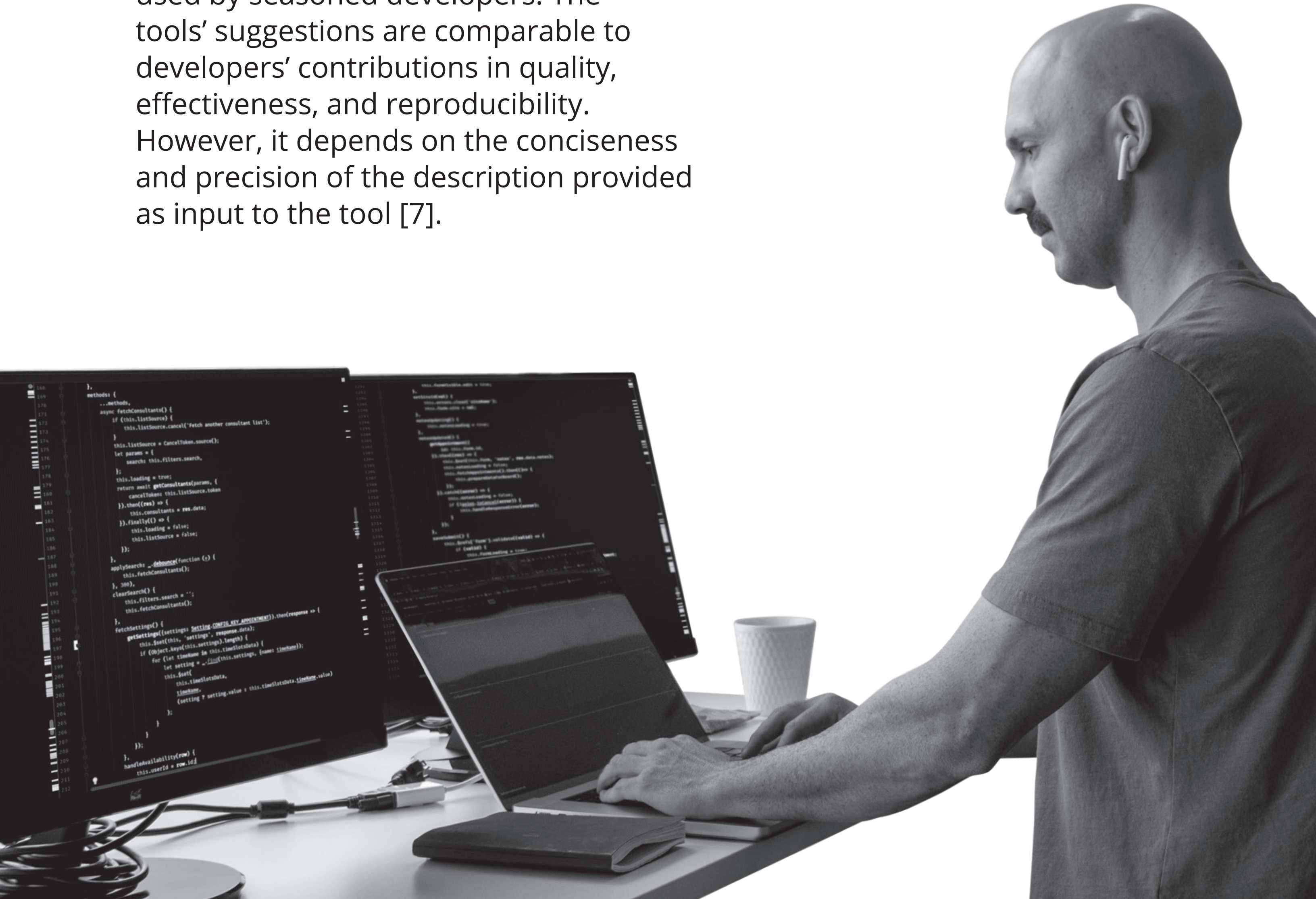
- **Developer training:** LLMs are very good at natural language tasks. By exploring the tools and giving them adequate descriptions, it is possible to generate documentation, processes, and even courses for developer training. Companies can adapt the results based on their business needs.

SECTION 3

The Upsides of Using AI-Based Coding Assistants

It’s already easy to see the advantages of using AI-based coding assistants—and their entire potential is still yet to be explored. While there are many upsides to the tools, we will focus on the most beneficial ones:

- **Efficient solutions:** Some researchers have demonstrated that AI coding tools can become an asset when used by seasoned developers. The tools’ suggestions are comparable to developers’ contributions in quality, effectiveness, and reproducibility. However, it depends on the conciseness and precision of the description provided as input to the tool [7].



- **Investment of time in what matters the most:** Designing, developing, and maintaining software involves many tasks. Some of these tasks are tricky, but some are straightforward. Transformer-based models can perform exceptionally well on simple tasks such as code description, simple code logic creation, and best practices documentation [7]. Developers and companies can then invest their time in the more important work of designing solid architectures, implementing best practices, preventing vulnerabilities, and understanding the client's needs. In addition, LLMs can significantly flatten the learning curve for developers by providing guidance, suggestions, and code snippets based on the large amounts of data in which they were trained [8].

- **Cost efficiency:** The quality and effectiveness of the code produced by LLMs allow companies to save money by enabling developers to code quickly, focusing on complex problem-solving, and effectively allocating resources. While AI suggestions may include bugs, their nature and context are often not as thorny as bugs created by junior developers.

  These buggy AI suggestions are based on vast datasets of coding patterns and solutions, so they often lead to more predictable and manageable bugs than the diverse and potentially complex ones coded by junior developers. This predictability can facilitate the debugging process and enhance the development workflow, balancing speed and quality [7].

SECTION 4

The Downsides of AI-Based Coding Assistants

While there is a great deal of excitement about the benefits of using AI-based coding tools, there are also downsides. However, these current limitations, such as the need for specific guidance to avoid vulnerabilities like SQL injection, are expected to decrease over time. Meanwhile, input and review from experienced developers remain crucial in guiding and enhancing the effectiveness of these tools. Some of these current challenges include:

- **Insecure code suggestions:** LLMs can introduce significant security challenges when using them in a variety of code-related tasks based on the data sets they have been trained in. For example:
  - SQL injection occurs when the tools suggest SQL queries that don't verify the user input. The system can interpret the input as SQL instead of ordinary user data, thereby bypassing security checks and potentially affecting the databases.
  - Hard-coded credentials pop up when some of the AI-coding assistants' suggestions include explicit credentials, which is a security concern. Developers must specifically prompt this kind of tool for secure code suggestions (e.g., with no vulnerabilities to SQL injection or hard-coded credentials) [4, 9].
- **Buggy or non-optimal solutions:** Some code or solutions proposed by AI-based coding tools have significant bugs or include non-reproducible or non-optimal snippets. Consequently, the tools can burden inexperienced developers who may need help to see or find the issues introduced [7].

- **Privacy and compliance concerns:** Several AI coding tools have faced privacy concerns, mainly because they store the user input in their library, creating potential leaks of user information. Some AI coding tools have enabled users to opt out of giving their data to mitigate this potential security risk. Additionally, some are creating vulnerability prevention systems and new features to identify and reduce insecure coding practices [1, 9].

- **Issues with complex challenges:** AI-based coding tools work exceptionally well with simple code challenges. However, when the developer is working on a complex, multi-factorial design, it will take more work to get concise and solid results out of the tool. All of the tools need a simple, machine-readable specification to produce code [7, 10]. For example, they need to understand the broader context, including particular design patterns, non-functional requirements, and specific user preferences.

- **Overdependence on the developer's experience:** AI coding tools cannot generate beyond the parameters the developer uses, and sometimes the tools need help understanding details in the input. Without this clarity, the tools can generate poor code or produce documentation difficult to maintain in the long term [4, 7, 8].

“AI-based coding assistants are a new technology with tremendous potential for growth and refinement. While they currently require further development, particularly in fully understanding natural language and structuring complex solutions, these tools are constantly evolving and learning, and should only get smarter and better with time and training.”

SECTION 5

Early Results of AI-Assisted Development

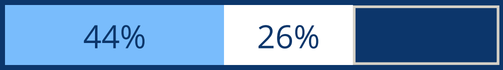
Several companies have already evaluated the changes in developers’ efficiency and code quality once they start using these tools. They have also assessed the adoption and perception of those using them. A 2023 McKinsey study [11] shows that depending on the coding task and the developer’s experience, it is possible to decrease a task’s completion time between 30% and 50%. In highly complex tasks, however, the time saved is less impactful, reaching a maximum of 10%. The study also showed that GenAI tools can improve developer experience. **Participants reported an increase in happiness, fulfillment, and flow state attainment when using AI tools, as they quickly unblocked them from repetitive or tricky tasks.**

The StackOverflow 2023 Developer Survey [12]

checked in with over 90,000 developers regarding their sentiments about AI tools. **Key findings include:**

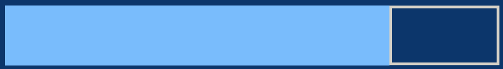
70%

of survey respondents already use (44%) or plan to use (26%) AI tools in their development process.



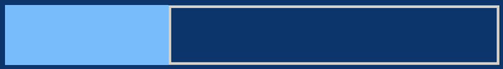
77%

feel favorable towards using AI tools as part of their development workflow.



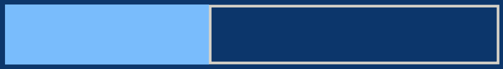
33%

see increasing productivity as the primary benefit of using AI tools



42%

trust the accuracy of the output of AI tools used in their development workflow.



While there is great interest in businesses exploring the potential for higher productivity gains with AI, to achieve success with the tools and minimize risk, these companies must rely on their engineering leaders to design a structured approach for adopting AI tools and train their developers. Tapping into their seasoned developers' knowledge will also have a positive impact on their junior teammates and the quality of their software.

SECTION 6

Best Practices for Using AI Tools in Coding

Code quality, developer productivity, and security can be improved with the proper use of AI-based coding assistants. At the same time, using the tools without the required practice, caution, and awareness of their limits can lead to bugs, insecure code, vulnerabilities, and compliance concerns. Here are some tips and best practices to consider when using AI coding tools.

1. The input is key

LLMs have difficulty understanding the input if the description is too long or contains multiple sentences. It is common for humans to use too many phrases or complex words to explain even a simple problem, which can lead to vague results. Focus on using programming-specific technical keywords that can help the tools recommend relevant solutions.

Adding appropriate context and guidance, such as sample code or tests, will aid the tool in finding the right solution for the developer. It is helpful to define a standard prompting template that provides that information [4, 7, 13], like this:

```
Unset
{Context}

{Problem or question}

{Special requirements}
```

Fig. 3. Prompt template example

The Prompt Template provides:

- **Context:** Share relevant information that the model needs to make the best recommendation. It is also useful to give the model a role to play. For example: "*You are a seasoned Python Developer who will propose efficient ways of building a movie recommendation system*."
- **Problem or question:** Describe the issue that needs to be solved. You could prompt the AI to:
    - Create code to do a specific action—"*Create a function to add two numbers.*"
    - Explain how to do something—"*Create a method that searches for a country's currency using the Bing Web Search API.*"
    - Give advice on how to solve an issue—"*Implement unit tests for the test.py file.*"
- **Special requirements:** Sometimes, we need the code to be proposed in a specific way. This part of the prompt should include the different requirements and decorators we want the model to consider. Some examples include: "*Use SOLID principles,*" "*Explain each line of code*," or "*Name each class as an animal.*"

2. Divide and conquer

AI coding tools such as GitHub Copilot are solid in proposing solutions for fundamental problems, but they need help when the challenge is more complex. The quality of the code presented by the tool will depend a great deal on the conciseness of the prompt provided. Divide complex challenges into several small topics so that the tool can satisfy the criteria described in the prompt [7, 14].

- Ask simple and specific questions.
- Build the question or problem in such a way that you would logically receive a short output, not an entire coding book.
- Articulate the key requirements and steps the model should take into account.
- Break the challenge down into small descriptions. An entire architecture is made of different classes, elements, or subsystems, so try to build the prompt in a way that the model can answer brief requirements or steps and not the entire architecture at once.

As an example, suppose we want to build a basic web application that allows users to register, log in, and post messages.

1. Instead of asking the AI coding assistant to generate the entire application at once, break it down into smaller blocks to build first: Registration, Login, Message Posting, and Display Messages.
2. For the user registration block, build a prompt telling the tool to generate a function of a registration form, including fields for username, password, and email.
3. Then ask it to consider input validation and a method to store the data in a MySQL database.

3. Expertise before confidence

AI-based coding assistants help to increase developer efficiency and significantly reduce the time invested in code generation, review, and rebuilding. Most of the time, AI-generated code is more advanced than solutions proposed by junior developers relative to correctness, optimality, and reproducibility. Sometimes, however, coding assistants struggle to get answers to our prompts and will produce buggy or insecure code. Therefore, keeping a human in the loop is always required.

Maintaining pair programming and peer review standards is essential to ensure the model's output meets the project requirements and quality standards. If all the code is appropriately reviewed and documented, it minimizes risks, decreases technical debt, and maintains a reliable codebase. Descriptive variable names and functions, consistent of coding patterns, clear comments, and rigorous Architecture Decision Records will always have a positive impact [4, 7, 13, 14].

4. Rigorous testing can save your day

As mentioned, some of the outputs generated by AI-based coding tools include bugs and severe security vulnerabilities. Testing the generated code before integrating it into production is essential to address these challenges and mitigate risks. Checking the integrity of the training data, validating use cases, and conducting unit and integration testing can significantly help ensure the code has been properly created and is free from bugs, vulnerabilities, and undesirable behaviors [4].

In addition, it's been demonstrated that AI-based coding assistants will make secure code suggestions when the developer explicitly requires the code to contain no vulnerabilities, such as SQL injection or hard-coded credentials. Use requirements in your prompt that can sound obvious, such as “code shouldn't be vulnerable to SQL injection” [9].

5. There is no ‘One Tool’ to rule them all

There is already an established list of viable AI-based coding assistants, but none are a perfect tool or one-size-fits-all solution. Each tool has its strengths, limitations, and use cases. Choosing the right AI-based coding tool should be a meticulous process that closely aligns the developers’ and company’s specific use cases, project requirements, and preferences. Speed, efficiency, pricing, support, documentation, privacy, security, and use cases are all key factors that need to be considered. By carefully evaluating these options and matching them to the project requirements and company goals, we can all leverage AI-based coding assistants’ power to improve our efficiency, code quality, productivity, and experience [13, 15].

Balancing the line between code generated by AI-based tools and the code we manually develop is also important. Developers should rely on their experience and knowledge to decide when the automatically generated code is adequate or if manual intervention is needed. In the end, AI-based assistants are an aid rather than a solution. By properly using them, developers can enhance their problem-solving, productivity, and code quality.

SECTION 7

Conclusion

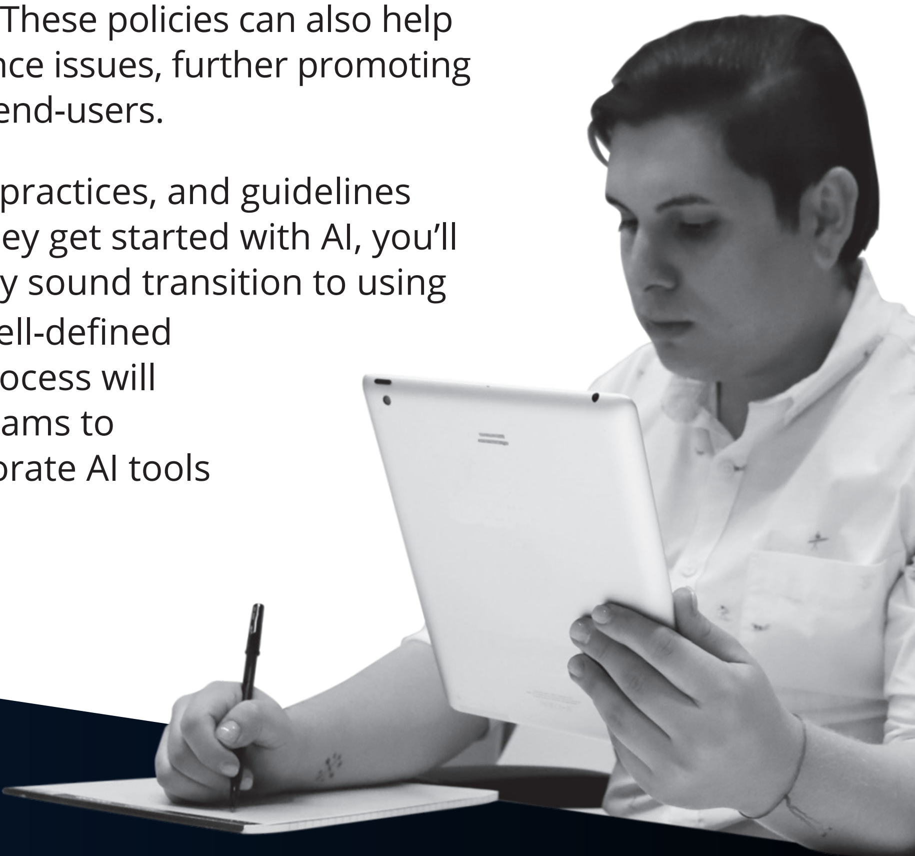
Policies, guidelines, and training are your ally

There are different topics and best practices to remember when working with AI-based coding tools. Organizations must establish clear policies, guidelines, and training for their developers to ensure responsible and effective integration of these tools into their software development processes.

Training and learning assets can decrease the learning curve and improve developer experience when beginning to incorporate AI into your software development process. Some assets, such as a prompt template, a prompts record, or a list with common phrases, can speed up an AI tool's adoption.

Creating business policies for AI usage will provide a clear framework for developers to understand their roles, responsibilities, and ethical boundaries when using AI tools. Writing a set of best practices will also ensure that AI tools are being used correctly and consistently, minimizing the risk of errors or poor outcomes. These policies can also help address potential legal and compliance issues, further promoting trust among your stakeholders and end-users.

By creating corporate policies, best practices, and guidelines for your developers to lean on as they get started with AI, you'll also promote a smoother and legally sound transition to using these coding tools. Establishing a well-defined structure at the beginning of this process will also empower your development teams to confidently and successfully incorporate AI tools into their day-to-day operations.



Gorilla Logic

Agile. Unstoppable.

Another vendor might be fine...

But with Gorilla Logic,
you’ll be Unstoppable.

GET TO KNOW US →

ABOUT

Gorilla Logic

For more than 20 years, Gorilla Logic has partnered with leading enterprises across industries to help define, architect, and deliver their most important digital products and platforms. Our deep technical and domain expertise + proven approach enable clients to efficiently innovate, scale, and modernize—creating secure products that their customers love.

With headquarters in the U.S. and nearshore development hubs across Latin America, our highly collaborative Agile teams bring a distinctive culture of tech-obsession and problem-solving to our clients' projects. Everyone at Gorilla Logic strongly believes that when our colleagues and clients win, we all win, which makes for a spirit of partnership unique in our industry.

Bibliography

[1] M. Gupta, C. Akiri, K. Aryal, E. Parker, and L. Praharaj, “From ChatGPT to ThreatGPT: Impact of Generative AI in Cybersecurity and Privacy,” IEEE Access, vol. 11, pp. 80218–80245, 2023, doi: 10.1109/access.2023.3300381.

[2] A. Vaswani et al., “Attention Is All You Need,” arXiv.org, Jun. 12, 2017. https://arxiv.org/abs/1706.03762

[3] E. L. Ouh, B. K. S. Gan, K. Jin Shim, and S. Wlodkowski, “ChatGPT, Can You Generate Solutions for my Coding Exercises? An Evaluation on its Effectiveness in an undergraduate Java Programming Course,” in Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, Jun. 2023. Accessed: Oct. 24, 2023. [Online]. Available: http://dx.doi.org/10.1145/3587102.3588794

[4] M.-F. Wong, S. Guo, C.-N. Hang, S.-W. Ho, and C.-W. Tan, “Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review,” Entropy, vol. 25, no. 6, p. 888, Jun. 2023, doi: 10.3390/e25060888.

[5] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, “Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming,” in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Apr. 2023. Accessed: Oct. 24, 2023. [Online]. Available: http://dx.doi.org/10.1145/3544548.3580919

[6] G. Gupta and A. Batchu, “Generative AI in Outsourced Software Development,” Gartner, Aug. 21, 2023. https://www.gartner.com/en/documents/4657599

[7] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. (Jack) Jiang, “GitHub Copilot AI pair programmer: Asset or Liability?,” Journal of Systems and Software, vol. 203, p. 111734, Sep. 2023, doi: 10.1016/j.jss.2023.111734.

[8] GuardRails, “AI-Assisted Coding: A Double-Edged Sword,” GuardRails, May 26, 2023. https://www.guardrails.io/blog/ai-assisted-coding-a-double-edged-sword/ (accessed Oct. 24, 2023).

[9] Y. Shi, N. Sakib, H. Shahriar, D. Lo, H. Chi, and K. Qian, “AI-Assisted Security: A Step towards Reimagining Software Development for a Safer Future,” in 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Jun. 2023. Accessed: Oct. 24, 2023. [Online]. Available: http://dx.doi.org/10.1109/compsac57700.2023.00142

[10] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, and H. Chen, “Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data,” in 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Jun. 2023. Accessed: Oct. 24, 2023. [Online]. Available: http://dx.doi.org/10.1109/compsac57700.2023.00117

[11] B. K. Deniz, C. Gnanasambandam, M. Harrysson, A. Hussin, and S. Srivastava, “Unleashing developer productivity with generative AI,” McKinsey & Company, Jun. 27, 2023. Accessed: Oct. 24, 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai

[12] “Stack Overflow Developer Survey 2023,” Stack Overflow. https://survey.stackoverflow.co/2023 (accessed Oct. 24, 2023).

[13] M. S., “How to Use GitHub Copilot: Using AI Pair Programmer in 2023,” Hostinger Tutorials, Jun. 22, 2023. https://www.hostinger.com/tutorials/how-to-use-github-copilot (accessed Oct. 24, 2023).

[14] R. Scarlett and M. Mannering, “How to use GitHub Copilot: Prompts, tips, and use cases,” The GitHub Blog, Jun. 20, 2023. https://github.blog/2023-06-20-how-to-write-better-prompts-for-github-copilot/ (accessed Oct. 24, 2023).

[15] A. Roshelova, “Revolutionizing Development: Exploring AI-Powered Code Tools — Copilot and Tabnine,” Medium, Mar. 27, 2023. Accessed: Oct. 24, 2023. [Online]. Available: https://medium.com/@aroshelova.tech/revolutionizing-development-exploring-ai-powered-code-tools-copilot-and-tabnine-6e1a88f1a2d7