# Exploring Conversational Programming through the CHOP Paradigm and Artificial Intelligence

Alin ZAMFIROIU, Costinela-Beatrice PĂȘTINICĂ,
Cătălin CRĂCIUN, Claudia CIUTACU, Theodor LUCA
Bucharest University of Economic Studies
alin.zamfiroiu@csie.ase.ro, pastinicacostinela22@stud.ase.ro, craciuncatalin20@stud.ase.ro,
ciutacuclaudia20@stud.ase.ro, theodorluca12@gmail.com

*Chat-Oriented Programming (CHOP) is an emerging conceptual approach to programming, focused on the interaction between programmers and conversational interfaces based on artificial intelligence (AI). Unlike the traditional paradigm, centered on writing code directly, CHOP proposes a software development model in which application components are built, modified, and analyzed through natural dialogues with AI agents. The goal of this paradigm is to reduce technical barriers and transform the programming process into a collaborative, conversational, and more accessible act. The paper explores techniques for automatic code generation assisted by artificial intelligence, highlighting a series of tools and platforms relevant to the Chat-Oriented Programming (CHOP) paradigm. It also analyzes the benefits of this approach in optimizing the software development process.*
***Keywords:*** *Chat, Programming, CHOP, AI code, Software development*
**DOI:** 10.24818/issn14531305/29.2.2025.01

## 1 Introduction

Chat-Oriented Programming (CHOP) is a conceptual approach to programming that focuses on the interaction of programmers with chat-based interfaces and artificial intelligence (AI) agents. CHOP aims to shift the programming paradigm from a traditional code-centric approach to one that involves verbal interactions between the programmer and the artificial intelligence agent being used.

CHOP allows programmers to interact with in a chat environment in natural or pre-defined language. Allowing for a simpler and easier way to program or control software applications, the system interprets these inputs and executes the respective actions.

According to [1] the programming has more phases or more paradigms regarding to the source of the information for the programmers, Figure 1:

- BOOP (Book Oriented Programming) refers to the phase in which programmers relied on books to improve their programming techniques and expand their knowledge. This period corresponds to a time when the internet was not yet well developed and, making books the primary source of information - though obtaining them was often difficult.
- GOOP (Google Oriented Programming) is the phase represented by the programmers that have used the Google Search Engine to find code examples or solutions to problems during the programming process. This phase starts with the rise of the internet, as information previously found in books became accessible online.
- SOOP (Stack Overflow Oriented Programming) is represented by the phase when the Stack Overflow platform became widely used for solving various coding issues. In this phase, programmers search the platform to check if others have encountered the same problem and whether solutions have already been provided. This approach helps save time by avoiding the need to browse multiple forums or official documentation sites.
- CHOP (Chat Oriented Programming) represents the actual phase in which the programmers use artificial intelligence and chat-based agents to assist with coding tasks. When a programmer encounters a problem, he asks the AI agent for help, and the agent quickly searches

for relevant information and solutions. This approach saves time by reducing the need to search through Stack Overflow or any other platforms, as it was common in previous phases.

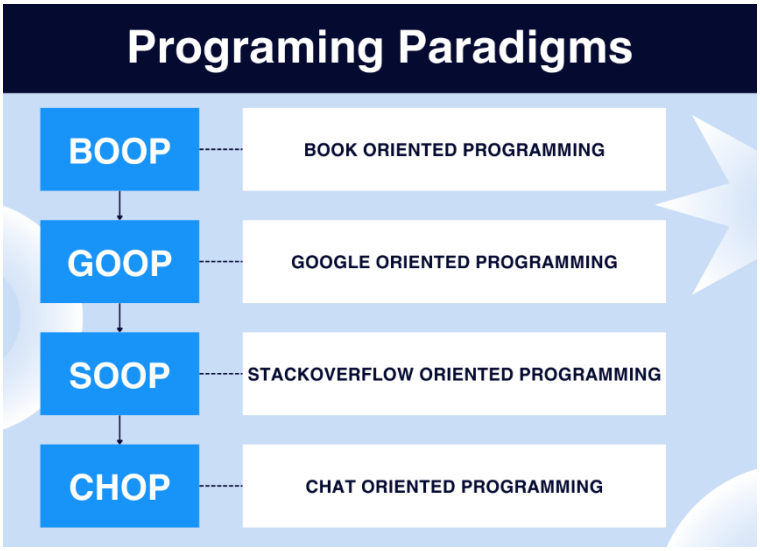


**Fig. 1.** Evolution of programming paradigms [1]

In this way, the programming paradigm continues to evolve based on the technologies programmers use for assistance. These paradigms are applicable regardless of the programming language being used.

## 2 Background on AI code generation

As AI-driven code generation has advanced over the years, it continues to keep up with the changes in its general technology. In the beginning, machine learning systems only relied on provided templates and checklists without being able to offer much help. Later, AI systems began learning patterns and allowed features such as intelligent autocompletion and bug finding to coexist. The introduction of deep learning and neural networks allowed AI to create entire code snippets, as well as restructure complicated programs and provide optimized suggestions based on previously learned data. The integration of cloud services provided a new level of collaboration and computational power that improves development using AI. With the rise of AI models like ChatGPT, chatbots have become much more advanced. Now, they can interpret, generate, and improve code in real time, making them more like interactive learning partners rather than just basic assistants [2].

The roots of AI in code generation trace back to early computational models, notably The Logic Theorist, created in 1956 by Allen Newell, Herbert Simon and Cliff Shaw, was one of the first examples of artificial intelligence (AI) and an early step toward AI-powered code generation. It was designed to mimic how people solve problems and could prove mathematical theorems from "Principia Mathematica", an important book on logic. According to a study by [3], this unique program demonstrated how machines were able to think for themselves, and served as the foundation for future writing and coding AI tools.

Another tool that was important for the development of AI was called Eliza. Eliza was one of the first AI programs to use natural language processing (NLP), a technology that forms the foundation of modern AI chatbots and virtual assistants, Figure 2. The program functioned by detecting key words in user input and generating responses based on predefined patterns. This made it appear as though ELIZA could understand emotions and engage in meaningful conversations. However, its responses were based purely on syntax rather than true comprehension, Eliza followed structured rules rather than actually understanding language or context [4].

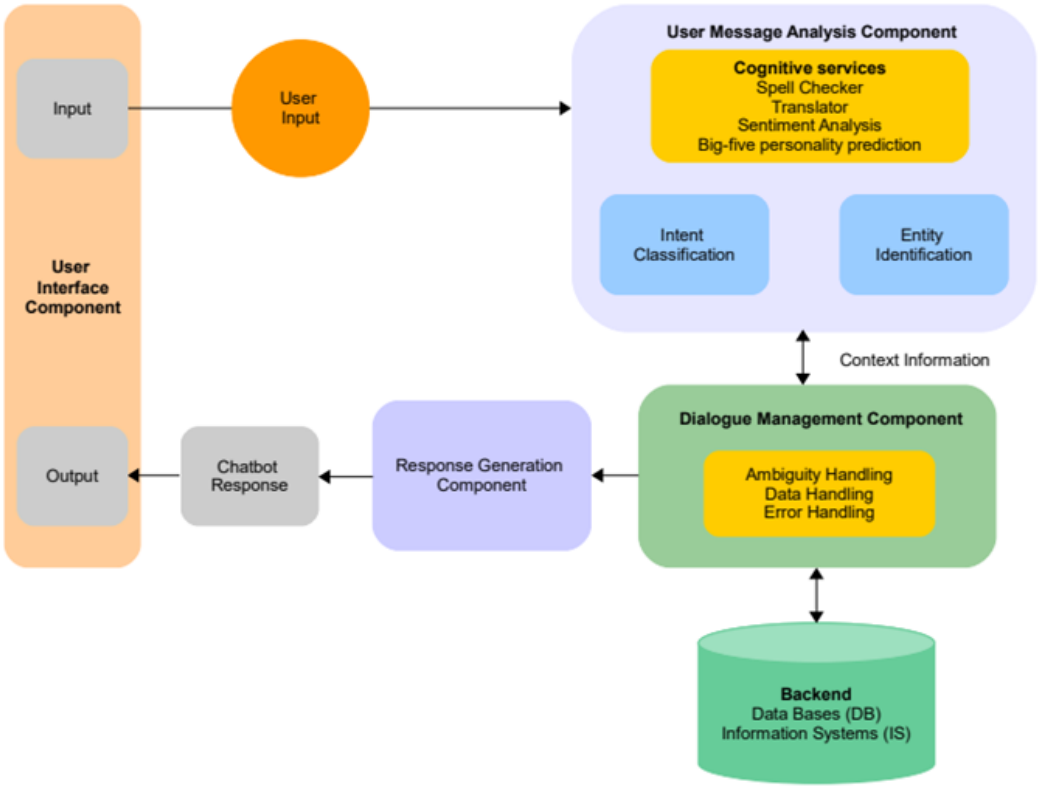


**Fig. 2.** General chatbot structure

Despite the promising beginnings, the evolution of AI came to a halt for the first time due to two major factors, referred to as the First AI Winter (1970-1980) and Second AI Winter (1980 - 1990). This emerged when the early attempts at AI were continuously failing to solve any real-life problems. This cycle of funding AI systems was put on hold, especially after the Lighthill Report AI review came out in 1973. [5]. Afterwards, interest in AI systems was notable in the 1980's due to the introduction of expert systems, which turned out to be overpriced and inadequate. As a result, confidence in AI declined, research funding dried up, and many AI companies went bankrupt.

In the 1990s, researchers began to see that new algorithms could help build deeper neural networks to handle tougher problems. One key development was the Long Short-Term Memory (LSTM) network, which solved the problem of vanishing gradients and allowed computers to better understand sequences of data. Then, in 2012, a major breakthrough occurred when scientists Krizhevsky, Sutskever and Hinton combined huge datasets (like ImageNet) with powerful GPUs. This combination greatly improved the ability of deep neural networks to classify images, a milestone marked by the success of AlexNet. Although many see this as the start of the deep learning era, these achievements were built on decades of earlier work in algorithm development.

The history of AI code generation is a chronicle of dramatic developments that have fundamentally altered the processes of software creation. Microsoft launched IntelliCode as part of Visual Studio in 2018, an AI-powered coding tool which advanced productivity by giving context-sensitive code suggestions during the development process.

The tremendous change came with large scale language models that were trained over huge code repositories. OpenAI's Codex demonstrated the ability to formulate code across multiple programming languages using a descriptive statement in plain English. This was a giant step in AI-driven coding. It led to the development of tools such as GitHub Copilot, which would suggest snippets of code and automatically complete the code

within widely used programming platforms, radically simplifying programming work.

Subsequently, there were other tools that automatically generated codes using advanced logic. Tabnine, originally known as Codota, went live in 2013 and now offers AI powered code filling services for different programming languages and development environments. The use of AI in coding was further cemented by DeepMind's invention AlphaCode in 2022 that competed in programming competitions and displayed incredible skills, earning an estimated rank within the top 54% of participants, according to [6].

The leaps from elementary code filling functions to Chat-Oriented Programming (CHOP) and Bot-Assisted Task Orchestrating (BATON) demonstrate how AI assisted development has evolved over the years. CHOP allows interactive coding where developers talk with the AI in real time to create and modify code. This approach supports collaboration and simplifies coding procedures. BATON is an evolution from CHOP which encompasses sophisticated AI features such as not only assisting in code writing, but executing a wide range of complex development tasks including setting up basic project features, automating testing, managing dependencies, and even application deployment. This is similar to an oversaw intelligent project manager who does not only assist you with tasks, but also guides you through the process. Implementing BATON in workflows significantly improves efficiency for development teams and relieves them from dealing with ordinary tasks.

## 3 Tools and platforms Supporting CHOP

Artificial intelligence has significantly reshaped software development practices, giving rise to new paradigms such as Chat-Oriented Programming (CHOP). Central to CHOP is the continuous, conversational interaction between programmers and intelligent assistants, enabling developers to tackle coding tasks more efficiently, obtain instant feedback, and mitigate cognitive strain through direct, context-rich exchanges. Prominent tools like ChatGPT, Gemini, DeepSeek, Sourcegraph Cody, and GitHub Copilot each offer unique functionalities tailored to streamline and enrich the programming experience in diverse environments.

ChatGPT, developed by OpenAI, has gained attention for its ability to interpret natural language prompts and generate coherent responses, making it useful in various programming-related tasks. It can produce relevant code examples, help debug common programming errors, and explain technical concepts in accessible terms, which particularly benefits beginners and students new to coding. Padilla et al. (2023) observed that using ChatGPT in higher education contexts significantly supports students by breaking down complex programming topics into understandable explanations, thereby enhancing the learning process and helping students build stronger programming foundations [7]. Nevertheless, studies caution users to carefully review the generated outputs, as ChatGPT may occasionally produce code with minor inaccuracies or suboptimal solutions. Therefore, when incorporating ChatGPT into educational or professional workflows, users should critically assess its suggestions and use them primarily as a complementary resource rather than relying exclusively on its recommendations.

Google's Gemini, a recent development by DeepMind, has quickly become an important tool for programmers seeking practical assistance from AI. Designed to integrate seamlessly with Google's ecosystem, Gemini enhances developers' access to a wide range of programming resources, making it particularly user-friendly. Siam et al. (2024) compared Gemini to other AI assistants and found that it excels in tasks requiring careful reasoning and step-by-step logic, performing slightly better than ChatGPT in complex, structured programming scenarios [8]. Additionally, Gemini is offered in multiple versions to accommodate different project needs, ranging from basic coding assistance suitable for everyday tasks to more powerful

models capable of handling demanding computational problems. This flexibility allows users to choose the appropriate level of support according to the complexity of their work.

DeepSeek approaches the idea of AI-supported programming differently, emphasizing structured reasoning and robust problem-solving capabilities. Instead of simply offering code suggestions or completing simple tasks, DeepSeek is specifically tailored for complex scenarios, such as intricate algorithmic challenges often found in competitive programming. By utilizing reinforcement learning techniques combined with internal self-verification processes, it systematically breaks down difficult problems into manageable steps, refining its solutions iteratively and methodically [9]. This step-by-step reasoning process makes DeepSeek especially valuable in educational contexts, where students must learn not just how to produce code but also how to develop and refine their logical thinking and algorithmic problem-solving skills. In these settings, DeepSeek acts less as a mere code-generating tool and more as a strategic partner that helps learners understand and internalize complex concepts through clearly structured reasoning.

In contrast to DeepSeek's algorithm-centric approach, Sourcegraph Cody is designed to support developers by integrating deeply with their existing project environments. Cody excels in providing context-aware assistance, quickly retrieving precise code snippets, documentation, or relevant information directly from extensive internal project repositories. Instead of general coding advice, Cody's strength lies in rapidly navigating large, complex codebases, reducing the mental effort typically associated with manual code searches or exhaustive documentation reviews. Stray et al. (2023) note that Cody's approach significantly streamlines the onboarding experience for new developers in large teams, enabling them to quickly grasp the project's structure and coding conventions through immediate and accurate access to the relevant code and documentation [10]. By serving as a bridge between individual developer queries and organizational knowledge, Cody contributes directly to improving team collaboration, knowledge sharing, and overall project efficiency, effectively complementing tools such as DeepSeek by handling the contextual and practical demands of professional software development.

GitHub Copilot, a collaborative effort by GitHub and OpenAI, introduced a practical and widely adopted solution for integrating AI directly into the coding process. Integrated seamlessly within popular coding environments, Copilot actively assists developers by offering immediate code suggestions as they type, effectively automating routine coding tasks. According to research conducted by Imai (2022), developers who regularly utilize Copilot report substantial improvements in their productivity and notably reduced mental fatigue, as the tool efficiently handles repetitive coding patterns and boilerplate code generation [11]. Despite these clear benefits, the same study urges caution, emphasizing that code produced by Copilot is not always optimal and may sometimes include insecure or inefficient coding practices. Thus, while Copilot enhances efficiency, developers are advised to thoroughly review and verify AI-generated code before incorporating it into critical parts of their projects.

Beyond these technical benefits, the use of AI-based tools like Copilot and ChatGPT also impacts developers psychologically, influencing their productivity and overall experience in software development. Recent studies, such as those by [12], highlight that developers using AI assistants generally experience lower cognitive strain, allowing them to allocate more mental resources to solving complex, creative problems rather than getting bogged down by tedious, repetitive coding tasks [13]. Similarly, in educational contexts, incorporating tools like ChatGPT into programming curricula has been shown to reduce anxiety and stress among students who find programming concepts challenging. By providing

continuous, approachable guidance, these tools help students develop greater confidence in their coding abilities and achieve better educational outcomes.

Nevertheless, the integration of such tools into both academic and professional settings is not without concerns. The paper [14] highlights significant ethical challenges arising from the misuse of AI coding assistants, particularly regarding plagiarism and academic dishonesty. Their experimental findings indicate that while AI-assisted coding significantly accelerates task completion, it complicates traditional plagiarism detection methods because AI-generated solutions are often unique, making them difficult to identify through conventional means. Consequently, they strongly advocate for establishing clear, explicit institutional guidelines and developing specialized tools to detect AI-assisted plagiarism effectively, thus safeguarding academic integrity and promoting responsible use of AI technologies within educational institutions.

The integration of CHOP tools into software development workflows has introduced noticeable changes in how pair programming is practiced. Traditionally, pair programming involves two developers working closely together, sharing responsibilities and exchanging ideas in real time. When an intelligent assistant becomes part of this dynamic, the structure of collaboration shifts. Rather than substituting the second human partner, the AI tends to take on a support role—offering suggestions, debugging guidance, or even proposing alternative implementations when a challenge arises.

**Table 1.** CHOP tools overview

| **Tool** | **Use Case** | **Strength** | **Caution** |
|---|---|---|---|
| **ChatGPT** | explaining the code, learning or debugging | interpretive and educational | needs careful code review |
| **Gemini** | structured programming | good at logical step-by-step problems | performance varies by model size |
| **DeepSeek** | algorithmic reasoning and problem-solving | structured, iterative reasoning | limited general coding support |
| **Cody** | large codebase projects | contextual assistance | narrow use case: integrated repositories only |
| **GitHub Copilot** | daily development productivity | instant code suggestions | may introduce bugs if unreviewed |

However, the same research also observed certain disruptions. Frequent AI suggestions may inadvertently reduce communication between human collaborators, especially if one developer begins to rely more heavily on the AI assistant. In other instances, the flow of the session may be interrupted by AI inputs that do not align with the pair's ongoing logic or task progression. For AI to support rather than displace interpersonal collaboration, clear integration strategies are required - ones that define how and when suggestions are incorporated and ensure that the exchange of human ideas remains central to the process.

Across the platforms discussed, Table 1: ChatGPT, Gemini, DeepSeek, Cody, and GitHub Copilot, a shared pattern emerges: each tool addresses specific challenges found in different stages or styles of programming. For individual developers, these tools provide immediate assistance, reduce time spent on routine implementation, and serve as a

valuable checkpoint when exploring unfamiliar concepts. In large development teams, context-aware systems like Cody offer faster onboarding and improve navigation through complex codebases, helping developers locate and interpret existing functionality more effectively. In educational settings, tools like ChatGPT and DeepSeek support deeper engagement with fundamental programming principles by enabling students to ask follow-up questions, test hypotheses, and receive structured feedback tailored to their understanding.

Rather than being viewed as interchangeable, these tools contribute best when deployed in alignment with the context in which they are used. While some may excel at language-specific implementation or algorithmic breakdowns, others are more effective in managing large-scale architecture or facilitating code comprehension. Used responsibly, with appropriate oversight and critical thinking, these systems enhance rather than replace the creative and analytical roles of developers and learners alike.

The use of CHOP tools suggests a shift in how knowledge is accessed, how tasks are distributed within development environments, and how learning evolves. Instead of positioning these assistants as autonomous agents, their value becomes clear when seen as collaborators—extensions of the programmer's reasoning process. Their integration, when guided by clear practices and ethical considerations, reinforces the strengths of human judgment while easing the burdens of routine or repetitive work.

## 4 Using CHOP for learning programming concepts

Learning programming has become much easier with Chat-Oriented Programming (CHOP). In the past, programmers had to rely on books to understand important concepts. Later, they searched for answers online using forums like StackOverflow or different documentations. However, with the rise of AI tools such as ChatGPT, Gemini, Claude, GitHub Copilot, Grok and Deepseek, the way we learn programming has changed significantly.

Nowadays, when you attempt to learn about programming, you no longer need to invest much time browsing various websites and documents. You can request AI to provide you with clear and comprehensive responses in just a few seconds. It adapts to your needs and will provide personalized responses if a topic is too difficult or you need a more detailed clarification. This allows for easier learning of the necessary programming paradigms such as data structures, algorithms, and object-oriented programming (OOP). By asking follow-up questions gives you a significant advantage as you can continually seek assistance from an expert across various subjects. The AI tool is capable of serving as a teacher upon your request and it can create diagrams to facilitate your understanding. Consequently, programming education has become increasingly interactive, tailored, and effective like never before.

CHOP can be easily used to learn the fundamentals of programming, including concepts like Object-Oriented Programming (OOP), data structures, design patterns, memory management, and different algorithms.

A significant benefit of CHOP in learning OOP is its capacity to quickly create examples in various programming languages. If a student is studying Java, Python, or C++, they can easily request an AI chatbot to explain the same OOP concept (such as inheritance or polymorphism) in various languages, allowing them to understand how syntax, structure, and memory management vary among languages.

In languages such as Java and Kotlin, memory management is automatic due to garbage collection, making object handling easier. On the other hand, C++ requires developers to manually allocate and free memory, requiring a more comprehensive grasp of memory management. CHOP assists students in recognizing these distinctions explicitly, enhancing their understanding of high-level and low-level memory management in object-oriented programming.

This flexible learning approach helps students

grasp not just the theory behind OOP but also how it's used in industries that interest them. By incorporating comparisons across multiple programming languages and real-world scenarios, CHOP strengthens both conceptual understanding and practical skills, making OOP more accessible and engaging.

CHOP also has a significant impact on learning data structures and algorithms. It can provide real-time visualizations, such as diagrams, making the overall experience more appealing and easier to understand by the user. Instead of relying on pure theory, learners can see how different data structures, such as linked lists, look and observe how sorting algorithms operate step by step, making the complex concepts look more intuitive. The same benefits also apply here because we can see how different data structures look in more programming languages. We can see how divergent they are, such as with built-in data structures in some languages (e.g. Java) and none in others (e.g. C) which require you to implement them by yourself. As students explore these differences, they learn to navigate data structures in a manner that is informed by performance tradeoffs and best practices.

As for algorithms, AI is tremendously useful here, since it helps determine what algorithms to use. It helps us understand the differences between algorithms that perform the same task (such as sorting algorithms), analyze their complexity, and when we should use algorithms in accordance with the necessity.

Security is another important area where CHOP helps learners stay updated with the latest technologies and algorithms. For example, some encryption methods used today will no longer be secure when quantum computing becomes more advanced. This means new solutions will be needed to keep data safe. CHOP allows learners to keep up with these changes, explore new cryptographic methods, and choose the best and most secure options for the future.

AI models have significantly transformed how students learn programming. Unlike traditional education methods, which usually follow a more standard and rigid structure, AI technologies can easily adapt to satisfy the user requests, making the overall experience feel more flexible and learner centered. These tools can provide personalized guidance to students, adapting to their learning pace and addressing their specific challenges. AI assistants are highly beneficial for learning because they can help debug code, provide real time feedback, customize exercises and can make the experience interactive. They also have a positive impact on student mental health, because they can reduce frustration and boost confidence and engagement in programming tasks. Since AI is available 24/7, learners can seek help any time, making the programming education more accessible and supportive.

Unlike traditional learning environments where students had to wait for scheduled classes or seek help from teachers, AI assistants are available at any time, without limitations. Whether it's late at night or during a study session on a weekend, the learner no longer must wait for anyone to assist him in clearing their doubts. AI-driven tools allow instant access to explanations, debugging assistance, and coding guidance whenever needed, making programming education more independent and self-paced.

CHOP introduces a more engaging way to learn programming by integrating gamification elements into the learning process. You can play different games with AI to better understand certain concepts. You can engage in Q&A sessions, ask it to provide code examples to find bugs, request algorithm optimizations, or test newly learned concepts through progressive challenges. This approach is beneficial because practicing syntax and logic in this way helps learners build confidence and develop a strong, in-depth understanding of the concepts.

CHOP helps students learn on their own and works like a teaching assistant, giving explanations and clearing up doubts about programming. AI tools like ChatGPT are great at explaining basic programming concepts, making them useful for learning. However, research shows that while AI can

explain simple topics well, it has trouble helping with more complex problems and adapting to each student's learning needs [15]. This means that CHOP is a great tool for learning the basics, but human teachers are still needed for solving harder problems and debugging code.

According to [15], while CHOP is a powerful tool for learning programming, it still has limitations in adaptability and personalization. AI can provide accurate explanations, but it cannot fully understand how students think or offer customized responses for unique problems.

CHOP is an extremely effective resource for learning programming, enhancing the process to be more interactive, accessible, and enjoyable. It helps students by providing immediate feedback, clarifications and custom exercises to help them gather essential concepts. However, there is still room for improvement especially when it comes to adapting to individual learning styles and handling more complex coding challenges. As AI continues to evolve, enhancing CHOP's ability to provide more tailored support and deeper problem-solving guidance will make it an even more powerful learning tool.

## 5 Advantages and Challenges of CHOP

Working with CHOP changes the way you approach code, not dramatically, but gradually, almost imperceptibly at first. You stop switching between tabs, you ask fewer questions on forums, and you reach for documentation less often. Instead, you start writing directly to a chatbot. Sometimes out of curiosity, other times because it's simply faster. And most of the time, it works well enough to keep going.

The clearest benefit is how much time you save on small, repetitive tasks. If you forget the syntax for a function, or can't recall how a specific library is imported, you no longer need to dig through documentation. You just ask. It keeps the momentum going, especially when you're in the middle of building something. That fluidity makes a big difference. It's especially useful when you're learning. Concepts that used to feel abstract, like pointers, inheritance, or recursion, become easier to grasp when explained in plain language, backed up by examples in different programming languages. You can ask the same question in three different ways, and you'll get an answer each time. That kind of persistence used to be a luxury. Now it's the default. In teams, CHOP plays a different role. It helps new developers understand unfamiliar codebases faster. Tools like Copilot or Cody can explain how a function works, or where a particular class is used. You still need to think, but you don't need to feel lost. That's no small thing in a large codebase.

Still, there are limits. The biggest one is that not everything generated by AI is correct. In fact, the more complex your question, the more likely it is that something will be off, an assumption, a missing detail, a function that looks right but doesn't actually do what it claims. The answers are confident, even when they're wrong. And if you trust them without checking, you're likely to run into problems you can't trace easily. There's also the risk of leaning too hard on it. When you're used to getting help instantly, you start skipping the part where you'd usually try to figure it out yourself. That makes things faster in the short term, but slower in the long run, especially when the AI is unavailable, or wrong, or simply doesn't understand what you're really asking.

In education, it gets more complicated. It's hard to tell how much of a student's code was actually written by them. The tools don't plagiarize in the traditional sense, but they do blur the line between help and substitution. That's not necessarily a flaw, but it raises real questions about learning, authorship, and what we actually mean when we say "understanding code." CHOP isn't good or bad on its own. It depends entirely on how it's used. When it's part of your process, not the whole process, it can be a powerful tool. But if it replaces thinking, or turns into a shortcut you rely on too heavily, it does more harm than good.

## 6 Use Case: The Role of AI in Software Product Development - Deepfake generator

Technological advancement has changed the usual paradigm of all aspects of our lives, surpassing the status of a technical tool and establishing itself as an increasingly active partner in the educational process. In particular, conversational models such as ChatGPT, Claude, Copilot, and Cody attract

attention not only through their algorithmic performance but through their ability to directly support learning processes through interaction, conceptual clarification, and assistance in solving complex problems. These models can become useful tools between technological complexity and the human need for contextual and practical understanding.

This chapter will analyze such an interaction, conducted within academic research dedicated to digital security and the deepfake phenomenon. At the center of this endeavor was the need to understand and adapt an existing application, available on GitHub, to support a scientific presentation on the impact of emerging technologies on identity and safety in the digital space. The dialogue with ChatGPT was essential in the process of understanding the source code, identifying the internal mechanisms of the application, and configuring new functionalities that would reflect the research objectives.

The analyzed application, DeepFaceLab, represents one of the most used open-source platforms for generating deepfake content. Its structure is organized around three essential processes: extracting faces from video materials, training a neural network model to learn facial features, and finally, conversion, overlaying a learned face onto target video content. Although efficient in demonstrating the visual substitution process, the original architecture of the application does not allow the generation of completely new content, but only the modification of an existing one. Starting from this functional limitation, the objective was to reconfigure the application so that it could generate synthetic, realistic, animated faces, without needing a source video clip in which to replace a pre-existing face. This adaptation involved not only modifying code segments, but also integrating another machine learning model with a fundamentally different architecture: GAN-type generative networks (Generative Adversarial Networks), capable of creating original content based on a preprocessed dataset.

In this process of functional reconversion, interaction with ChatGPT offered essential support in several stages. The first consisted of understanding the structure of the application, identifying relevant files, and clarifying the role of each script. Files such as *scripts/extract.py, scripts/train.py, and scripts/convert.py* were analyzed in detail to determine how they could be modified so that the application would no longer function exclusively as a replacement system, but would allow the generation of completely new images. The explanations were precise, and the code recommendations were accompanied by technical justifications, which facilitated understanding the internal logic of the application. Subsequently, the integration of the StyleGAN3 model, developed by NVIDIA, a high-performance generative model optimized for facial analysis, was proposed. ChatGPT provided details about how to configure this model, about its computational requirements, and about the exact steps to convert a set of images extracted from videos into a format compatible with training the GAN. The interventions also covered aspects of image optimization, resizing to 1024x1024 pixels, and formatting the dataset according to the specific requirements of the model.

An aspect that contributed to understanding and restructuring the application was the support in understanding the conceptual differences between autoencoder networks, used standardly in DeepFaceLab, and generative networks. While the former are built to map one image to another through reconstructional learning, the latter are capable of creating completely new content without faithfully reproducing a previous example. This difference was explained in a concise and practical way, through concrete code examples and by describing the dual architecture of GANs (generator and discriminator).

After integrating the StyleGAN3 model and training it on a relevant dataset, the next stage consisted of animating the generated faces to simulate natural expressions and movements. For this purpose, the First Order Motion Model was also introduced into the project,

which allows the transfer of facial movement from a source video to a static image, thus generating animated, realistic, and convincing video content. Again, the technical stages were explained specifically: downloading the model, configuring input files, and running generation scripts.

The entire journey was structured not only according to technical needs, but also in relation to the didactic purpose of the project. Explanations were offered based on the individual learning rhythm, and recommendations were adapted to the level of complexity necessary at each stage. In this way, the conversational model did not act as a solution generator, but as a technical consultation partner, which supported understanding the algorithmic logic, the relationships between components, and the ethical implications of the technology used.

This learning experience using AI models highlights the real potential of conversational models to support complex technical processes, without replacing critical thinking or cognitive effort, but positioning themselves at the intersection between tool and partner. The interaction with ChatGPT facilitated not only the understanding of advanced programming and artificial intelligence concepts, but also the structuring of a coherent technological approach, integrated into an academic framework. Without claiming an exclusive role in the formation process, this type of assistance opens new perspectives on how learning can be approached in the digital age, more flexible, more contextualized, and closer to the real needs of applied research and adapted to the level and needs of users.

## 7 Conclusions

Chat-Oriented Programming (CHOP) represents a natural evolution of the way developers interact with technology, marking the transition from traditional programming approaches to conversational models, supported by artificial intelligence. By using tools such as ChatGPT, Copilot, Gemini or DeepSeek, the software development process becomes more accessible, efficient and adaptable, both in educational contexts and in industry. CHOP significantly contributes to reducing technical barriers, providing real-time contextual support and facilitating interactive learning of complex concepts such as object-oriented programming, data structures or fundamental algorithms.

However, the integration of this paradigm also brings important challenges, especially in terms of the accuracy of the generated code, the risks of over-dependence and the ethical dilemmas related to originality and academic evaluation. For CHOP to reach its full potential, it is essential to formulate clear usage guidelines, along with a continuous development of AI models to make them more reliable, more customizable and more transparent. In conclusion, CHOP should not be seen as a substitute for human thinking, but as a conversational partner that can amplify creativity, efficiency, and learning in the field of programming.

## References

[1] A Kukic, (2024, July 27) ChatOriented Programming (CHOP) in action, Available online: https://sourcegraph.com/blog/chat-oriented-programming-in-action, Accessed: 09.02.2025

[2] F. Khennouche, Y.Elmir, N.Djebari, Y.Himeur, A.Amira, "Revolutionizing Customer Interactions: Insights and Challenges in Deploying ChatGPT and Generative Chatbots for FAQs", 2023, pp.2, DOI: https://arxiv.org/abs/2311.09976

[3] L. Gugerty, "Newell and Simon's Logic Theorist: Historical Background and Impact on Cognitive Modeling", Clemson University, 2017, pp.2-3, DOI: 10.1177/154193120605000904, accessed: 15.03.2025

[4] C. Bassett, "The computational therapeutic: exploring Weizenbaum's ELIZA as a history of the present", 2018, pp.805-806, DOI: https://doi.org/10.1007/s00146-018-0825-9, accessed: 15.03.2025

[5] Z. Krdzic, "AI Winter: The Reality Behind Artificial Intelligence History", 2024,

available online: https://aibc.world/learn-crypto-hub/ai-winter-history/, accessed: 15.03.2025.

[6] The AlphaCode team, "Competitive programming with AlphaCode", 2022, available online: https://deepmind.google/discover/blog/competitive-programming-with-alphacode/, accessed: 16.03.2025

[7] J. R. C. Padilla, M. D. L. Montefalcon, and A. A. Hernandez, "Language AI in Programming: A Case Study of ChatGPT in Higher Education Using Natural Language Processing," in *Proceedings of the International Conference on Signal Processing and Communications (ICSPC)*, 2023, accessed: 12.03.2025.

[8] Md K. Siam, H. Gu, and J. Q. Cheng, "Programming with AI: Evaluating ChatGPT, Gemini, AlphaCode, and GitHub Copilot for Programmers," in *Proceedings of the 7th International Conference on Computational Applications (ICCA)*, 2024, accessed: 12.03.2025.

[9] DeepSeek AI, "DeepSeek-R1: First-Generation Reasoning Model," GitHub Repository, 2024. Available online: https://github.com/deepseek-ai/DeepSeek-R1, accessed: 14.03.2025

[10] V. Stray, N. B. Moe, N. Ganeshan, and S. Kobbenes, "Generative AI and Developer Workflows: How GitHub Copilot and ChatGPT Influence Solo and Pair Programming," in *Proceedings of the 57th Hawaii International Conference on System Sciences (HICSS)*, 2024, accessed: 14.03.2025.

[11] S. Imai, "Is GitHub Copilot a Substitute for Human Pair-Programming? An Empirical Study," in *Proceedings of the 44th International Conference on Software Engineering (ICSE C*[20] SAP LeanIX, "History of AI".

[12] Grimskär, H., & Johansson, N. (2024). AI Code Generation: Trust & Risk Awareness Across Educational Levels.

[13] M. Valovy and A. Buchalcevova, "The Psychological Effects of AI-Assisted Programming on Students and Professionals," Preprint, Prague University of Economics and Business, November 2023, accessed: 12.03.2025.

[13] Karnalim, O., Sujadi, S. F., & Nathasya, R. A. (2024, March). Automated Code Readability Feedback on Student Awareness. In *International Conference on Smart Technologies & Education* (pp. 56-66). Cham: Springer Nature Switzerland.

[15] Z. Ahmed, S. S. Shanto, A. I. Jony, (2024) Potentiality of generative AI tools in higher education: Evaluating ChatGPT's viability as a teaching assistant for introductory programming courses, STEM Education, Volume 4, Issue 3, 165–182. Available online: https://doi.org/10.3934/steme.2024011, Accessed: 16.03.2025

**Alin ZAMFIROIU** has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2009. In 2011 he has graduated the Economic Informatics Master program organized by the Academy of Economic Studies of Bucharest and in 2014 he finished his PhD research in Economic Informatics at the Academy of Economic Studies. He works as a Senior Researcher at "National Institute for Research & Development in Informatics, Bucharest". He has published as author and co-author of journal articles and scientific presentations at conferences.

**Costinela-Beatrice PĂȘTINICĂ** is currently pursuing a bachelor's degree in Economic Informatics at the Faculty of Economic Cybernetics, Statistics, and Informatics (Class of 2025), at the Bucharest University of Economic Studies. Her research focuses on the crossroads of artificial intelligence, digital transformation, and geopolitics. She has presented interdisciplinary papers at academic conferences, with recent work centered on ethical deepfake detection, combining machine learning with critical insights on power, identity, and digital trust.

**Claudia CIUTACU** is a graduate of the Faculty of Cybernetics, Statistics, and Economic Informatics (class of 2023) and is currently pursuing a master's degree in IT&C Security at the Bucharest University of Economic Studies. Passionate about mobile technology and cybersecurity, she works as an Android developer where she combines her technical expertise with creative problem-solving to build innovative applications.

**Cătălin Ionuț CRĂCIUN** has graduated the Faculty of Cybernetics, Statistics, and Economic Informatics in 2023 and is currently pursuing a master's degree in IT&C Security at the Bucharest University of Economic Studies. With a strong focus on mobile innovation and secure app development, he works as a mobile developer, contributing to the creation of reliable and intuitive mobile solutions that meet modern user needs.

**Theodor LUCA** has graduated from the Faculty of Cybernetics, Statistics, and Economic Informatics at the Bucharest University of Economic Studies in 2024. Interested in the digital transformation of processes, he focuses on integrating modern technologies into complex workflows. His research also extends to transdisciplinary areas such as e-Learning and cost optimization of digital infrastructures.