Practical guide for using generative AI when learning a programming language

Version: Dec-2025

Raluca–Maria Vedislav*, Dennis Grewe* and W. Daniel Scherz†
* Esslingen University of Applied Sciences, Esslingen, Germany
† University of Applied Sciences Konstanz, Konstanz, Germany

***Abstract*—Artificial intelligence (AI) is currently changing the way we learn and teach—especially when it comes to programming. Tools such as ChatGPT, GitHub Copilot, and Gemini can help you code by making suggestions, finding errors, or explaining concepts. This can make it much easier to get started with a new programming language and help you progress faster.**

But AI can do even more: it can adapt to your learning pace, suggest suitable exercises, or guide you step-by-step through difficult tasks. This creates a learning environment that supports you individually – almost like a personal tutor.

At the same time, the use of AI also brings challenges. If you rely too much on automatic suggestions, your understanding of the basics and your own problem-solving skills can easily fall behind. Instead of mastering a new programming language, you don't really understand the concepts or can't use the language specifically to find a solution to a problem. It is therefore important to use AI consciously and reflectively – as a tool, not as a substitute for your own thinking.

This practical guide shows you how to use AI effectively when learning a programming language. It presents proven methods, tips, and examples that will help you benefit from the strengths of AI without neglecting independent thinking. The aim is to support you in using AI in such a way that you become a better, more creative, and more sustainable programmer in the long term.

***Index Terms*—Generative AI; prompt engineering; LLM; Coding Assistant; Teaching**

I. Introduction

Artificial intelligence (AI) is ubiquitous today – even or especially in academia. Whether writing, researching, or programming, AI tools can make learning and working much easier. At the same time, they raise new questions while using them on a daily basis: How can I use AI effectively? How much support is "okay"? And how can I ensure that I still learn something sustainable by myself?

This practical guide is designed to help you find out exactly that. If you are just starting out with programming, you will find many practical tips here on how to use AI tools in a targeted manner to improve your learning process. In the following sections, you will learn about the opportunities AI offers in everyday programming, where potential pitfalls lie, and how to avoid them. You will also get an overview of the most popular AI tools used in programming and how to make the most of them.

A. Purpose of this guide

AI is changing the way we learn. Tools such as ChatGPT, GitHub Copilot, and AI-based learning platforms can tailor content to your needs and help you solve problems. However, the proper use of AI also raises questions – for example, about data protection and the traceability of your results.

This practical guide is designed to show you how to use AI in programming in a way that really helps you, rather than slowing you down. The guide provides guidance on when AI is useful, when it is better to think for yourself, and how to approach the results critically.

B. Why is this guide important?

AI tools can be incredibly useful for learning—but only if you know how to use them properly. Without clear rules, you may end up relying too heavily on AI or copying content without understanding it.

This guide will help you use AI responsibly and thoughtfully so that you can develop long-term skills such as problem-oriented thinking, understanding programming language concepts, and learning to program in a sustainable way.

1) Risks associated with AI use: The use of AI tools carries certain risks, depending on how you use these tools. The most obvious ones are:

- **Copy & paste without understanding:** It is tempting to simply adopt AI code—but without real understanding, you learn little and it is not sustainable. It's similar to using Google to look something up in an online encyclopedia: it helps us in the short term, but we usually only remember the content to a limited extent.
- **Dependence on AI:** If you rely too heavily on AI suggestions, you will no longer train your own problem-solving thinking. The AI does most of the "thinking," which creates an illusion of progress, even though you haven't actually accomplished much.
- **Data protection and ethics:** AI systems work with large amounts of data – be careful not to share sensitive information (e.g., personal data, data from third parties such as fellow students, company information, internal university information, etc.).

2) Opportunities offered by AI: The use of AI tools promises the following opportunities:

- **Code generation:** AI tools help generate program code or even larger components of a software project. This can save time and provide content support.
- **Debugging:** AI tools can be used specifically to find errors in program code and make suggestions for fixing them.
- **Personalization:** AI can adapt learning content to your pace and level of knowledge, helping you achieve your learning goals.

C. The typical learning curve with AI in programming

When you start using AI tools in programming, you often go through several phases. This is often referred to as the *hype cycle* including phases like:

- At first, you are very enthusiastic—suddenly everything seems easier.
- Then comes disillusionment when you realize that AI cannot do everything.
- Finally, you find a healthy balance: you use AI as a support, but not as a crutch.

Being aware of this cycle already helps you to better assess the use of a tool such as an AI tool.

Note: AI will not replace programmers anytime soon – but those who use AI wisely will have a clear advantage in the long run over those who use the tools incorrectly or not at all.

II. Introduction to AI

AI is no longer a topic for the future—it shapes how we learn, work, and program today. Below, you will find a clear overview of the most important basics, terms, and models so that you know what terms such as "machine learning," "LLM," and "ChatGPT" mean and how you can use them effectively in programming.

A. Types of AI models – and what they mean for you

Not all AI is the same. There are many different approaches—some are trained for specific tasks, while others try to "think" more generally. Table I provides a simplified overview to help you distinguish between the most important concepts.

1) By capabilities: From specialized to hypothetical intelligence.: Narrow AI (also known as "weak AI") is the dominant form today. It specializes in solving clearly defined tasks particularly well—such as generating text, recognizing images, or translating language. Examples include ChatGPT, Siri, and GitHub Copilot. Its strength lies in its high precision and reliability within its area of expertise. Its weakness: it has no real understanding beyond the context and cannot transfer knowledge flexibly.

General AI (also known as "strong AI") strives to think like humans—that is, to learn flexibly, abstract knowledge, and solve complex problems in different domains. Such systems currently exist only in theory. Their development is one of the major research goals of AI, but it also poses ethical, safety, and societal challenges.

Finally, superintelligent AI is a hypothetical concept that describes intelligence that surpasses human capabilities in almost all areas. This scenario is often discussed in futurology, but remains speculative. Its potential strengths—unlimited learning ability, problem-solving skills, efficiency—are also its greatest risks when control and transparency are lacking.

2) By functionality: How AI thinks and learns.: This classification considers how an AI system processes information. *Reactive machines* such as early chess computers (e.g., IBM Deep Blue) act solely on the basis of current inputs and cannot take previous experiences into account. They are reliable and predictable, but not very adaptable.

AI with memory represents a further development: systems learn from past data to improve future decisions. This category includes most modern machine learning models. Their advantage lies in their ability to recognize patterns over time; however, their weakness is their heavy dependence on the quality and representativeness of the training data.

Theory of Mind models represent a theoretical intermediate step. They are designed not only to process data, but also to understand emotions, intentions, and social dynamics. Currently, only research prototypes exist for this purpose. However, a fully functional self-aware AI with its own consciousness or self-awareness does not yet exist. Both concepts illustrate how far-reaching the vision of AI is beyond today's applications.

3) By technology: The building blocks of modern AI systems.: The technological classification describes the principles on which AI systems are based. *Machine learning (ML)* forms the foundation of many applications. It uses statistical methods to recognize patterns in data. ML is particularly effective when large amounts of data are available, but can be susceptible to bias.

This categorization helps you better understand AI systems—from simple voice assistants to complex learning models. Most systems in use today, including ChatGPT and GitHub Copilot, belong to what is known as *narrow AI*.

B. Important subfields of AI

AI consists of many subfields that complement each other. Some focus on learning from data, others on language, images, or decision-making. The following table gives you an overview of what they stand for and where you encounter them in everyday life.

Classification	Type	Description
By capabilities	Narrow AI (Weak AI)	Can only solve specific tasks (e.g., chatbots, code assistants)
	General AI (Strong AI)	Goal: human-like thinking ability (still a long way off)
	Super-intelligent AI	Hypothetical: would surpass humans
By functionality	Reactive machines	Only respond to current inputs
	AI with memory	Use past data for better results
	Theory of Mind	Could recognize feelings or intentions (currently theoretical)
	Self-aware AI	Does not (yet) exist
By technology	Machine learning (ML)	Systems that learn from data
	Deep learning	Complex neural networks with many layers
	Natural language processing (NLP)	Understands and processes human language
	Robotics	Intelligent machine control and interaction
	Computer Vision	Recognizes objects and patterns in images or videos
	Expert Systems	Makes decisions based on predefined rules

Table I
CLASSIFICATION APPROACHES IN ARTIFICIAL INTELLIGENCE ACCORDING TO **russell2021artificial**

Subfield	Description	Typical applications
Machine learning (ML)	Systems learn independently from data	Image recognition, spam filters, predictions
Deep learning (ML)	Uses neural networks to recognize complex patterns	Speech and image recognition, chatbots
Natural language processing (NLP)	AI understands and generates language	Chatbots, translations, text analysis
Expert systems	Make decisions based on rules	Medicine, finance, support systems

Table II
OVERVIEW OF KEY SUBFIELDS OF ARTIFICIAL INTELLIGENCE

Note: Many modern AI applications that you use every day – from translation services to recommendation systems – are based on a combination of these subfields.

C. Large Language Models (LLMs)

Chatbots such as ChatGPT, Anthropic Claude, and Google Gemini are based on so-called large language models (LLMs). These models are trained to understand natural language, recognize patterns, and generate new texts. They are based on billions of text examples from the internet, books, and other sources, and typically use the Transformer approach. From a technical perspective, these are statistical, probabilistic models: they estimate the probability distribution $p(w_t \mid w_{<t})$ for the next token/word from the previous context and generate outputs auto-regressively, i.e., token by token. The output is created by repeatedly selecting (deterministically or stochastically) the next token. In other words, such models estimate the next word in a sentence based on the text generated so far and the information they have received from the user as input.

D. Overview of current LLMs

Below you will find an overview of what are probably the most widely used LLMs today. New ones are added regularly, so this list is not guaranteed to be complete.

1) GPT (Generative Pre-Trained Transformer): The Generative Pre-trained Transformer, known as GPT, is a model developed by OpenAI that is ideal for beginners in programming. It offers precise explanations and can generate, analyze, and improve programming code [1]. The model has been continuously developed since its release in November 2022, with each new model offering new features and improved performance. The free version includes the more compact models such as GPT-4o mini and GPT-4.1 mini, although use of this version may often have limitations in terms of the number of requests or supported features.

In addition, there are two other paid packages [2]:

- **Plus subscription:** Provides extended access to more powerful models such as GPT-4, GPT-4.1, GPT-5 with additional features and without the limitations of the free version.
- **Pro subscription:** Provides access to the latest and most powerful models, including GPT-4o, GPT-4.1, GPT-5 with maximum functionality and priority usage.

Table III provides an overview of the current models with the date of publication, a brief description, and the package in which they are included.

Model name	Release date	Keywords	Available in pricing plan
GPT-4	March 2023	Multimodal, Text & Image	Plus, Pro, Team, Enterprise
GPT-4o	May 2024	Language, Image, Audio	Plus, Pro, Team, Enterprise
GPT-4o mini	July 2024	Fast, affordable	Free, Plus, Pro, Team, Enterprise
GPT-4.1	April 2025	Improved programming assistance	Plus, Pro, Team, Enterprise
GPT-4.1 mini	May 2025	Compact and efficient	Free, Plus, Pro, Team, Enterprise
GPT-5	August 2025	Merger of "thinking" and "non-thinking" models	Free, Plus, Pro, Team, Enterprise

Table III
OVERVIEW OF GPT MODELS ACCORDING TO **openai2025models**

2) Anthropic's Claude: Claude is an advanced AI model from Anthropic that was developed with the goal of creating responsible and ethical AI systems. Claude stands out for its ability to serve as a supportive tool in educational institutions and other areas, with a particular emphasis on data protection and academic integrity. For example, no student data is used to train the generative models [3].

A comprehensive analysis of over half a million anonymized conversations between students and Claude has shown that the AI is mainly used in the areas of creating and improving learning materials (39.3% of conversations) and technical explanations and problem solving (33.5% of conversations). This includes creating practice questions, editing essays, summarizing academic content, debugging programming tasks, and solving mathematical problems [4], [5].

In February 2025, Claude 3.7 Sonnet was introduced. Its special feature: users can choose whether they prefer a quick response or a version in which the model thinks longer and outputs step-by-step solutions before providing an answer [5], [6]. With each new version of Claude, the model's ability to generate complex programming tasks and program code increases.

3) GitHub Copilot: GitHub Copilot was developed by Microsoft and was intended to serve as an extension of the IntelliSense concept. IntelliSense is an integrated AI whose suggestions are limited to names of methods or properties. GitHub Copilot can take suggestions for loops, conditions, or entire methods [7]. For example, when creating a new function, the code can be generated based on the rest of the file. Copilot also has a chat integration that allows communication directly in the IDE.

There are a few tips and tricks for getting GitHub Copilot to provide better suggestions. Functions should have meaningful names. It may also be useful to add a comment before the function that briefly describes what the function is supposed to do. Conversely, GitHub Copilot can also help with creating comments. If there are other files that are relevant to the project, they should also be opened so that Copilot can take them into account [7].

GitHub Copilot has three different pricing packages, each working with different models, such as Anthropic Claude 3.5 Sonnet, OpenAI GPT-4.1, or the newer models from Anthropic's Claude or OpenAI GPT models. The free package includes up to 50 chat interactions per month and access to the models from Anthropic Claude, Google Gemini, and OpenAI GPTs. GitHub Copilot can be integrated into several IDEs, such as Visual Studio Code, Visual Studio, JetBrainsIDE, Eclipse, Xcode, or Azure Data Studio [8].

4) Other models and tools in the programming context: In addition to large multimodal language models such as GPT or Claude, there is a growing number of specialized AI models and development environments that are specifically geared toward programming.

Cursor is a modern, AI-powered development environment based on OpenAI and Anthropic models, but heavily optimized for the *development workflow*. Cursor offers context-sensitive code suggestions, refactoring aids, and allows you to discuss code changes directly in the editor with a chatbot. This provides intuitive access, especially for learners, as AI suggestions are visible in the direct work context [9].

Gemini Code, developed by Google DeepMind, combines language understanding with multimodal inputs and is available via the *Google Colab* platform or in IDEs such as *VS Code*. It offers learners interactive support through contextual explanations, test generation, and code commenting [10].

Code Llama (Meta AI) is a derivative of LLaMA specialized for programming tasks. It is freely available and used in many open-source IDE integrations. It is particularly interesting for teaching purposes as it can be run locally, thus also allowing for *privacy-friendly scenarios* in educational environments [11].

StarCoder and **SantaCoder** (Hugging Face and ServiceNow) are open alternatives with a focus on reproducible research and education. They support numerous programming languages and can be self-hosted in university labs to demonstrate to students how to build their own code models [12].

Commercial assistants such as **Amazon CodeWhisperer** or **Tabnine**, on the other hand, focus on integration into existing enterprise IDE ecosystems. They offer stable, production-ready support, but are less transparent in terms of training data and model architecture [13], [14].

In summary, the systems differ primarily in terms of accessibility, degree of specialization, and transparency:

- **Proprietary systems** (e.g., Copilot, CodeWhisperer, Gemini) are powerful but come at a cost and are less open.
- **Open models** (e.g., Code Llama, StarCoder) are ideal for teaching, research, and data protection scenarios.
- **Integrated environments** such as Cursor or Replit AI create new learning paradigms in which AI becomes a direct part of the programming process.

III. Possible uses of AI in programming

Generative AI tools such as ChatGPT, Claude, or GitHub Copilot offer a wide range of support options in the programming learning process. It depends on what you use the tools for and how you use them. The following sections provide an overview of typical areas of application, specific tips for use, and point out what you should pay attention to when formulating prompts, i.e., the commands and instructions between you and the AI.

A. Typical areas of application

As already described in Chapter I, AI systems can be used in the context of programming in the following situations:

- **Code generation:** Creation of program code such as functions, code blocks, or even larger, more complex code passages and files based on a task description.
- **Debugging:** Analysis and correction of error messages or faulty behavior.
- **Concept explanation:** Explanation of programming concepts and contexts such as arrays, loops, or pointers.
- **Syntax help:** Looking up commands, parameters, or function signatures.
- **Structure suggestions:** Support in building a program structure or planning the solution path.
- **Error detection and correction:** Identification and resolution of typical runtime or compilation errors.

B. Tips for effective use

When using AI tools, there are a few things to keep in mind that can positively or negatively influence the result. When interacting with the AI tool, known as prompting, the following aspects should be considered to ensure a positive result.

- **Provide context:** Describe the context in which you interact with the AI, e.g.: Which programming language do you use? What exactly is your goal with the interaction: support in problem solving, error analysis, required explanation, etc.? You should describe what you expect from the AI and what role you see the AI playing in your interaction (e.g., explanatory aid, tutor, etc.). The more precisely you describe the context, the better the AI can help you.
- **Be specific rather than general:** The clearer your entire request is, the more helpful the answer will be. If you only describe what you want in general terms, the result may be less helpful, similar to a Google search.
- **Include error messages:** When debugging, provide the exact error text (possibly copy the error message) including the affected code. The more information the AI has, the better it can advise you on error analysis and correction.
- **Formulate your goal:** Describe in detail how you would like the AI to respond, e.g., whether you would like an explanation, improvement, or example.

C. Warning: No substitute for independent thinking

The possibilities presented here make it clear that AI tools can be a helpful addition, but they do not replace the active learning process. Simply adopting solutions without reflection does not lead to the acquisition of skills in the long term. It remains essential to engage with the problem, the solution, and the functionality of the code yourself.

IV. Tips for using AI in programming

Before working with AI tools, users should be aware of some basic guidelines and realistically assess the capabilities of AI in order to minimize the risk of incorrect or misleading results.

A. AI can hallucinate

AI-powered development tools can provide impressive support, often delivering fast, well-formulated solutions to everyday programming tasks. But they also have their weaknesses: errors often occur, especially with lesser-known programming languages, new libraries, or very specific questions. What is particularly problematic is that AI tools also present incorrect answers with confidence and linguistic persuasiveness. Since they often "hallucinate"[1], i.e., provide plausible-sounding but invented information, critical questioning is essential [7].

B. AI results are not reproducible

Even with identical prompt input [2] and the same model version, the AI can produce different results each time it is executed. These outputs can vary in quality, sometimes they are more helpful, sometimes less precise [7].

C. Colloquial language is also okay

In this guideline, we will look at how to best use AI and how to best communicate with AI. In principle, it is important that the AI receives clearly formulated instructions so that it knows what the user specifically wants. However, it is also acceptable for the prompts to be described in colloquial language, as long as it is clear what the AI is supposed to do [7].

D. Responsibility

The user is solely responsible for what happens with the output of the AI. Information should always be critically questioned and analyzed before it is adopted or reused [7].

V. Prompt Engineering

Prompt engineering is a field that deals with the creation, optimization, and refinement of prompts [17]. Prompts form the basis of interaction between humans and AI [18] and can be simply defined as concrete instructions or questions given to a language model to control its behavior and generate desired outputs [19] (e.g., in the form of text in a chat window). The quality of the outputs generated by AI depends heavily on the wording of the prompt entered by the user [17].

[1] In AI, "hallucinate" refers to the generation of plausible-sounding but factually incorrect or invented content [15].

[2] A prompt is a targeted input, e.g., an instruction that prompts an AI system to produce a specific response or output [16].

A. The elements of a prompt

An effective prompt typically consists of four elements [19] (cf. Figure 1):

- **Instruction**: The instruction describes the specific task that the model is to perform. It guides the model's behavior toward the desired result.
- **Context:** The context provides the model with additional background information or framework conditions that are relevant for processing the task. Context helps the model to better classify the task and generate more appropriate responses. This includes, for example, definitions, examples, information about the target group, or thematic restrictions.
- **Input data**. The input data is the specific content that the model is supposed to process. It is at the center of the prompt and defines what the response should refer to. Input data can be, for example, a question, a section of text, a code snippet, or a problem statement.
- **Output indicator:** The output indicator describes the form in which the result should be delivered. It specifies whether, for example, a short answer, a detailed explanation, a list, a table, or a certain tone of voice is desired.

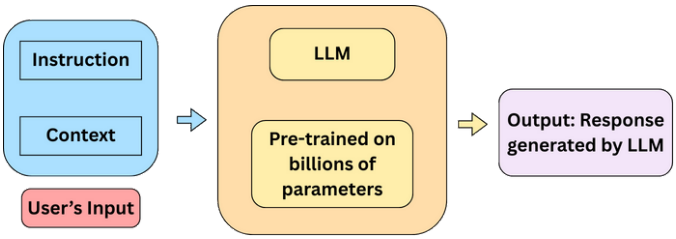

Figure 1. The components of a prompt based on [20]

B. Methods in prompt engineering

In the field of prompt engineering, numerous methods are known for specifically controlling the behavior of pre-trained models for specific tasks [20]. A study by Sahoo P. et al. [20] from 2024 presents an overview of more than 29 methods, divided into different categories. The following section presents a selection of these methods and briefly explains each one using an example.

1) Zero-Shot Prompting: With this technique, the model only receives a task description—without additional examples. It solves the task solely through its knowledge of language, meaning, and logical thinking learned in pre-training [18], [21]. Unlike classical ML methods, which rely on datasets with predefined correct answers (so-called labels), zero-shot prompting utilizes the generalization ability of large language models. This allows tasks such as text classification, logical reasoning, or text generation to be solved without domain-specific training [22].

However, the quality of the output depends heavily on the clarity and structure of the prompt. Imprecise instructions can easily lead to incorrect or inappropriate results [23], so a well-designed prompt is crucial [24].

Figure 2 shows an example of zero-shot prompting with the AI's response. It can be seen that the AI was not given an example, only the instruction.

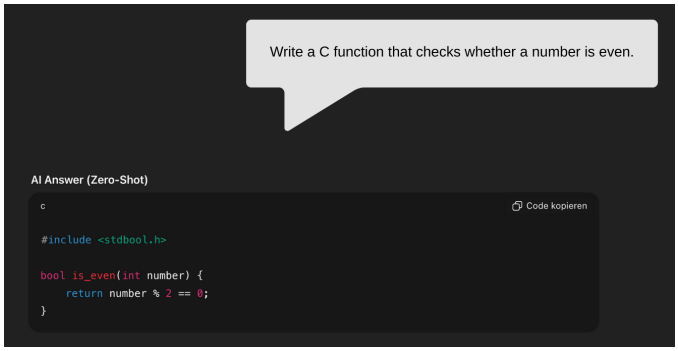

Figure 2. Example of zero-shot prompting

2) Few-Shot Prompting: In few-shot prompting, a few selected input-output examples are provided to the AI model as part of the input. These serve to clarify the structure and requirements of a task and thus specifically control the model's response behavior [18], [20]. In contrast to zero-shot prompting, where no examples are used, it has been shown that even a few high-quality demonstrations can significantly improve the model's performance on complex tasks [22].

However, one disadvantage of few-shot prompting is the increased space requirement: the example data must be embedded within the prompt, which can lead to limitations depending on the length of the examples and the actual text input, especially for models with a limited context window [20].

Figure 3 shows an example of zero-shot prompting with the AI's response. It can be seen that the AI was given two examples, which the model used to recognize structure and style.

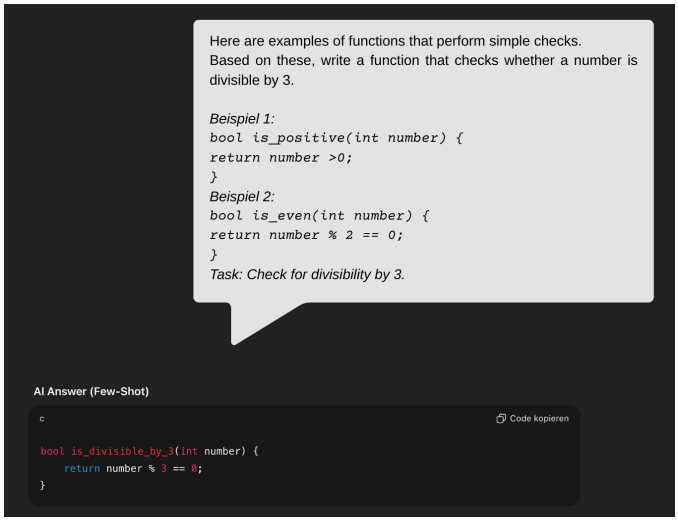

Figure 3. Example of few-shot prompting

3) Chain-of-Thought (CoT): This prompting was introduced for tasks that require logical or multi-step thinking,

such as math or everyday logic problems, to encourage large language models to think in a more structured and step-by-step manner. The advantages of this approach are that a large problem is broken down into several smaller problems. This allows more computing power to be used for tasks that require more complex thinking. Secondly, a “chain of thought” provides insight into the behavior of the model and can show how it arrived at a particular answer. This allows errors in the thought process to be identified and corrected [20], [25].

Figure 4 shows an example of chain-of-thought prompting with the AI’s response. The steps in the comment show the AI’s thought process.

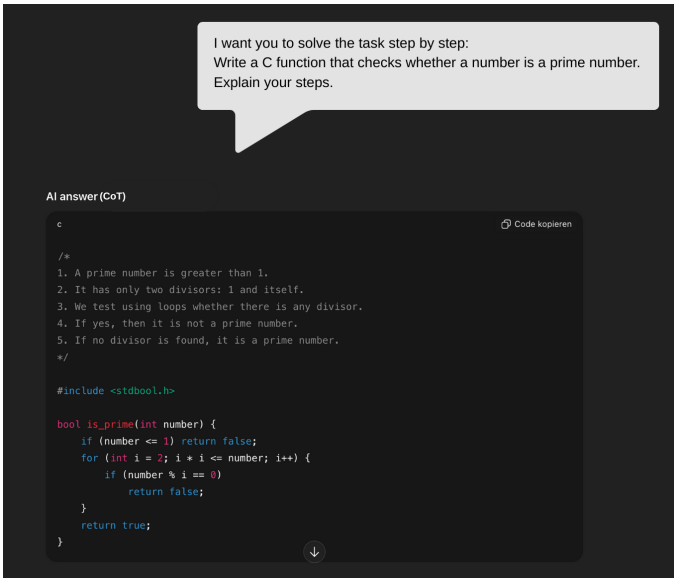

Figure 4. Example of chain-of-thought prompting

4) Scratchpad Prompting: Scratchpad prompting is an approach that allows the model to take notes on complex, multi-step tasks before arriving at the final solution. Instead of asking the model for the final answer immediately, it is encouraged to explicitly write down its intermediate steps [20], similar to how humans use a notepad ("scratchpad") when doing calculations. This technique increases the traceability of the results and reduces errors, especially in algorithmic or mathematical questions [26].

Figure 5 shows an example of scratchpad prompting with the AI’s answer. The output of intermediate steps and explanatory comments simulate a "scratchpad".

C. Challenges in prompt engineering

Despite numerous methods in prompt engineering, there are still a number of challenges when dealing with large language models. Prompts play an important role in controlling AI, but they are susceptible to various pitfalls: ambiguous wording, over-fitting to training data, or unrealistic expectations. It is important that users are aware of these challenges and actively incorporate them into the design of prompts. The following section presents key challenges in prompt engineering with corresponding examples [19], [20].

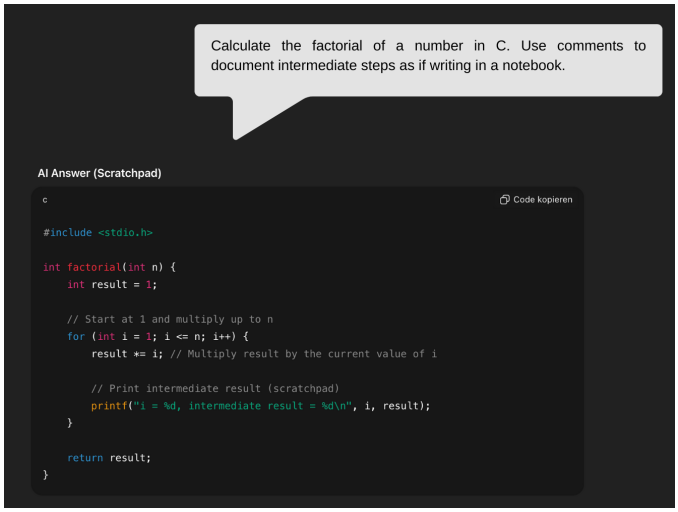

Figure 5. Example of a bad prompt on ambiguity

1) Ambiguity: A common problem in prompt engineering is ambiguity, i.e., the vagueness or lack of clarity of a prompt. If instructions are formulated too generally or unspecifically, this often leads to superficial or imprecise responses from the AI [19], [20].

Figure 6 shows an example of an imprecise prompt. This prompt leaves open what type of list is meant (e.g., array or linked list), what data is to be processed (e.g., integers, strings), and what “process” specifically means (e.g., sort, search, calculate).

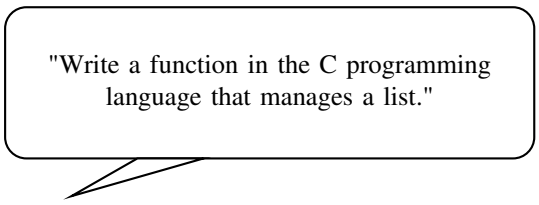

Figure 6. Example of a bad prompt on ambiguity

Figure 7 shows a revised, improved prompt. The revised version provides clear instructions on the data type, the specific task, and the expected output. This enables the language model to generate a suitable and functional solution.

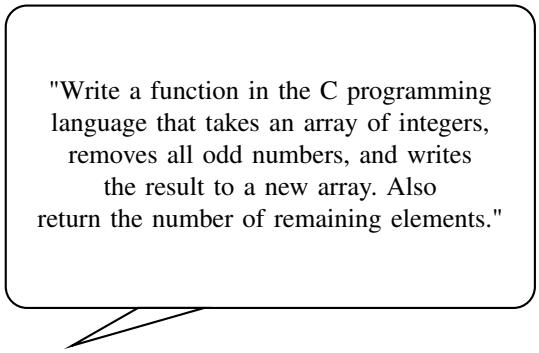

Figure 7. Example of a revised prompt on ambiguity

2) Over-fitting: Another problem in prompt engineering is over-fitting, i.e., the over-adaptation of a prompt to a very specific example. A prompt can be so specific that the AI generates a suitable answer, but this answer only applies to this exact example and cannot be transferred to similar questions [19].

Figure 8 shows an example of an over-fitted prompt. Although this prompt performs the function correctly, the task is so heavily tailored to a specific sequence of numbers that the solution is unusable for other cases. As a result, the AI does not "learn" anything about the underlying logic or a more general principle.

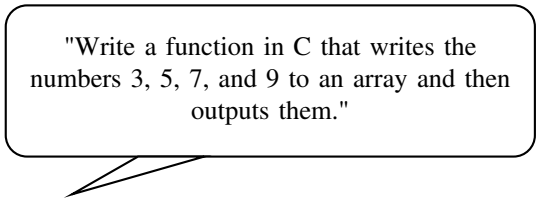

Figure 8. Example of an over-fitted prompt

Figure 9 shows an example of a generally formulated prompt. This prompt requests the same functionality, but allows for broader application and promotes the model's ability to generalize.

A prompt that focuses too strongly on one example may lead to the desired output in the short term, but it prevents the model from responding flexibly or creatively to similar tasks [19].

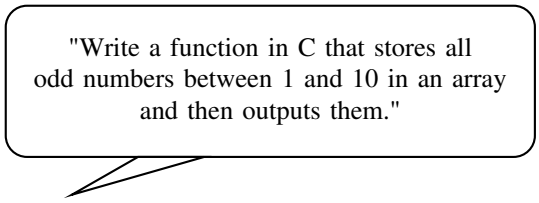

Figure 9. Example of a generally well formulated prompt

3) Unrealistic expectations of AI capabilities: A common mistake in prompt engineering is the assumption that AI models such as ChatGPT are capable of delivering perfect or deterministically correct solutions for any task. Language models are not omniscient systems—they generate responses based on patterns from training data, not on real-world knowledge or deterministic logic. In this context, so-called hallucinations can also occur—i.e., seemingly plausible but factually incorrect or invented outputs [19].

Figure 10 shows an example of a prompt with overly high expectations of AI. This prompt assumes that AI is capable of always selecting the optimal sorting algorithm for any amount and distribution of data.

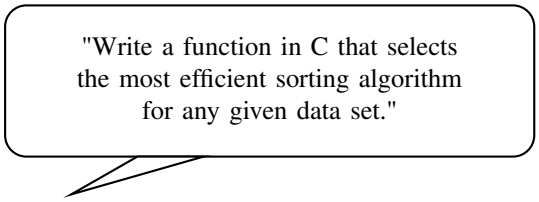

Figure 10. Example of a bad prompt with unrealistic expectations of the AI

Figure 11 shows the revised requirement for the AI. This prompt sets a concrete, realistically achievable task that the model can reliably accomplish.

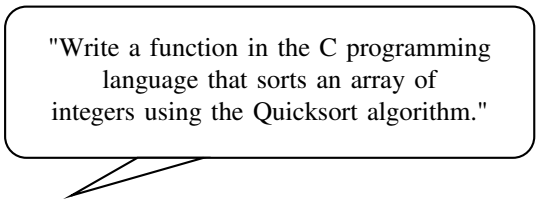

Figure 11. Example of a revised prompt with realistic expectations of the AI

Instead of placing excessive expectations on the model, prompts should be formulated in such a way that they are based on the realistic capabilities of the AI. It is also important to always critically reflect on responses, because hallucinations can cause the AI to present incorrect or fabricated content with great conviction. Technical review and additional research therefore remain essential, especially for technical applications.

VI. General Best Practices

This chapter presents general practical tips designed to improve AI's understanding of instructions.

1) **Foreign Language vs. English** Although AI models such as ChatGPT also understand foreign-language prompts, such as German, it is recommended to ask questions in English whenever possible. The reason: Much of the training material is in English, which increases the likelihood of more accurate and higher-quality responses, especially for complex or technical questions. In addition, English is the standard language for many software projects and is a prerequisite for using many other AI models that have limited support for other languages [7].

2) **Additional information** Before asking the actual question, the technical context should be specified—for example, the programming language, libraries, tools, or development environment used. This allows for a more targeted and appropriate response [7], [19].
3) **Define constraints** Important additional information should also be specified, e.g., whether compact code is desired, whether as few libraries as possible should be used, whether security aspects need to be considered, or whether there are special requirements for efficiency or recursion [7].
4) **Second attempt** If the AI delivers a completely different result than expected on the first attempt, or is on the "wrong track," a second attempt may help. To do this, simply try a new prompt.
5) **New session when changing topics** Since the models remember the information from a chat history, a new session should be started when changing topics so that the old information is not transferred to the new topic [7].
6) **Clarity and precision** The clearer and more precise a prompt is formulated, the more targeted and relevant the AI's response will be. General queries often lead to unspecific results. Specific details and context reduce room for interpretation and improve the quality of the output [27].

VII. Conclusion

The use of AI tools, such as LLMs or coding agents, can be a valuable aid in learning to program, especially when it comes to understanding concepts, solving specific programming problems, and reflecting on one's own solutions. In order to make the most of this potential, it is essential to use AI outputs in a conscious and reflective manner.

This practical guide shows that understanding basic concepts such as how LLMs work, how to construct effective prompts, and knowledge of the strengths and limitations of AI are crucial for achieving high-quality results. At the same time, it is clear that AI systems are not error-free expert systems – they hallucinate, do not always deliver reproducible results, and do not replace independent thinking or learning.

The goal should therefore be to understand AI as a tool: used correctly, it can trigger thought processes, suggest solutions, and make learning more efficient. Used incorrectly or uncritically, however, there is a risk of misunderstandings, faulty code, or overestimating machine responses.

By following the advice in this practical guide, defining clear prompts, and critically reflecting on the results, the use of AI in programming education can be made not only more effective but also more sustainable.

References

[1] G. Yenduri et al., *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*, en, May 2023. Accessed: May 18, 2025. [Online]. Available: https://arxiv.org/abs/2305.10435v2.

[2] *Models - OpenAI API*, de-DE. Accessed: May 18, 2025. [Online]. Available: https://platform.openai.com.

[3] *Claude for Education | Partnering with Universities on Responsible AI \ Anthropic*, en. Accessed: May 18, 2025. [Online]. Available: https://www.anthropic.com/education.

[4] *Anthropic Education Report: How University Students Use Claude*, en. Accessed: May 18, 2025. [Online]. Available: https://www.anthropic.com/news/anthropic-education-report-how-university-students-use-claude.

[5] *Claude 3.7 Sonnet System Card*, en. Accessed: May 18, 2025. [Online]. Available: https://www.semanticscholar.org/paper/Claude-3.7-Sonnet-System-Card/9ff93dfa8f445c932415d335c88852ef47f1201e.

[6] *Claude 3.7 Sonnet and Claude Code*, en. Accessed: May 18, 2025. [Online]. Available: https://www.anthropic.com/news/claude-3-7-sonnet.

[7] M. Kofler, B. Öggl, and S. Springer, *Coding mit KI: das Praxisbuch für die Softwareentwicklung*, ger, 1. Auflage. Bonn: Rheinwerk Computing, 2025, ISBN: 978-3-367-10346-1.

[8] *Take flight with GitHub Copilot*. Accessed: May 19, 2025. [Online]. Available: https://github.com/features/copilot/plans.

[9] C. AI, *Cursor – the ai-powered code editor*, https://www.cursor.com/, Accessed: 2025-10-06, 2025.

[10] G. DeepMind, *Introducing gemini for developers*, https://deepmind.google/technologies/gemini/, Accessed: 2025-10-06, 2025.

[11] M. AI, *Code llama: Open foundation models for code*, https://ai.meta.com/research/publications/code-llama/, Accessed: 2025-10-06, 2023.

[12] H. Face and S. Research, *Starcoder: A large language model for code*, https://huggingface.co/bigcode/starcoder, Accessed: 2025-10-06, 2023.

[13] A. W. Services, *Amazon codewhisperer – ai coding companion*, https://aws.amazon.com/codewhisperer/, Accessed: 2025-10-06, 2024.

[14] T. Ltd., *Tabnine – the ai assistant for developers*, https://www.tabnine.com/, Accessed: 2025-10-06, 2024.

[15] D. J. Siebert, *Halluzinationen von generativer KI und großen Sprachmodellen (LLMs) - Blog des Fraunhofer IESE*, de, Sep. 2024. Accessed: May 17, 2025. [Online]. Available: https://www.iese.fraunhofer.de/blog/halluzinationen-generative-ki-llm/.

[16] S. Burgermeister, *Was ist Prompting | IU Akademie Blog*, de, Nov. 2023. Accessed: May 19, 2025. [Online]. Available: https://www.iu-akademie.de/blog/was-ist-prompting/.

[17] S. Arvidsson and J. Axell, "Prompt engineering guidelines for LLMs in Requirements Engineering," eng, Aug. 2023, Accepted: 2023-08-07T08:59:27Z. Accessed: May 16, 2025. [Online]. Available: https://gupea.ub.gu.se/handle/2077/77967.

[18] C. I. Chang, "A Survey of Techniques, Key Components, Strategies, Challenges, and Student Perspectives on Prompt Engineering for Large Language Models (LLMs) in Education," en, Mar. 2025, Publisher:

Preprints. DOI: 10.20944/preprints202503.1808.v1. Accessed: May 16, 2025.

[19] L. Giray, "Prompt Engineering with ChatGPT: A Guide for Academic Writers," en, *Annals of Biomedical Engineering*, vol. 51, no. 12, pp. 2629–2633, Dec. 2023, ISSN: 1573-9686. DOI: 10.1007/s10439-023-03272-4. Accessed: May 16, 2025.

[20] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*, en, Feb. 2024. Accessed: May 17, 2025. [Online]. Available: https://arxiv.org/abs/2402.07927v2.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[22] T. Brown et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[23] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large Language Models are Zero-Shot Reasoners*, en, May 2022. Accessed: May 17, 2025. [Online]. Available: https://arxiv.org/abs/2205.11916v4.

[24] T. Debnath, M. N. A. Siddiky, M. E. Rahman, P. Das, and A. K. Guha, "A Comprehensive Survey of Prompt Engineering Techniques in Large Language Models," *TechRxiv*, 2025.

[25] J. Wei et al., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, en, Jan. 2022. Accessed: May 17, 2025. [Online]. Available: https://arxiv.org/abs/2201.11903v6.

[26] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "STaR: Bootstrapping Reasoning With Reasoning," en, Oct. 2022. Accessed: May 17, 2025. [Online]. Available: https://openreview.net/forum?id=_3ELRdg2sgI.

[27] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, "Unleashing the potential of prompt engineering for large language models," English, *Patterns*, vol. 0, no. 0, May 2025, Publisher: Elsevier, ISSN: 2666-3899. DOI: 10.1016/j.patter.2025.101260. Accessed: May 19, 2025.