

AI ENGINEERING PRODUCTIVITY COOKBOOK

0111 110000
1010 00010110
0101 11110100
01001 0010011
1000011011110
11000110001
00010011101
100111100101
01001110111
0110001011
1100001001
001010111
1001100011
010001011



intellias

Contents

Foreword	5
Chapter 1: From AI theory to practice.....	7
From early pioneers to modern breakthroughs	8
Paradigm shift with generative AI.....	9
Which AI assistant to choose?	11
Key takeaways	17
Chapter 2: Coding with GitHub Copilot Business	18
Project scope and tool stack.....	19
Code completion and suggestions	21
Refactoring and optimization.....	27
Debugging and bug fixing.....	36
Key takeaways	40
Chapter 3: Advanced AI capabilities	41
AI in software development	42
AI for DevOps.....	55
AI-driven no-code and low-code platforms.....	61
Key takeaways	64
Chapter 4: Collaborative development with AI	65
Pair programming with AI.....	66
Code review and quality assurance.....	74
Knowledge sharing and documentation	78
Key takeaways	81

Chapter 5: Secure use of AI82

 Privacy and security risks83

 AI risks mitigation.....84

 Key takeaways87

Chapter 6: Building AI products from business and user perspectives88

 Two main ways of using AI.....90

 Trust in AI by design.....92

 Bias in AI models98

 Key takeaways101

Chapter 7: Intellias AI portfolio102

 Compl-AI, a solution for ensuring regulatory compliance of business processes103

 IntelliAssistant, a platform for enhancing corporate productivity and employee experience ... 104

 AI Copilot, a digital assistant for software engineering106

 Key takeaways107

Chapter 8: The future of AI in software development.....108

 Emerging trends.....109

 Key takeaways113

Afterword114

About the authors

We are the engineering team from Intellias Technology Office, who participated in the AI Copilot program and contributed to the design and development of three proprietary AI-based products: Compl-AI, IntelliAssistant, and IntelliCopilot. Having contributed to these AI initiatives in a variety of roles, such as software developers, DevOps engineers, test automation engineers, product managers, and project managers. We are excited to share our insights on the opportunities, risks, and limitations associated with AI technologies in software engineering.

We hope you enjoy reading this book as much as we enjoyed compiling our findings and exploring the potential of AI.



Denys Babych

Engineering Manager



Illia Diachenko

Head of Web Center of Excellence



Kateryna Ovechenko

Head of Customer Experience Center of Excellence



Mykyta Khaustov

Principal JS Engineer



Nikita Postolakii

Head of Excellence



Oleksandr Poliukh

Lead DevOps (AWS) Engineer



Olha Vdovych

Head of Documentation Unit



Dmytro Vedetskyi

Head of Cloud and DevOps



Volodymyr Hordiienko

Principal DevOps Engineer (AWS)



Oleksii Bratenkov

Senior Software Engineer (Java)



Roman Kyryliuk

Senior .NET Engineer



Serhii Seletskyi

Senior Solution Architect



Yony Muneton

Senior CRM and Power Apps Engineer



Vasyl Huzinskyi

Manager of Design Unit



Khrystyna Matiitsiv

Visual Designer

Foreword

Welcome to the world of AI-assisted coding! In this “AI engineering productivity cookbook”, we have collected practical insights and findings on the use of AI in software development. Over the past four months, Intellias has leveraged AI-assisted services to energize its software development processes, measure productivity gains, and make efficiency improvements across various verticals.

Our experience shared in this book comes from the successful implementation and ongoing rollout of the AI Copilot program. This program, which is integral to Intellias’ AI-assisted engineering efforts, encompasses both internal research and development projects as well as client engagements across six diverse industry verticals: retail, high tech, financial services, insurance, travel, healthcare, and internal product development.

The objectives of this program include:

- ▶ Enhancing performance measurement and evaluation
- ▶ Building internal capabilities for Copilot use and knowledge dissemination
- ▶ Educating engineers on Copilot best practices
- ▶ Collaborating with clients to refine Copilot applications
- ▶ Evaluating security measures
- ▶ Comparing various Copilot offerings on the market
- ▶ Providing recommendations and options for broader company rollout

As expected the AI Copilot program has yielded remarkable results. It has significantly **increased productivity**, with 65% of participating engineers reporting over a **25% increase**. Other notable results include:

- ▶ **53% increase in focus on task completion**
- ▶ **81% reduction in frustration related to routine tasks**

- ▶ 79% acceleration in information retrieval
- ▶ 51% enhancement in code review efficiency

These improvements underscore Intellias' commitment to harnessing AI technology to cultivate a more efficient and effective engineering environment. Through the AI Copilot program, Intellias has enhanced clients' operational efficiency, reducing time-to-production and significantly boosting code deployment speeds, leading to a more agile and responsive development cycle.

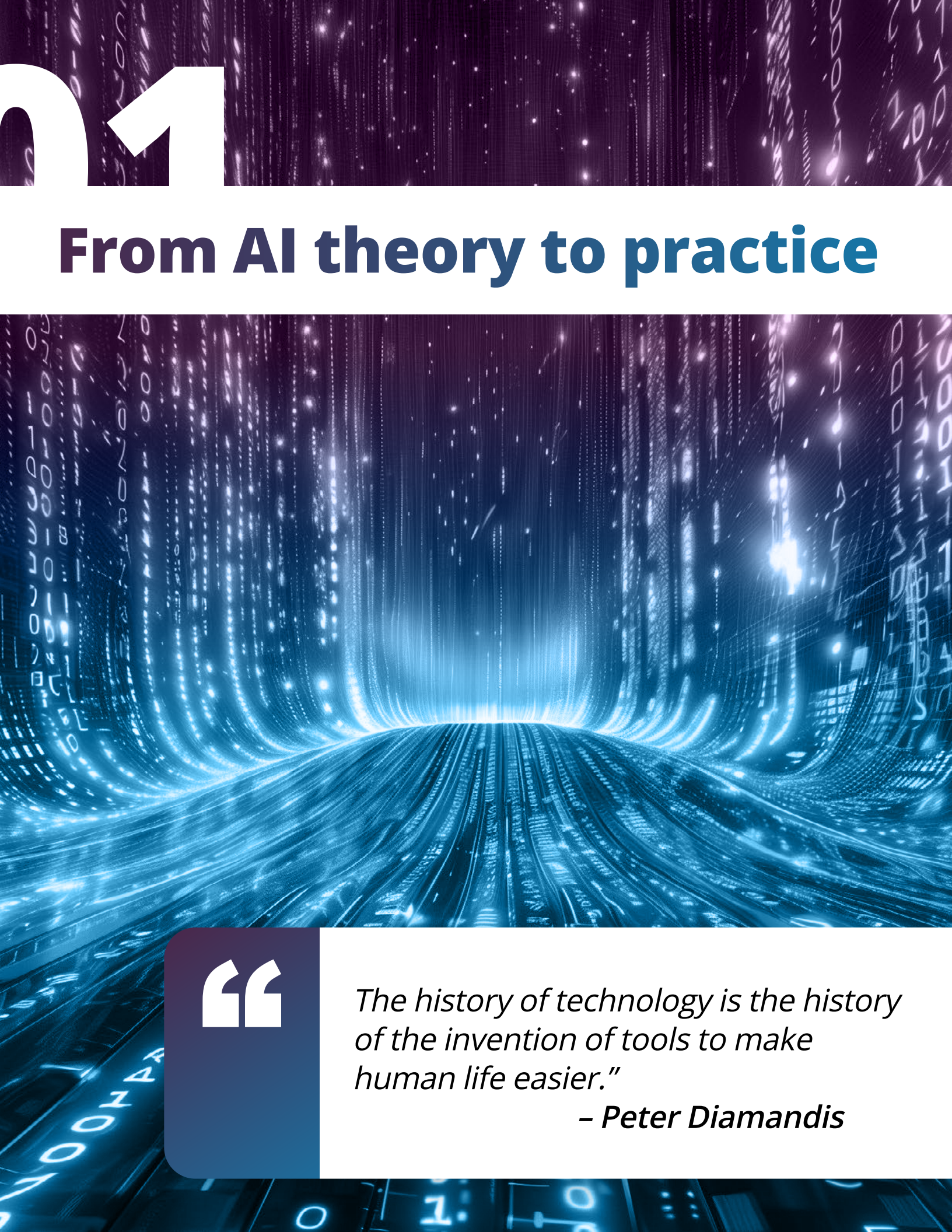
Whether you're an experienced software developer or just getting started, learning practical applications of AI will help you boost productivity, making coding easier and more fun. The book will be of interest to **Software Developers, Test Automation Engineers, and DevOps Engineers**, especially those working with **Java, JavaScript, and .NET stacks**. In our research, we explore how the integration of top-rated AI tools such as **GitHub Copilot Business, Tabnine, and Amazon CodeWhisperer** as well as **our propitiatory coding assistant** can optimize the development workflow in terms of comfort level and opportunities in code documentation, debugging, predicting potential bugs, and optimizing code efficiency.

For organizations considering introducing AI tools, this book will showcase the capabilities and benefits of low-code and no-code systems, accelerating software development across various roles and methodologies.

With the promise of AI-driven innovation and acceleration comes the responsibility to address ethical, security, and compliance considerations. We elaborated our own approach to ensuring that AI serves as a catalyst for progress by implementing **ethical coding practices and security measures**, as well as **bringing to market our own product that checks the compliance of business processes and systems**.

As we prepare for the future of software development, it's clear that embracing emerging trends in AI will be key to staying ahead of the curve. From enhanced code generation and auto-completion to AI-powered testing and debugging, the possibilities are limitless. By embracing these trends, we position ourselves to shape the future of technology and drive innovation in an ever-evolving digital landscape.

So, whether you're starting on an AI tool selection journey or looking to take your productivity to the next level, this book will be your companion in navigating the world of AI-assisted development. Get ready to code faster, smarter, and better with the power of the right AI tool at your fingertips!



01

From AI theory to practice

“

The history of technology is the history of the invention of tools to make human life easier.”

– Peter Diamandis

From early pioneers to modern breakthroughs

The idea of using computers to generate program code is not entirely new. Its roots trace back to the early days of computing. In the 1960s, pioneers in the field of artificial intelligence and computer science started exploring ways to automate software development tasks, laying the foundation for what would eventually evolve into generative AI.

One of the earliest attempts to automate programming can be attributed to **Grace Hopper**, a pioneering computer scientist known for her work on the UNIVAC I and COBOL programming languages. Hopper's **creation of the first compiler in the early 1950s** revolutionized software development by **translating high-level programming languages into machine code**, which automated the process of code translation.

In the **late 1960s and early 1970s**, researchers such as **John McCarthy**, known as the father of AI, and **Alan Kay**, a pioneer in object-oriented programming, **started investigating the potential of AI techniques**, such as expert systems and rule-based reasoning, to assist in software development tasks. McCarthy's work laid the foundation for symbolic AI, which aimed to mimic human-like reasoning and problem-solving abilities using symbolic representations and logical inference.

Throughout the **1980s and 1990s**, advancements in machine learning and natural language processing further fueled interest in using AI for code generation. Researchers explored the **application of machine learning algorithms to program synthesis** and experimented with using NLP techniques to parse and generate code from natural language specifications.

In the early **2000s**, researchers at institutions like **MIT and Stanford** began the development of automated programming systems aimed at **the automatic generation of code from high-level specifications or user demonstrations**.

Fast forward to the **2010s**, and the advent of deep learning and neural networks revolutionized the field of AI, enabling significant breakthroughs in generative models. Key developments, such as the **introduction of recurrent neural networks, long short-term memory networks, and transformer architectures**, paved the way for powerful generative AI models capable of understanding and generating complex sequences of data, including code.

In recent years, models like **OpenAI's GPT** series have demonstrated remarkable capabilities in natural language understanding and generation, including code generation tasks. These models leverage **large-scale**

pre-training on vast amounts of text data to learn rich representations of language and context, enabling them to generate coherent and contextually relevant code snippets given appropriate prompts.

Overall, while the idea of using computers to generate code has been explored for decades, recent advancements in AI and machine learning have brought us closer than ever to realizing the vision of truly intelligent code generation systems.

Paradigm shift with generative AI

The adoption of generative AI for code generation holds profound implications for the IT industry.

- ▶ **Cost efficiency in software development.** Integrating generative AI into development workflows can lead to significant cost savings in software development. By automating repetitive coding tasks, reducing manual effort, and accelerating development cycles, generative AI enables companies to optimize resource allocation, minimize overhead costs, and deliver projects more efficiently. This cost efficiency can make software development more accessible to a wider range of organizations, particularly startups and small businesses, changing the environment and making innovation more achievable for everyone in the industry.
- ▶ **Future development of programming languages.** Generative AI could change how we write, understand, and use programming languages. As AI-generated code becomes more common, programming languages might adapt to suit AI's needs, like natural language processing and machine learning. This could bring new ways of coding and tools designed to make the most of AI in software development, sparking innovation and expanding what we can do with programming.



- ▶ **Impact on software development methodologies.** IT leaders must also recognize the transformative impact of generative AI on software development practices and adjust their approaches accordingly. We need to understand how AI technologies will reshape the software development lifecycle and popular methodologies like Agile and Scrum. Future-efficient approaches to software development will involve tight integration with code generation and analysis tools, enabling developers to leverage AI-driven insights and automation throughout the development process.
- ▶ **Impact on the workforce and job market.** The widespread adoption of generative AI for code generation may lead to significant changes in the job market for programmers. While AI can automate coding tasks and boost productivity, there's a risk that many programmers may face job displacement. To mitigate this risk and secure jobs, developers need to proactively develop new skills and competencies that enable them to collaborate effectively with generative AI technologies. This entails mastering AI-driven tools and methodologies such as AI/ML, data science, and machine learning, understanding how to leverage AI for code generation and analysis, and adapting to new development paradigms where humans and AI work in tandem to achieve optimal results. By embracing these skills, developers can remain indispensable contributors to the software development process, ensuring their continued relevance in the evolving landscape of technology.



In this new paradigm, IT leaders play a crucial role in fostering a culture of innovation and continuous learning within their teams. They must provide resources and support for developers to acquire the necessary skills and adapt to the changing demands of the industry. By utilizing generative AI and taking care to educate talents to use it effectively, IT leaders can enhance efficiency, productivity, and competitiveness within their organizations.

Overall, the adoption of generative AI for code generation has the potential to transform the IT industry, driving cost efficiency, shaping the future of programming languages, and influencing the workforce dynamics. By embracing AI-driven technologies, companies can stay ahead of the curve, unlock new opportunities, and drive innovation in the ever-evolving IT landscape.

In this book, we explore how generative AI can enhance software development across various roles such as software developers, devops engineers, and quality control engineers (chapters 2 and 3). We delve into topics related to the collaboration between engineers and generative AI, analyze its impact on ethical considerations, security, and user experience (chapters 5 and 6), and contemplate potential future AI trends in the IT industry (chapter 8).

Which AI assistant to choose?

The AI tool market is booming, with dozens of companies vying for prominence with their AI offerings. Given the abundance of options, let's evaluate and select the most fitting tool for software development, whether for organizational or individual needs.

In this book, we compare the power of coding assistants, delving into their strengths and weaknesses to serve as a compass for those embarking on the adoption journey. We focus on some of the most popular choices in the developer community: GitHub Copilot Business, Tabnine, and Amazon CodeWhisperer. Our goal is to uncover how these tools help developers overcome coding challenges, from navigating complex code complications to streamlining unit tests and refactoring.

Assessment metrics

We have selected the three most popular stacks – **Java**, **.NET**, and **JavaScript** – to assess them from the point of user experience output quality. Our developers used [GitHub Copilot Business](#), [Tabnine](#), and [Amazon CodeWhisperer](#) for a week, giving feedback on a predefined list of metrics, which you can see below.

Please note that the metrics that remain unassessed are of very poor quality.

Disclaimer

This book presents our subjective insights and practical findings collected during our research into AI tools during the first half of 2024. As many innovations and shifts in tooling and techniques occur every month, this snapshot may have evolved by the time you read this book.

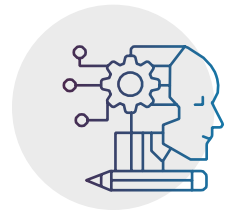
Metric	GitHub Copilot Business	Tabnine	Amazon CodeWhisperer
Bug fixing measures the AI tool's efficiency in identifying bugs, errors, and potential vulnerabilities in the code and suggesting or implementing appropriate fixes. It includes the AI's ability to understand the context and dependencies within the code, ensuring that the proposed fixes do not introduce new issues.	★ ★ ★	★ ★	★ ★
Chat evaluates the AI tool's communication capabilities to understand complex queries, provide relevant and context-aware responses, and maintain a coherent and logical dialogue. This involves technical explanations and the ability to engage in problem-solving discussions, offering alternative solutions or approaches whenever requested.	★ ★ ★ ★ ★	★ ★ ★	★ ★
Code completion focuses on the AI's ability to predict and suggest the completion of blocks of code as the developer types, enhancing coding speed and reducing syntactical errors. This includes not just the completion of simple statements but also more complex constructs, considering the local code context, variable types, function signatures, and even coding style.	★ ★ ★ ★ ★	★ ★ ★ ★	★ ★
Code explanation assesses the AI's ability to provide clear, concise, and accurate descriptions for blocks of code, including its functionality, logic, and known issues. This is crucial for improving code readability and maintainability and simplifying the onboarding process for new team members.	★ ★ ★ ★ ★	★ ★ ★ ★	★
Code generation (prompts in chat) evaluates how effectively the AI tool can produce functional code blocks or entire functions, how it adheres to best practices and generates code that is not only syntactically correct but also logically sound in terms of the project's code base and architecture.	★ ★ ★ ★ ★	★ ★ ★ ★	★ ★
Personal satisfaction reflects the user experience and overall satisfaction with the AI tool. This includes such aspects as ease of use, integration into your workflow, intuitiveness, the relevance of provided solutions, and how well the tool meets your project requirements.	★ ★ ★ ★ ★	★ ★ ★ ★	★
Refactoring evaluates the AI tool's ability to suggest and execute improvements in the code structure without altering its external behavior. This includes simplifying complex code, enhancing readability, reducing redundancy, and adhering to the SOLID principles, among other best practices, to maintain and improve the code base's health in the long run.	★ ★ ★ ★	★ ★	★
Testing assesses the AI tool's efficiency in generating test cases that cover a wide range of inputs and edge cases for the code, ensuring that the tests are meaningful, relevant, and comprehensive. It also examines the AI's ability to understand and implement testing frameworks and methodologies appropriate to each stack.	★ ★ ★ ★	★ ★	★
TOTAL	★ ★ ★ ★ ★	★ ★ ★	★ ★

Key features of the winner

As evident from the feedback, developers preferred GitHub Copilot Business over Tabnine and Amazon CodeWhisperer. Therefore, we will focus primarily on this tool. While it's important to note that each tool possesses unique features that may excel in certain scenarios, GitHub Copilot Business emerges as the industry standard, consistently excelling across various aspects.

Dror Weiss, CEO of Tabnine, highlights that in contrast to Copilot's monolithic architecture, which can be summarized as a "one model to rule them all" approach, Tabnine offers a more flexible and decentralized solution. Copilot's design is inherently centralized, necessitating that only Microsoft has the capability to both train and host the model due to the significant computing resources required for these tasks. Tabnine, however, provides machine learning models of various sizes, ranging from 400 million to 3 billion parameters. This diversity allows Tabnine to deploy the optimal model for any given code prediction task, presenting several key advantages:

- ▶ **Types of code suggestions.** Unlike Copilot, which is not optimized to suggest code in the middle of a line, Tabnine's smaller, more efficient AI models offer dynamic, real-time feedback as the user types. This capability ensures that the AI assistance remains in sync with the user's workflow.
- ▶ **Customizability and training.** Copilot uses a universal AI model, offering the same general assistance derived from an aggregate of GitHub data to all users, regardless of their project's specifics. In contrast, Tabnine's modular approach facilitates the creation of personalized AI models. This democratization of AI model creation empowers teams to train models tailored to their unique needs, significantly enhancing the value delivered to their organization.
- ▶ **Security and privacy considerations.** Tabnine users have the flexibility to decide where the model runs: in the cloud, on a local developer machine, or on a self-hosted server through Tabnine Enterprise. This flexibility is enabled by Tabnine's efficient AI models, designed to operate effectively on moderately equipped hardware.



Small UI quirks

At first glance, all the applications appear to do the same things, as their functionalities are quite common. However, each application has some unique features that set it apart. For example, GitHub Copilot Chat highlights variables and function names in different colors, which makes them more readable.

Here is how it appears in **GitHub Copilot Business**:

GitHub Copilot used @workspace /explain

Used 1 reference >

The provided TypeScript code defines a function named `before`. This function takes two parameters: `n` and `func`. The purpose of this function is to create a new function that will invoke `func` only a certain number of times (`n` times). After `func` has been invoked `n` times, subsequent calls to the created function will return the result of the last invocation of `func`.

Let's break down the function:

1. `let result;`: This line declares a variable `result` that will be used to store the result of the last invocation of `func`.
2. `if (typeof func !== 'function') {...}`: This if statement checks if `func` is a function. If `func` is not a function, it throws a `TypeError`.
3. `return function (...args) {...}`: This line returns a new function that takes any number of arguments (`...args`). This is the function that will be invoked when the returned function is called.
4. Inside the returned function:
 - `if (--n > 0) {...}`: This if statement checks if `n` is greater than 0. If it is, it decrements `n` by 1 and then invokes `func` with the provided arguments and the same this context. The result of `func` is stored in `result`.
 - `if (n <= 1) {...}`: This if statement checks if `n` is less than or equal to 1. If it is, it sets `func` to `undefined`, effectively preventing `func` from being invoked again.
 - `return result;`: This line returns the result of the last invocation of `func`.

The function `before` is then exported as a default export, which means it can be imported with any name in another module. This function can be useful in scenarios where you want to limit the number of times a function can be invoked.

❖ Can you provide an example of how to use the `before` function from the `lodash` library?

GitHub Copilot used @workspace /explain

Used 1 reference >

The provided TypeScript function, `transformStringFieldUpdateInput`, is an asynchronous function that takes two arguments: `input` and `transform`. The `input` argument can be of type `undefined`, `string`, or an object with an optional `set` property of type `string`. The `transform` argument is a function that takes a string as input and returns a Promise that resolves to a string.

The purpose of this function is to apply the `transform` function to the `input` if it is a string or if it is an object with a `set` property of type `string`. The transformed value is then returned as a Promise.

The function first checks if `input` is an object and if it has a `set` property of type `string`. If both conditions are true, it applies the `transform` function to `input.set` and returns a Promise that resolves to an object with a `set` property containing the transformed value.

Next, the function checks if `input` is an object. If it is, but it doesn't have a `set` property of type `string`, the function simply returns the `input` as is.

If `input` is not an object, the function checks if it is a string. If it is, it applies the `transform` function to `input` and returns a Promise that resolves to the transformed value.

Finally, if `input` is neither an object nor a string (which means it must be `undefined`), the function simply returns the `input` as is.

This function is likely used in the context of a Prisma application, where the `set` property is used in update operations to specify the new value for a field. The `transform` function could be used to perform operations such as hashing a password before storing it in the database.

❖ Can you provide an example of how to use the `before` function from the `lodash` library?

Tabnine looks like this.

Tabnine

The selected code defines a function called `before` that accepts two arguments: `n` and `func`. The function returns a new function that invokes `func` a maximum of `n` times. If the number of calls to the restricted function is greater than or equal to `n`, subsequent calls to the restricted function return the result of the last `func` invocation.

The code starts by defining a variable called `result` and setting its value to `undefined`. Then, it checks if the value of `func` is a function and throws a `TypeError` if it's not.

The function then returns a new function that accepts arguments. This inner function starts by decrementing the value of `n` by 1. If `n` is greater than 0, it invokes `func` with `this` binding and the arguments passed to the restricted function, and stores the result in `result`.

The inner function then checks if `n` is less than or equal to 1. If it is, it sets the value of `func` to `undefined` to prevent further invocations.

Finally, the inner function returns `result`.

The code exports the `before` function as the default export.

❖ What happens if the value of 'n' is initially set to 0 in the 'before' function?

References (1)

1. `before.ts: 0-32`

Tabnine

The selected code is a helper function that can be used to transform a string field update input before sending it to the database. It takes two arguments: `input`, which can be a string, an object with a `set` property that is a string, or undefined, and `transform`, which is a function that takes a string and returns a Promise that resolves to a transformed string.

The function first checks if the input is an object and if it has a `set` property that is a string. If so, it returns an object with a `set` property that is the result of calling the `transform` function on the `set` property.

If the input is not an object or it does not have a `set` property, the function checks if the input is a string. If so, it returns the result of calling the `transform` function on the input string.

If the input is neither an object nor a string, the function returns the input unchanged.

❖ What is the purpose of the `transformString Field UpdateInput` function?

References (1)

1. `prisma.util.ts:12-28`

And Amazon CodeWhisperer gives us this view:

The code you have selected is a utility function for updating Prisma database fields.

It takes an input value and a transform function as generic type parameters. The transform function allows modifying the value before updating the database (e.g. hashing a password).

It checks the type of the input and handles different cases:

- If the input is an object with a 'set' property, it applies the transform to that property and returns an object of the original type.
- If the input is a string, it applies the transform and returns the result as the original type.
- If the input is another type of object, it simply returns it without modifying.
- For all other types like undefined/null it returns the input unchanged.

This allows updating different field types like single values or nested objects in a consistent way, while applying any transforms before the actual database update.

The generic typing ensures the returned value is still of the expected original type after the transform.

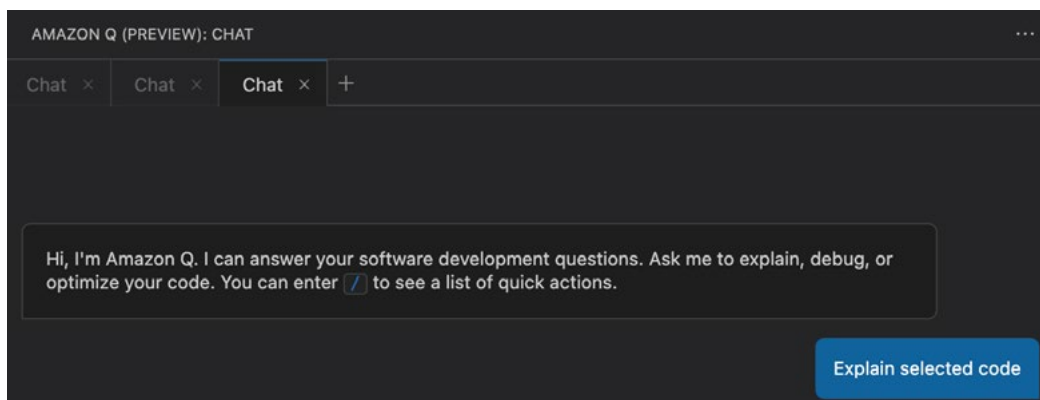
Let me know if any part needs more explanation!

Another example is how **Tabnine** offers convenient tiny helpers above each function – ‘test’, ‘explain’, ‘document’, ‘ask’. These helpers are always readily accessible, eliminating the need to navigate through nested menus or type in a chat.

```
tabnine: test | explain | document | ask
```

```
export async function transformStringFieldUpdateInput<
  T extends undefined | string | { set?: string }
>(input: T, transform: (input: string) => Promise<string>): Promise<T> {
  if (typeof input === "object" && typeof input?.set === "string") {
    return { set: await transform(input.set) } as T;
  }
  if (typeof input === "object") {
    if (typeof input.set === "string") {
      return { set: await transform(input.set) } as T;
    }
    return input;
  }
  if (typeof input === "string") {
    return (await transform(input)) as T;
  }
  return input;
}
```

Another convenient feature available exclusively in **Amazon CodeWhisperer** is tabs in the chat interface. These tabs help users maintain distinct contexts for each task or question, eliminating the need to clear the conversation every time they ask something not related to the previous conversation.



While each tool is worth consideration, our primary focus is on exploring the functionality of GitHub Copilot Business in more detail throughout the following chapters.

Key takeaways

- ▶ Generative AI offers **significant cost savings** by automating repetitive coding tasks, reducing manual effort, and accelerating development cycles.
- ▶ Generative AI has the potential to **reshape programming languages** to suit AI's needs, which could lead to new coding methods and tools.
- ▶ The widespread adoption of generative AI may lead to **changes in the IT job market**.
- ▶ By providing resources and support for developers to acquire AI skills, **IT leaders can enhance productivity** and competitiveness within their organizations.
- ▶ Among the evaluated AI tools, **GitHub Copilot Business emerges as the preferred choice** for developers due to its dynamic code suggestions and customization options.

In the next chapter, we will dive deeper into the capabilities and benefits of GitHub Copilot Business, exploring its applications across various roles in software development.



02

Coding with GitHub Copilot Business



Ultimately, it's not going to be about man versus machine. It's going to be about man with machines."

– Satya Nadella

Project scope and tool stack

In this chapter, we showcase the capabilities of GitHub Copilot Business in AI-assisted development. In addition to exploring the theoretical aspects, we also delve into our practical findings obtained while refactoring the Interview portal—our internal system, with backend developed with NodeJS and NestJS and frontend built with React.

At Intellias, the Interview Portal facilitates the interviewing of candidates for job openings. The portal serves as a tool for collecting interview feedback from both recruiters and technical interviewers, building analytics on soft and hard skills, giving recommendations on skills gaps, defining candidate job level, as well as finding applicant feedback records for further reuse.

As the portal is built with NodeJS and ReactJS, we will give most of our examples in TypeScript. However, to engage our technical readers who work with .NET and Java, we will also include examples for these technologies.

Below you can see the interface of the Interview Portal, whose code base we explore in more detail in the next sections.

PORTAL

Applicants

Applicants list (All)

Full Name	Job profile	Interview resolution	Applicant status
Emma Heath	Manager	Manager	Closed
Adern Jarvis	JavaScript Engineer (WebGL)	Junior	Hired
Abi Dawson	Python Engineer	Senior	Closed
Jason Mclean	Language Teacher	Senior	Closed
Jared Keith	IT Support Specialist	Middle	Closed
Tim Harper	CRM Engineer (Amdocs)	Middle	Closed
Adam John	Business Analyst	Requirements Analyst	Closed
Anisha Curry	Python Engineer	Senior	Closed
Nikolas Lee	.NET Engineer	Senior	Closed
Skyla Alexander	.NET Engineer	Senior	Closed

Filters

Status in the company

☒ All ☐ Hired ☐ Not hired

Type of technical interviewers

☒ All ☐ Certified ☐ Not Certified

Resolution

Applicant Employment Type

Applicant origin (Internal/External)

Interview date

Start date → End date

Scoring

From

None

By profile

Profile

Clear All

Submit

Tim Harper
EMP-900169 / cv

Manager Feedback

History

Position

Job Profile

Status

Required level

Date

C++ Test Vacancy

C++ Engineer (Generalist)

Finished

Senior

Interview Feedback

Download report

Job Level

Interview score

Skill analytics

Senior

4 Hire

General feedback:

Based on the demonstrated soft skills and technical competencies, I highly recommend considering the candidate for hiring. His clear communication, adaptability, and problem-solving abilities are essential traits for success in collaborative environments. His commitment to learning and professionalism further enhance their suitability for the role. Hiring him would not only benefit the team but also contribute positively to the organization's goals and objectives.

Soft skills:

The candidate shown great soft skills vital for teamwork and project success. His clear communication ensures everyone understands complex tech concepts. He adapts well to changes and solves problems creatively. He's detail-oriented, meets deadlines, and is committed to learning. His professionalism and team spirit create a positive work environment. He should keep using these skills to excel in his role.

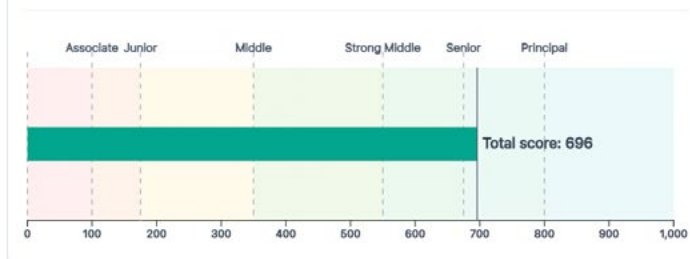
Recommendation:

No specific recommendations

Expert:

 **John Brown**
Principal C++ Developer

The number of conducted interviews: 5



Results visualization

C++ skillcheck

Coding Skills

★★★★★ Proficient

Code completion and suggestions

GitHub Copilot Business is an AI-powered code completion assistant designed to enhance coding efficiency, speed, and user experience. The assistant can suggest entire lines and even blocks of code. What sets GitHub Copilot Business apart is its sophisticated understanding of the coding context in which it is deployed. By integrating GitHub Copilot Business into software development workflows, we embrace a tool that not only accelerates code production but also elevates the quality of our output through context-aware assistance.

Let's get back to refactoring the Interview Portal and start writing CRUDs for the backend. This involves creating modules, controllers, services, and models, which we do using GitHub Copilot Business.

As you can see from the examples, after the developer clicks Tab, Copilot suggests updates marked in gray that should be added to the file.

After we create a file called Interview.controller.ts and start typing, Copilot generates a bunch of code for RESTFUL controllers. Copilot is aware of the context. It understands your stack and suggests highly relevant completions. All we need to do is review the suggestions and press the Tab button to accept them.

Disclaimer

As you can see from the examples, after the developer clicks Tab, Copilot suggests updates marked in gray that should be added to the file.

```
export class InterviewController {
  constructor(private readonly interviewService: InterviewService) {}

  @Get()
  async findAll(): Promise<Interview[]> {
    return this.interviewService.findAll();
  }

  @Get('id')
  async findOne(@Param('id') id: string): Promise<Interview> {
    return this.interviewService.findOne(id);
  }

  @Post()
  async create (@Body() createInterviewDto: CreateInterviewDto): Promise<Interview> {
    return this.interviewService.create(createInterviewDto);
  }

  @Put('id')
  async update(@Param('id') id: string, @Body() updateInterviewDto: UpdateInterviewDto): Promise<Interview> {
    return this.interviewService.update(id, updateInterviewDto);
  }

  @Delete('id')
  async remove (@Param('id') id: string): Promise<void> {
    return this.interviewService.remove(id);
  }
}
```

JAVA

```
package com.example.copilotworkshop.controller;

import com.example.copilotworkshop.model.CreateInterviewDto;
import com.example.copilotworkshop.model.Interview;
import com.example.copilotworkshop.model.UpdateInterviewDto;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/interviews")
public class InterviewController {

    private final InterviewService interviewService;

    public InterviewController(InterviewService interviewService) {
        this.interviewService = interviewService;
    }

    @PostMapping
    public Interview createInterview(@RequestBody CreateInterviewDto
interviewDto) {
        return interviewService.createInterview(interviewDto);
    }

    @PutMapping("/{id}")
    public Interview updateInterview(@PathVariable String id,
@RequestBody UpdateInterviewDto interviewDto) {
        return interviewService.updateInterview(id, interviewDto);
    }

    @DeleteMapping("/{id}")
    public void remove(@PathVariable String id) {
        interviewService.removeInterview(id);
    }

    @GetMapping("/{id}")
    public Interview findOne(@PathVariable String id) {
        return interviewService.findOne(id);
    }

    @GetMapping
    public List<Interview> findAll() {
        return interviewService.findAll();
    }
}
```

.NET

```
using System.Data.Entity;
using WebApplication1.Models;

namespace WebApplication1
{
    public class InterviewRepository
    {
        private readonly InterviewsDbContext _context;

        public InterviewRepository(InterviewsDbContext context)
        {
            _context = context;
        }

        public void Add(InterviewModel interview)
        {
            _context.Interviews.Add(interview);
            _context.SaveChanges();
        }

        public IEnumerable<InterviewModel> GetAll()
        {
            return _context.Interviews.ToList();
        }

        public InterviewModel GetById(int id)
        {
            return _context.Interviews.FirstOrDefault(i => i.Id == id);
        }

        public void Update(InterviewModel interview)
        {
            _context.Entry(interview).State = EntityState.Modified;
            _context.SaveChanges();
        }

        public void Delete(int id)
        {
            var interview = GetById(id);
            _context.Interviews.Remove(interview);
            _context.SaveChanges();
        }
    }
}
```

Next, we update the import paths, and it works! Quite impressive, isn't it?

```
import {Get, Post, Put, Delete, Param} from '@nestjs/common';

export class InterviewController {
  constructor(private readonly interviewService: InterviewService) {}

  @Get()
  async findAll(): Promise<Interview[]> {
    return this.interviewService.findAll();
  }

  @Get('id')
  async findOne(@Param('id') id: string): Promise<Interview> {
    return this.interviewService.findOne(id);
  }

  @Post()
  async create (@Body() createInterviewDto: CreateInterviewDto): Promise<Interview> {
    return this.interviewService.create(createInterviewDto);
  }

  @Put('id')
  async update(@Param('id') id: string, @Body() updateInterviewDto: UpdateInterviewDto):
  Promise<Interview> {
    return this.interviewService.update(id, updateInterviewDto);
  }

  @Delete('id')
  async remove (@Param('id') id: string): Promise<void> {
    return this.interviewService.remove(id);
  }
}
```

OK, let's proceed. Imagine that we are developing an interview service and need to filter interviews by an applicant's ID. All we need to do is start typing the method name. Copilot will grasp the context and provide a suggestion:

```
import { Interview } from "@interview-portal/lib-interfaces";

export class InterviewsService {
    private interviews: Interview[] = [];

    filterInterviewsByApplicantId(| applicantId: string): Interview[] {
        return this.interviews.filter(interview => interview.applicantId === applicantId);
    }
}
```

TYPESCRIPT

```
public InterviewModel GetInterview(int id)|
{
    return _context.Interviews.FirstOrDefault(i => i.Id == id);
}
```

.NET

```
public List<Interview>|filterInterviewsByApplicantId(String applicantId) {
    return interviews.stream()
        .filter(interview -> interview.getApplicantId().equals(applicantId))
        .toList();
}
```

JAVA

Let's explore a slightly more complex scenario: we need to remove a technical interview and store it in the archive for future reporting needs.

```
async deleteInterviewAndArchive(
    id: string,
    emplId: string,
): Promise <{id: string; applicantId: string; emplId: string; data: ITechnicalFeedback;}> {
    const deletedResource = await this.technicalFeedbacksDbService.deleteInterview(id, emplId);
    if(deletedResource) {
        const archivedResource = await this.technicalFeedbacksArchiveDbService.add(deletedResource);
    }
    return null;
}
```

TYPESCRIPT

Copilot understands pretty well what needs to be done here. You might consider updating the return values or wrapping this request in a single transaction later, but it has just saved you a few precious minutes of your time.

Now, let's delve into the last example, which is backend-specific. Suppose you need to write plain SQL queries (we're using CosmosDB for these examples). If Copilot cannot locate the corresponding models in your project, it suggests a plain SQL query for a method trying to fetch all the recruiter feedback.

```
async getAll(emplId: string): Promise<
{ id: string; emplId: string; data: IRecruiterFeedback; _ts: number }[]> {
  const { resources } = await this.container.items.query({
    query: `SELECT * from c WHERE c.emplId = "${emplId}"`,
  }).fetchAll();

  return resources;
}
```

TYPESCRIPT

```
public List<InterviewModel> GetAllInterviews()
{
  List<InterviewModel> interviews = new List<InterviewModel>();
  using (mssqlConnection) {
    mssqlConnection.Open();
    SqlCommand command = new SqlCommand("SELECT * FROM Interviews", mssqlConnection);
    using (SqlDataReader reader = command.ExecuteReader()) {
      while (reader.Read()) {
        InterviewModel interview = new InterviewModel();
        interview.Id = reader.GetInt32(0);
        interview.Name = reader.GetString(1);
        interview.Position = reader.GetString(2);
        interview.Date = reader.GetString(3);
        interview.Status = reader.GetString(4);
        interviews.Add(interview);
      }
    }
  }
  return interviews;
}
```

.NET

```
public List<Interview> getAllInterviews() {
  List<Interview> interviews = new ArrayList<>();
  try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery("SELECT * FROM interviews");
    while (resultSet.next()) {
      Interview interview = new Interview();
      // Assuming Interview has id, name, and date fields
      interview.setId(resultSet.getString("id"));
      interview.setName(resultSet.getString("name"));
      interview.setDate(resultSet.getDate("date"));
      interviews.add(interview);
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
  return interviews;
}
```

JAVA

Let's explore Copilot's capabilities in general programming. Imagine that we need to write a sorting algorithm. Most of us can't remember the exact implementation of every sorting algorithm and might waste time googling the answer. But not with Copilot! The AI writes the algorithm for you, saving you a ton of time. Fascinating, isn't it?

```
public bubbleSortFeedbacksByTime|(feedbacks: any[]) {  
  for (let i = 0; i < feedbacks.length; i++) {  
    for (let j = 0; j < feedbacks.length - 1 - i; j++) {  
      if (feedbacks[j]._ts < feedbacks[j + 1]._ts) {  
        const temp = feedbacks[j];  
        feedbacks[j] = feedbacks[j+1];  
        feedbacks[j+1] = temp;  
      }  
    }  
  }  
  return feedbacks;  
}
```

TYPESCRIPT

```
public int[] BubbleSort|(int[] arr)  
{  
  int n = arr.Length;  
  for (int i = 0; i < n - 1; i++)  
  {  
    for (int j = 0; j < n - i - 1; j++)  
    {  
      if (arr[j] > arr[j + 1])  
      {  
        // swap temp and arr[j]  
        int temp = arr[j];  
        arr[j] = arr[j + 1];  
        arr[j + 1] = temp;  
      }  
    }  
  }  
  return arr;  
}
```

.NET

```
public int[] bubbleSort|(int[] arr) {  
  int n = arr.length;  
  for (int i = 0; i < n - 1; i++) {  
    for (int j = 0; j < n - i - 1; j++) {  
      if (arr[j] > arr[j + 1]) {  
        int temp = arr[j];  
        arr[j] = arr[j + 1];  
        arr[j + 1] = temp;  
      }  
    }  
  }  
  return arr;  
}
```

JAVA

Refactoring and optimization

GitHub Copilot Business can assist in pinpointing code smells and areas that need refactoring. By analyzing the code base, Copilot can suggest improvements such as extracting methods, renaming variables for clarity, or replacing complex conditional blocks with more readable alternatives. Developers can prompt Copilot with comments or questions about potential refactoring strategies, turning it into an interactive code review tool.

- ▶ **Code simplification.** GitHub Copilot Business can suggest more concise and readable versions of existing code blocks, helping reduce complexity and improve maintainability.
- ▶ **Identifying code smells.** By analyzing the code context, GitHub Copilot Business can highlight potential “code smells” — indicators of deeper problems in the code that may need refactoring. While it does not explicitly label these issues, its suggestions often lead developers to recognize and address them.
- ▶ **Automated pattern recognition.** GitHub Copilot Business identifies repetitive code patterns and suggests ways to abstract them into reusable functions or components, thereby adhering to the DRY (Do not repeat yourself) principle.

Let’s examine some practical examples and ask Copilot to refactor the Interview Portal’s controller, checking the access rights for the users for CRUD operations. You can do this by opening the prompt and writing your request there.

please, refactor this method to check if the user has rights to read, delete, and update the interviews



Accept

Discard



1 change



```
import { Get, Post, Put, Delete, Param, UseGuards } from '@nestjs/common';
import { AuthGuard } from 'your-auth-guard-package';

export class InterviewController {
  constructor(private readonly interviewService: InterviewService) {}

  @Get()
  @UseGuards(AuthGuard)
  async findAll(): Promise<Interview[]> {
    return this.interviewService.findAll();
  }

  @Get('id')
  @UseGuards(AuthGuard)
  async findOne(@Param('id') id: string): Promise<Interview> {
    return this.interviewService.findOne(id);
  }

  @Post()
  @UseGuards(AuthGuard)
  async create(@Body() createInterviewDto: CreateInterviewDto): Promise<Interview> {
    return this.interviewService.create(createInterviewDto);
  }

  @Put('id')
  @UseGuards(AuthGuard)
  async update(@Param('id') id: string, @Body() updateInterviewDto: UpdateInterviewDto): Promise<Interview> {
    return this.interviewService.update(id, updateInterviewDto);
  }

  @Delete('id')
  @UseGuards(AuthGuard)
  async remove(@Param('id') id: string): Promise<void> {
    return this.interviewService.remove(id);
  }
}
```



please, refactor this controller to check all the user has rights to read, delete, and update the interviews



```
public class InterviewController : Controller {
    public List<InterviewModel> Interviews { get; set; }
    public InterviewModel GetInterview() {
        return new InterviewModel(1, "John Doe", "Software Developer", "2021-01-01", "Scheduled");
    }
    public InterviewModel GetInterview(int id) {
        return Interviews.FirstOrDefault(x => x.Id == id);
    }
    public void DeleteInterview(int id) {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        Interviews.Remove(interview);
    }
}
```

```
public class InterviewController : Controller
{
    public List<InterviewModel> Interviews { get; set; }

    // Check if the user has rights to read the interview
    [HttpGet]
    public IActionResult GetInterview(int id)
    {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        if (interview == null)
        {
            return NotFound();
        }

        // Check user rights here

        return Ok(interview);
    }

    // Check if the user has rights to delete the interview
    [HttpDelete]
    public IActionResult DeleteInterview(int id)
    {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        if (interview == null)
        {
            return NotFound();
        }

        // Check user rights here

        Interviews.Remove(interview);

        return NoContent();
    }

    // Check if the user has rights to update the interview
    [HttpPut]
    public IActionResult UpdateInterview(int id, [FromBody] InterviewModel updatedInterview)
    {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        if (interview == null)
        {
            return NotFound();
        }

        // Check user rights here

        interview.Name = updatedInterview.Name;
        interview.Position = updatedInterview.Position;
        interview.Date = updatedInterview.Date;
        interview.Status = updatedInterview.Status;

        return Ok(interview);
    }
}
```

.NET (BEFORE)

.NET (AFTER)

JAVA

```
package com.example.copilotworkshop.controller;

import com.example.copilotworkshop.model.CreateInterviewDto;
import com.example.copilotworkshop.model.Interview;
import com.example.copilotworkshop.model.UpdateInterviewDto;
import java.util.List;
import lombok.RequiredArgsConstructor;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequiredArgsConstructor
public class InterviewController {

    private final InterviewService interviewService;

    @PreAuthorize("hasAuthority('READ_INTERVIEW')")
    @GetMapping
    public List<Interview> findAll() {
        return interviewService.findAll();
    }

    @PreAuthorize("hasAuthority('READ_INTERVIEW')")
    @GetMapping("/{id}")
    public Interview findOne(@PathVariable String id) {
        return interviewService.findOne(id);
    }

    @PostMapping
    public Interview createInterview(@RequestBody
    CreateInterviewDto interviewDto) {
        return interviewService.createInterview(interviewDto);
    }

    @PreAuthorize("hasAuthority('UPDATE_INTERVIEW')")
    @PutMapping("/{id}")
    public Interview updateInterview(@PathVariable String id,
    @RequestBody UpdateInterviewDto interviewDto) {
        return interviewService.updateInterview(id, interviewDto);
    }

    @PreAuthorize("hasAuthority('DELETE_INTERVIEW')")
    @DeleteMapping("/{id}")
    public void removeInterview(@PathVariable String id) {
        interviewService.removeInterview(id);
    }
}
```



Github Copilot

Hi @Intellias, how can I help you?

— I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and [share feedback](#) so that we can learn and improve.



Intellias

Please refactor this controller to check if the user has rights to read, delete and update the interview. Please use Spring security



Github Copilot



To implement the security checks, you can use Spring Security's `@PreAuthorize` annotation. This annotation can be used to determine if a user has the necessary permissions to perform certain actions.

First, you need to enable method security in your Spring Security configuration.

This can be done by adding `@EnableGlobalMethodSecurity(prePostEnabled = true)` to your configuration class.

```
import org.springframework.security.config.
@EnableGlobalMethodSecurity(prePostEnabled
= true)public class SecurityConfig extends
```

💡 How can I define and assign authorities to users in my user management system?

Ask Copilot a question or type '/' for commands

📄 InterviewController.java ×

+



Alternatively, we may want to integrate Swagger into our application and document APIs with the help of Copilot:

please, update this controller with swagger docs



Accept

Discard



1 change



```
import {Get, Post, Put, Delete, Param} from '@nestjs/common';
import {ApiTags, ApiOperation} from "@nestjs/sqagger";

@ApiTags("interviews")
export class InterviewController {
  constructor(private readonly interviewService: InterviewService) {}

  @Get()
  @ApiOperation({ summary: "Get all interviews" })
  async findAll(): Promise<Interview[]> {
    return this.interviewService.findAll();
  }

  @Get('id')
  @ApiOperation({ summary: "Get an interview by ID" })
  async findOne(@Param('id') id: string): Promise<Interview> {
    return this.interviewService.findOne(id);
  }

  @Post()
  @ApiOperation({ summary: "Create a new interview" })
  async create (@Body() createInterviewDto: CreateInterviewDto): Promise<Interview> {
    return this.interviewService.create(createInterviewDto);
  }

  @Put('id')
  @ApiOperation({ summary: "Update an interview by ID" })
  async update(@Param('id') id: string, @Body() updateInterviewDto: UpdateInterviewDto): Promise<Interview> {
    return this.interviewService.update(id, updateInterviewDto);
  }

  @Delete('id')
  @ApiOperation({ summary: "Delete an interview by ID" })
  async remove (@Param('id') id: string): Promise<void> {
    return this.interviewService.remove(id);
  }
}
```

TYPESCRIPT

Impressive, isn't it? As you can see, Copilot can easily handle these tasks in just a few seconds. Now, what if we want to do a more complicated refactoring? In the previous section, we were developing a bubble sorting algorithm for our interviews. As you probably know, its complexity is not ideal and in most cases equals $O(n^2)$. Let's ask Copilot to a merge sort algorithm to reduce the complexity.

update this method to use merge sort sorting algorithm



Accept

Discard



1 change



```
public mergeSortFeedbacks(feedbacks: IFeedback[]): IFeedback[] {
    if(feedbacks.length <= 1) {
        return feedbacks;
    }

    const middle = Math.floor(feedbacks.length / 2);
    const left = feedbacks.slice(0, middle);
    const right = feedbacks.slice(middle);
    return this.merge(
        this.mergeSortFeedbacks(left),
        this.mergeSortFeedbacks(right)
    );
}

private merge(left: IFeedback[], right: IFeedback[]): IFeedback[] {
    let result: IFeedback[] = [];
    let leftIndex = 0;
    let rightIndex = 0;
    while(leftIndex < left.length && rightIndex < right.length) {
        if(left[leftIndex].rating >= right[rightIndex].rating) {
            result.push(left[leftIndex]);
            leftIndex++;
        } else {
            result.push(right[rightIndex]);
            rightIndex++;
        }
    }
    result = result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
    return result;
}
```



update this method to use merge sort sorting algorithm



.NET (BEFORE)

```
public int[] BubbleSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                // swap temp and arr[j]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```

.NET (AFTER)

```
public int[] MergeSort(int[] arr) {
    if (arr.Length <= 1) {
        return arr;
    }

    int mid = arr.Length / 2;
    int[] left = new int[mid];
    int[] right = new int[arr.Length - mid];
    for (int i = 0; i < mid; i++) {
        left[i] = arr[i];
    }
    for (int i = mid; i < arr.Length; i++) {
        right[i - mid] = arr[i];
    }
    left = MergeSort(left);
    right = MergeSort(right);
    return Merge(left, right);
}

public int[] Merge(int[] left, int[] right) {
    int[] result = new int[left.Length + right.Length];
    int i = 0, j = 0, k = 0;
    while (i < left.Length && j < right.Length) {
        if (left[i] < right[j]) {
            result[k] = left[i];
            i++;
        }
        else {
            result[k] = right[j];
            j++;
        }
        k++;
    }
    while (i < left.Length) {
        result[k] = left[i];
        i++;
        k++;
    }
    while (j < right.Length) {
        result[k] = right[j];
        j++;
        k++;
    }
    return result;
}
```

JAVA (BEFORE)

```
public int[] bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```

Github Copilot

Hi @Intellias, how can I help you?

— I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and [share feedback](#) so that we can learn and improve.

Intellias

Update a method bubbleSort to use merge sort sorting algorithm

Github Copilot

Sure, I can help you with that. Here's a step-by-step plan:

1. Create a helper method merge that takes two sorted arrays and merges them into a single sorted array.
2. In the mergeSort method, check if the array length is less than or equal to 1. If it is, return the array as it is already sorted.
3. Find the middle index of the array.
4. Recursively call mergeSort on the left half and the right half of the array.
5. Merge the two sorted halves using the merge method.

Here's the updated InterviewService class with the mergeSort method:

JAVA (AFTER)

```
public int[] mergeSort(int[] arr) {
    if (arr.length <= 1) {
        return arr;
    }
    int midpoint = arr.length / 2;
    int[] left = new int[midpoint];
    int[] right;
    if (arr.length % 2 == 0) {
        right = new int[midpoint];
    } else {
        right = new int[midpoint + 1];
    }
    for (int i = 0; i < midpoint; i++) {
        left[i] = arr[i];
    }
    for (int j = 0; j < right.length; j++) {
        right[j] = arr[midpoint + j];
    }
    int[] result = new int[arr.length];
    left = mergeSort(left);
    right = mergeSort(right);
    result = merge(left, right);
    return result;
}

private int[] merge(int[] left, int[] right) {
    int[] result = new int[left.length + right.length];
    int leftPointer, rightPointer, resultPointer;
    leftPointer = rightPointer = resultPointer = 0;
    while (leftPointer < left.length || rightPointer < right.length) {
        if (leftPointer < left.length && rightPointer < right.length) {
            if (left[leftPointer] < right[rightPointer]) {
                result[resultPointer++] = left[leftPointer++];
            } else {
                result[resultPointer++] = right[rightPointer++];
            }
        } else if (leftPointer < left.length) {
            result[resultPointer++] = left[leftPointer++];
        } else if (rightPointer < right.length) {
            result[resultPointer++] = right[rightPointer++];
        }
    }
    return result;
}
```

It works! We've just optimized our code to use a more efficient sorting algorithm.

Refactoring and optimization

With GitHub Copilot Business, you can complete complex refactoring tasks, spanning multiple classes. However, the effectiveness of the tool in handling such tasks depends on several factors, such as the clarity of instructions, the complexity of the code base, and the articulation of required changes through comments and prompts.

GitHub Copilot Business generates code and refactoring suggestions based on the context provided by the developer. To effectively handle complex refactoring tasks involving multiple classes, consider these tips:

1

- ▶ **Break down the task.** If the refactoring task is complex and involves multiple steps or changes across various classes, try breaking it down into smaller, more manageable tasks. GitHub Copilot Business can be more effective when dealing with specific, well-defined tasks.

2

- ▶ **Provide clear instructions.** When working on a specific refactoring task, provide clear and concise comments or instructions in your code about what you want to achieve. GitHub Copilot Business uses the context provided by these instructions to generate relevant code suggestions.

3

- ▶ **Iterative refinement.** You might not get the perfect solution in the first attempt. Use GitHub Copilot Business suggestions as a starting point and iteratively refine the code. You can adjust the instructions or comments based on the suggestions provided by Copilot to guide it towards the desired outcome.

4

- ▶ **Manual review and testing.** After using GitHub Copilot Business for refactoring, manually review the suggested changes and thoroughly test the refactored code. This ensures that the refactoring does not introduce any bugs or regressions and the code's readability, maintainability, and performance are improved.

5

- ▶ **Leverage other tools.** For complex refactoring tasks, consider using GitHub Copilot Business in conjunction with other tools and IDE features designed for refactoring. Many IDEs feature powerful refactoring tools designed specifically for languages such as Java, which can help automate certain aspects of the refactoring process.

Summing up, GitHub Copilot Business can assist with complex refactoring tasks, including those involving multiple classes. However, ensuring the quality and integrity of the refactored code requires clear instructions, iterative refinement, and a combination of tools and manual review.

Debugging and bug fixing

GitHub Copilot Business can spot potential bugs in real-time as developers write and execute code. By analyzing the context and common coding patterns, Copilot can suggest corrections and improvements on the fly, while also understanding the root causes of various application issues.

Let's consider this scenario: we are developing the Applicant results module, which collates various data pieces such as self-evaluation, manager feedback, interview results, and more. While adding numerous services, controllers, and external libraries, we suddenly encounter an error shown in the console (NestJS-specific):

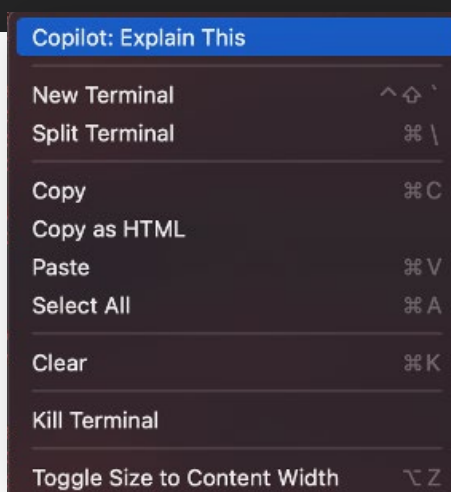
```
[Nest] 17652 - 03/24/2024, 1:55:24 PM LOG [InstanceLoader] CacheModule dependencies initialized +3ms
[Nest] 17652 - 03/24/2024, 1:55:24 PM LOG [InstanceLoader] CacheModule dependencies initialized +0ms
[Nest] 17652 - 03/24/2024, 1:55:24 PM ERROR [ExceptionHandler] Nest can't resolve dependencies of the ApplicantResultsService (CACHE_MANAGER, ApplicantResultsDbService, LoggerService, ?, ApplicantsDbService). Please make sure that the argument EmployeesDbService at index [3] is available in the ApplicantResultsModule context.
```

Potential solutions:

- Is ApplicantResultsModule a valid NestJS module?
- If EmployeesDbService is a provider, is it part of the current ApplicantResultsModule?
- If EmployeesDbService is exported from a separate @Module, is that module imported within ApplicantResultsModule?

```
@Module({
  imports: [ /* the Module containing EmployeesDbService */ ]
})
```

If you are not very familiar with NestJS, figuring out what's wrong can be challenging. However, there's a simple solution: just select the error from the terminal and ask Copilot to explain it.



The error message from your terminal indicates that NestJS can't resolve the dependencies of the `ApplicantResultsService`. Specifically, it's having trouble with the `EmployeesDbService` at index [3].

This usually means that `EmployeesDbService` is not provided in the `ApplicantResultsModule` or any of its imported modules.

You can fix this by adding `EmployeesDbService` to the `providers` array in the `ApplicantResultsModule`, or in a module that `ApplicantResultsModule` imports. Here's how you might do this:

```
@Module({
  providers: [ApplicantResultsService, EmployeesDbService],
})
export class ApplicantResultsModule {}
```

If `EmployeesDbService` is provided in a different module, you'll need to import that module into `ApplicantResultsModule`:

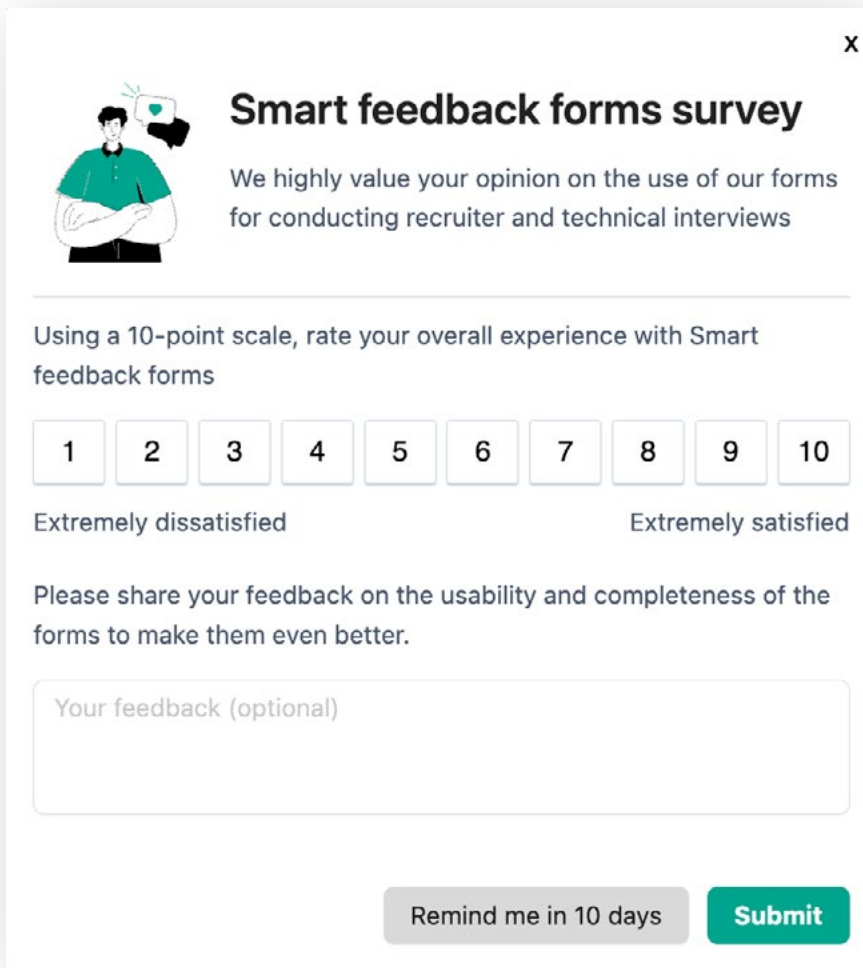
```
@Module({
  imports: [EmployeesModule],
  providers: [ApplicantResultsService],
})
export class ApplicantResultsModule {}
```

In this case, `EmployeesModule` should look something like this:

```
@Module({
  providers: [EmployeesDbService],
  exports: [EmployeesDbService],
})
export class EmployeesModule {}
```

Ah, right! We have forgotten to include the `EmployeesModule` in the `imports` array of our `applicants-result.module.ts` file! Now, it should work perfectly fine!

```
@Module({
  imports: [
    DbModule,
    CoeInfoModule,
    ApplicantsModule,
    CacheModule.register({
      ttl: 60 * 60 * 24 * 7,
    }),
  ],
  providers: [
    ApplicantResultsService,
    ApplicantResultsDbService,
    ApplicantResultsListener,
  ],
  controllers: [ApplicantResultsController],
})
export class ApplicantResultsModule {}
```



A screenshot of a 'Smart feedback forms survey' modal. At the top left is an illustration of a person with a speech bubble. The title 'Smart feedback forms survey' is in bold. Below it, a subtitle reads: 'We highly value your opinion on the use of our forms for conducting recruiter and technical interviews'. The main instruction says: 'Using a 10-point scale, rate your overall experience with Smart feedback forms'. Below this is a row of 10 numbered buttons (1-10). Under button 1 is the text 'Extremely dissatisfied' and under button 10 is 'Extremely satisfied'. A paragraph follows: 'Please share your feedback on the usability and completeness of the forms to make them even better.' Below this is a text input field with the placeholder 'Your feedback (optional)'. At the bottom are two buttons: 'Remind me in 10 days' (disabled) and 'Submit' (active).

Let's move on to the next bug. We are implementing a feature that will collect user feedback about our platform. We prompt users for feedback once every two months. Additionally, users have an option to postpone giving feedback by 10 days. Visually, the prompt we are developing resembles the one shown below:

Let's envision a method that determines whether we need to display the prompt or not:

```
async checkIfShouldAskForFeedback(user: IUserMetadataFromContext) {
  const response = {
    shouldProvideFeedback: true,
  };

  const record = await this.feedbackDbService.getLastUserFeedback(user.email);
  const timeout =
    (record.isPostponed ? TIMEOUT_IN_DAYS_FOR_POSTPONED : TIMEOUT_IN_DAYS) * MILLIS_IN_DAYS;
  const lastFeedbackProvidedDate = new Date(record.createdDate).getTime();
  const nextFeedbackExpectedDate = lastFeedbackProvidedDate + timeout;
  if (nextFeedbackExpectedDate - Date.now() > 0) {
    response.shouldProvideFeedback = false;
  }

  return response;
}
```

Let's ask Copilot to fix the scenario where the last feedback of the current user is missing, particularly when the user accesses the Interview Portal for the first time. We can use the prompt `/fix` command.

```
async checkIfShouldAskForFeedback(user: IUserMetadataFromContext) {  
  const response = {  
    shouldProvideFeedback: true,  
  };  
};
```

`/fix`

To fix the issue, I would add a null check for the `record` variable before accessing its properties.

Accept

Discard



1 change



```
const record = await this.feedbackDbService.getLastUserFeedback(user.email);  
if (record) {  
  const timeout = (record.isPostponed ? TIMEOUT_IN_DAYS_FOR_POSTPONED : TIMEOUT_IN_DAYS) * MILLIS_IN_DAYS;  
  const lastFeedbackProvidedDate = new Date(record.createdDate).getTime();  
  const nextFeedbackExpectedDate = lastFeedbackProvidedDate + timeout;  
  if (nextFeedbackExpectedDate - Date.now() > 0) {  
    response.shouldProvideFeedback = false;  
  }  
}  
return response;  
}
```

As you can see, Copilot adds an "if" statement to check if the record exists.

For a more advanced detection of bugs, we can use a new Copilot agent called "@workspace". Although the response time of this agent might be a bit slower, the quality and accuracy of the provided solutions might be better.

Based on our practical experience with GitHub Copilot Business, let's walk through some key debugging and bug-fixing best practices:

1

- **Leverage Copilot to understand bug contexts.** Sometimes, the value of GitHub Copilot Business lies not only in the direct fixing of bugs but also in its ability to provide context or alternative approaches to a problem. Use it to explore different solutions or understand how similar issues have been addressed in other projects.

2

- **Collaborate with Copilot on debugging.** Treat Copilot as a virtual pair programmer. Explain the problem as if you were explaining it to a human collaborator. Comment on your code with the bug you're trying to solve or the expected behavior. Copilot's suggestions can give you fresh perspectives that you might not have considered.

3

- **Continuously review and test.** Thoroughly review and test every suggestion provided by GitHub Copilot Business, just like any other code change. Ensure that the suggested fix not only resolves the bug but also does not introduce any new issues or dependencies into your code base.

Key takeaways

- ▶ Integrating GitHub Copilot Business into the development workflow **serves as a virtual pair programmer**, providing real-time guidance, refactoring, and debugging.
- ▶ GitHub Copilot Business **provides context-aware code completions** and **helps identify code smells**, which results in faster delivery and higher code quality.
- ▶ While **handling complex refactoring tasks** involving multiple classes, we recommend breaking down tasks, providing clear instructions, and combining Copilot's suggestions with manual review.

In the next chapter, let's explore various applications of AI in software development, from personal assistance and documentation to automated testing, DevOps transformation, and the emergence of AI-driven no-code and low-code platforms.



03

Advanced AI capabilities

“

*Progress is impossible without change,
and those who cannot change their
minds cannot change anything.”*

– George Bernard Shaw

AI in software development

As we continue with our AI research, it's clear that GitHub Copilot Business has very impressive capabilities when working with natural language. Let's explore the capabilities that go beyond real-time suggestions and fixes, focusing on personal consultation on technical matters, generating technical documentation, and writing appropriate commit messages based on recent code changes.

Personal assistance

Imagine that we are starting to develop the Interviewer Portal and need to decide on which stack to go with: Azure or AWS. Each stack's database plays a critical role in this decision.

While resorting to Google for answers is an option, it may entail sifting through a pile of irrelevant, outdated, or overly complex information, potentially leading to confusion. Let's try to shorten our journey and ask Copilot to provide us with a comprehensive summary.

What is CosmosDB and which benefits it has

 GitHub Copilot

Used 1 reference >

Cosmos DB is a globally distributed, multi-model database service provided by Microsoft Azure for managing data at a planet-scale. It's designed to enable you to write globally distributed applications, hence the name Cosmos.

Here are some of the benefits of Cosmos DB:

1. **Global Distribution:** You can distribute your data to any number of Azure regions worldwide to ensure your data is close to your users.
2. **Multi-Model Support:** It supports multiple data models, including key-value, document, column-family, and graph.
3. **Multi-API Support:** It provides multiple APIs for accessing data including SQL, MongoDB, Cassandra, Tables, or Gremlin.
4. **Elastic Scalability:** It offers elastic scalability of throughput and storage worldwide.
5. **Low Latency:** It guarantees low latency reads and writes.
6. **Comprehensive SLAs:** It offers comprehensive Service Level Agreements (SLAs) for throughput, latency, availability, and consistency.
7. **Automatic Indexing:** It automatically indexes all data and allows you to use SQL-like language to query it.

After getting an initial overview, we can narrow down our inquiry and ask Copilot to compare CosmosDB and AWS DynamoDB.

can you compare it with Amazon DynamoDB?

 GitHub Copilot

Used 1 reference >

Sure, here's a comparison between Azure Cosmos DB and Amazon DynamoDB:

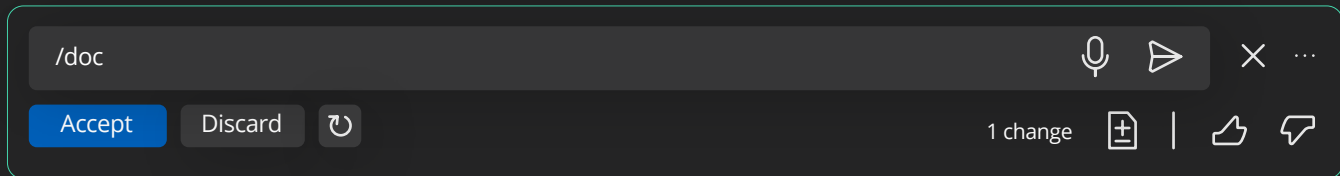
1. **Data Model:** Cosmos DB supports multiple data models (key-value, document, column-family, and graph), while DynamoDB primarily supports key-value and document data models.
2. **APIs:** Cosmos DB provides multiple APIs for data access, including SQL, MongoDB, Cassandra, Tables, or Gremlin. DynamoDB, on the other hand, provides a proprietary web service API.
3. **Global Distribution:** Both services offer global distribution, but Cosmos DB allows you to distribute your data to any number of Azure regions worldwide with a single click. DynamoDB requires you to manage replication across regions manually.
4. **Consistency Models:** Cosmos DB offers five consistency models (Eventual, Consistent Prefix, Session, Bounded Staleness, and Strong), providing a flexible approach to consistency based on your application's needs. DynamoDB offers two consistency models: eventual and strong.
5. **Pricing:** DynamoDB charges for read/write capacity, storage, and data transfer. Cosmos DB charges for provisioned throughput and consumed storage.
6. **Indexing:** Both support automatic indexing of all data. However, in DynamoDB, you need to define secondary indexes to query on attributes other than the primary key.
7. **Serverless:** Both offer serverless options, allowing you to pay only for the resources your application actually consumes.
8. **Integration:** DynamoDB integrates well with other AWS services, while Cosmos DB integrates well with Azure services.

Remember, the best choice between the two depends on your specific use case, the needs of your application, and the cloud platform you're most comfortable with.

This is a simple example of Copilot's capabilities as a personal assistant. For more advanced research, please see Chapter 4, where we explore how Copilot serves as a partner in pair programming, unveiling new approaches to the software development workflow.

Documentation

We all like to work on well-documented projects. But, when it comes to writing documentation ourselves, we usually aren't so keen. In the AI era, we can delegate this responsibility to Copilot. Now, it's as simple as selecting the class or method, opening a prompt, and running the `/doc` command!



```
/**
 * Creates a new interview.
 * @param interview - The interview object to be created.
 * @returns A promise that resolves to the created interview.
 */
async createInterview(interview: InterviewCopy): {
  return this.interviewsDbService.createInterview(interview);
}

/**
 * Retrieves all interviews.
 * @returns A promise that resolves to an array of interviews.
 */
async getInterviews(): {
  return this.interviewsDbService.getInterviews();
}

/**
 * Retrieves an interview by its ID.
 * @param id - The ID of the interview to retrieve.
 * @returns A promise that resolves to the retrieved interview.
 */
async getInterview(id: string): {
  return this.interviewsDbService.getInterview(id);
}

/**
 * Updates an existing interview.
 * @param id - The ID of the interview to update.
 * @param interview - The updated interview object.
 * @returns A promise that resolves to the updated interview.
 */
async updateInterview(id: string, interview: InterviewCopy): {
  return this.interviewsDbService.updateInterview(id, interview);
}
```

```
public class InterviewController : Controller
```



```
/|
```

doc	Add documentation comment for this symbol
explain	Explain the code
fix	Propose a fix for problems in the selected code
help	Get help on Copilot chat
optimize	Analyze and improve running time of the selected code
tests	Create unit tests for the selected code

```
/doc Command
```



```
{
    public List<InterviewModel> Interviews { get; set; }

    [HttpGet]
    [OpenApiOperation("GetInterview")]
    [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(InterviewModel))]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public IActionResult GetInterview(int id)
    {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        if (interview == null)
        {
            return NotFound();
        }

        // Check user rights here
        return Ok(interview);
    }

    [HttpDelete]
    [OpenApiOperation("DeleteInterview")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public IActionResult DeleteInterview(int id)
    {

```

```

public class InterviewController : Controller
{
    public List<InterviewModel> Interviews { get; set; }

    /// <summary>
    /// Retrieves an interview by its ID.
    /// </summary>
    /// <param name="id">The ID of the interview.</param>
    /// <returns>The interview with the specified ID.</returns>
    [HttpGet]
    [ApiOperation("GetInterview")]
    [ProducesResponseType(StatusCodes.Status200OK, Type = typeof(InterviewModel))]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public IActionResult GetInterview(int id)
    {
        var interview = Interviews.FirstOrDefault(x => x.Id == id);
        if (interview == null)
        {
            return NotFound();
        }

        // Check user rights here

        return Ok(interview);
    }
}

```

```

    /// <summary>
    /// Deletes an interview by its ID.
    /// </summary>
    /// <param name="id">The ID of the interview to delete.</param>
    /// <returns>No content if the interview is successfully deleted, otherwise NotFound.</returns>
    [HttpDelete]
    [ApiOperation("DeleteInterview")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    public IActionResult DeleteInterview(int id)
    {

```

Below is an example of JSDoc comments. Let's take a look at how Copilot can generate inline comments for methods in JavaScript and .NET.

```
async checkIfShouldAskForFeedback(user: IUserMetadataFromContext) {  
  
    const response = {  
        shouldProvideFeedback: true,  
    };  
  
    // |check if user has already provided feedback  
    const record = await this.feedbackDbService.getLastUserFeedback(user.email);  
  
    if(!record) {  
        return response;  
    }  
  
    const timeout =  
        (record.isPostponed ? TIMEOUT_IN_DAYS_FOR_POSTPONED : TIMEOUT_IN_DAYS) * MILLIS_IN_DAYS;  
  
    const lastFeedbackProvidedDate = new Date(record.createdDate).getTime();  
  
    const nextFeedbackExpectedDate = lastFeedbackProvidedDate + timeout;  
  
    if(nextFeedbackExpectedDate - Date.now() > 0) {  
        response.shouldProvideFeedback = false;  
    }  
  
    return response;  
}
```

TYPESCRIPT

```
public IActionResult UpdateInterview(int id, [FromBody] InterviewModel updatedInterview)  
{  
    //Get the interview by ID  
    var interview = Interviews.FirstOrDefault(x => x.Id == id);  
    if (interview == null)  
    {  
        return NotFound();  
    }  
  
    // Check user rights here  
  
    interview.Name = updatedInterview.Name;  
    interview.Position = updatedInterview.Position;  
    interview.Date = updatedInterview.Date;  
    interview.Status = updatedInterview.Status;  
  
    return Ok(interview);  
}
```

.NET

Another documentation challenge is to ask Copilot to generate a large doc in a README file for the CRON service that's responsible for cleaning outdated records from the database.

document how `#file:interview.cron.service.ts` works in the file `#file:README.md`

 GitHub Copilot

Used 2 reference >

Sure, here's a brief explanation of how the `InterviewCronService` works that you can add to your README.md:

`# InterviewCronService`

The ``InterviewCronService`` is a service in our application that is responsible for cleaning up interview copies in our database. It uses the NestJS Cron package to schedule this cleanup task to run every day at 2 AM.

`## How it works`

1. The ``cleanCopies`` method is decorated with the ``@Cron(CronExpression.EVERY_DAY_AT_2AM)`` decorator, which schedules it to run at 2 AM every day.
2. When ``cleanCopies`` runs, it first calls the ``getAllRecords`` method to fetch all interview copies from the database. This method fetches records in batches of 10 to avoid loading too many records into memory at once.
3. For each record, it checks if the record's ``createdDate`` is more than 30 days ago using the ``checkIfTimeHasPassed`` method. If it is, the record is deleted from the database.
4. After all records have been checked and any old records have been deleted, it logs how many items were deleted.

`## Error handling`

If an error occurs during this process, it is caught and logged, and an event is tracked using the ``LoggerService``.

`## Dependencies`

This service depends on the ``InterviewCopyDbService`` for interacting with the database, and the ``LoggerService`` for logging and event tracking.

This explanation can be added to the README.md file under a new section titled "InterviewCronService".

Automated testing with AI

The situation with tests is pretty much the same as with documentation: nobody likes writing them. But, few would argue that tests significantly improve software reliability, making it easier to maintain and refactor. As a result, good test coverage boosts both the developer experience and code readability, enhancing its quality and credibility.

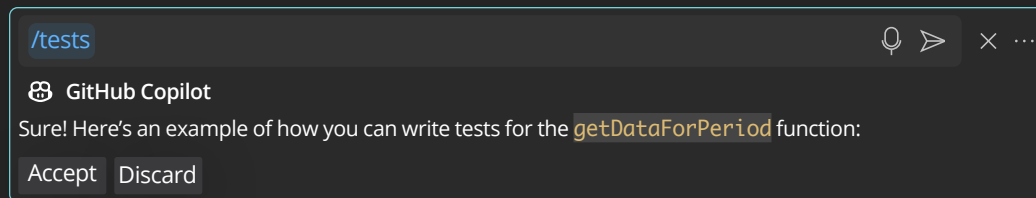
Let's explore how Copilot can help us with automated testing. Imagine there is a util function in our code:

```
import dayjs from "dayjs";

export enum TopInterviewersPeriods {
  CurrentMonth = "CurrentMonth",
  CurrentYear = "CurrentYear",
  LastWeek = "LastWeek",
  LastMonth = "LastMonth",
}

export function getDateForPeriod(period: TopInterviewersPeriods): {from: number; to: number;} {
  const currentDate = new Date();
  switch (period) {
    case TopInterviewersPeriods.CurrentMonth: {
      return {
        from: dayjs(currentDate).startOf("month").valueOf(),
        to: currentDate.valueOf(),
      };
    }
    case TopInterviewersPeriods.CurrentYear: {
      return {
        from: dayjs(currentDate).startOf("year").valueOf(),
        to: currentDate.valueOf(),
      };
    }
    case TopInterviewersPeriods.LastWeek: {
      return {
        from: dayjs(currentDate).subtract(1, "week").valueOf(),
        to: currentDate.valueOf(),
      };
    }
    case TopInterviewersPeriods.LastMonth: {
      const firstDayOfPreviousMonth = dayjs(currentDate)
        .subtract(1, "month")
        .startOf("month");
      const lastDayOfPreviousMonth = dayjs(currentDate)
        .subtract(1, "month")
        .endOf("month");
      return {
        from: firstDayOfPreviousMonth.valueOf(),
        to: lastDayOfPreviousMonth.valueOf(),
      };
    }
  }
}
```

The function is quite simple. We use it to filter interviews by periods: current month, current year, and last month. Let's use our prompt and ask Copilot to generate tests for this simple file.



```
import { getDateForPeriod } from './dates';

describe('getDateForPeriod', () => {
  it('should return the correct date range for CurrentMonth', () => {
    const result = getDateForPeriod('CurrentMonth');
    const currentDate = dayjs();
    expect(result.from).toBe(currentDate.startOf('month').valueOf());
    expect(result.to).toBe(currentDate.startOf('day').valueOf());
  });
  it('should return the correct date range for CurrentYear', () => {
    const result = getDateForPeriod('CurrentYear');
    const currentDate = dayjs();
    expect(result.from).toBe(currentDate.startOf('year').valueOf());
    expect(result.to).toBe(currentDate.startOf('day').valueOf());
  });
  ....
})
```

After accepting the suggestion and making minor tweaks (typically, Copilot doesn't include proper import statements), we get the following test suite, ready to be executed immediately:

```
import dayjs from "dayjs";
import { getDateForPeriod, TopInterviewersPeriods } from "../utils/dates";

describe("getDateForPeriod", () => {
  const currentDate = new Date();

  it("should return the correct date range for CurrentMonth", () => {
    const result = getDateForPeriod(TopInterviewersPeriods.CurrentMonth);
    const expected = {
      from: dayjs(currentDate).startOf("month").valueOf(),
      to: currentDate.valueOf(),
    };
    expect(result).toEqual(expected);
  });

  it("should return the correct date range for CurrentYear", () => {
    const result = getDateForPeriod(TopInterviewersPeriods.CurrentYear);
    const expected = {
      from: dayjs(currentDate).startOf("year").valueOf(),
      to: currentDate.valueOf(),
    };
    expect(result).toEqual(expected);
  });

  it("should return the correct date range for LastWeek", () => {
    const result = getDateForPeriod(TopInterviewersPeriods.LastWeek);
    const expected = {
      from: dayjs(currentDate).subtract(1, "week").valueOf(),
      to: currentDate.valueOf(),
    };
    expect(result).toEqual(expected);
  });

  it("should return the correct date range for LastMonth", () => {
    const result = getDateForPeriod(TopInterviewersPeriods.LastMonth);
    const firstDayOfPreviousMonth = dayjs(currentDate)
      .subtract(1, "month")
      .startOf("month");
    const lastDayOfPreviousMonth = dayjs(currentDate)
      .subtract(1, "month")
      .endOf("month");
    const expected = {
      from: firstDayOfPreviousMonth.valueOf(),
      to: lastDayOfPreviousMonth.valueOf(),
    };
    expect(result).toEqual(expected);
  });
});
```

```

PASS src/utils/dates.test.ts
  getDateForPeriod
    ✓ should return the correct date range for TopInterviewersPeriods.CurrentMonth (2 ms)
    ✓ should return the correct date range for TopInterviewersPeriods.CurrentYear (1 ms)
    ✓ should return the correct date range for TopInterviewersPeriods.LastWeek
    ✓ should return the correct date range for TopInterviewersPeriods.LastMonth

Test Suites: 1 passed, 1 total
Tests: 4 passed, 4 total
Snapshots: 0 total
Time: 0.421 s
Ran all test suites.

```

As you can see from the example above, testing with Copilot has never been easier! But what if we have a really complicated class and need to test some non-trivial behavior? That's not a problem either. Just provide more context after running the `/test` command. Let's try to do so using our Interviews CronJob Service and InterviewRepository in TypeScript:

`/tests`

Copilot generated code may be incorrect

```

import { Injectable } from '@nestjs/common';
import { Cron, CronExpression } from '@nestjs/schedule';
import { InterviewCopyDbService } from '../interview-copy.db.service';
import { InterviewCopy } from '@interview-portal/lib-interfaces';
import { LoggerService } from '../logger/logger.service';
import { EventLogTypes } from '../logger/event-log-types';

const COPY_LIFETIME_IN_DAYS = 30;
const MILLIS_IN_DAYS = 86_400_000;

@Injectable()
export class InterviewCronService {
  constructor(
    private readonly copiesDbService: InterviewCopyDbService,
    private readonly logger: LoggerService,
  ) {}

  private checkIfTimeHasPassed(copy: InterviewCopy) {
    if (!copy.createdDate) {
      return true;
    }
  }

```

So, what has been generated? First of all, Copilot understood it needed to mock the dependencies on other services and created the corresponding mocks.

```
describe('InterviewCronService', () => {
  let service: InterviewCronService;
  let copiesDbService: InterviewCopyDbService;
  let logger: LoggerService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        InterviewCronService,
        {
          provide: InterviewCopyDbService,
          useValue: {
            getAll: jest.fn(),
            deleteCopy: jest.fn()
          },
        },
        {
          provide: LoggerService,
          useValue: {
            log: jest.fn(),
            error: jest.fn(),
            trackEvent: jest.fn(),
          },
        },
      ],
    }).compile();

    service = module.get<InterviewCronService>(InterviewCronService);
    copiesDbService = module.get<InterviewCopyDbService>(InterviewCopyDbService);
    logger = module.get<LoggerService>(LoggerService);
  });

  afterEach(() => {
    jest.clearAllMocks();
  });
});
```

And what about the help with test cases? Let's take a look.

Copilot has generated two working test cases: one for a successful execution and another for an error state. You might have noticed that the tests are not ideal. Sometimes, Copilot might try to mock private methods of your class. Other times, it tries to test things that have nothing to do with your code, such as logger implementation or external services calls.

Nevertheless, even for complex tests, Copilot provides an excellent starting point. You don't need to perform any of these tasks manually: mocking, creating test case logic, searching docs to find the name of a specific class or method to mock error states—Copilot will handle it for you.

The only thing you need to keep in mind is when dealing with large, complex tests, it's important to take an incremental approach, adding complexity and additional context step by step instead of providing a bunch of text to the Copilot prompt.

```
describe('cleanCopies', () => {
  it('should delete copies that have expired', async () => {
    const mockRecords = [
      { id: 1, createdAt: new Date('2022-01-01') },
      { id: 2, createdAt: new Date('2022-01-02') },
      { id: 3, createdAt: new Date('2022-01-03') },
    ];
    const mockDeletedCount = 2;
    jest.spyon(service, 'getAllRecords').mockResolvedValue(mockRecords);
    jest.spyon(service, 'checkIfTimeHasPassed').mockReturnValue(true);
    jest.spyon(copiesDbService, 'deleteCopy').mockResolvedValue();
    await service.cleanCopies();
    expect(service.getAllRecords).toHaveBeenCalled();
    expect(service.checkIfTimeHasPassed).toHaveBeenCalledTimes(mockRecords.length);
    expect(copiesDbService.deleteCopy).toHaveBeenCalledTimes(mockDeletedCount);
    expect(logger.log).toHaveBeenCalledWith(
      `Cleaned interview copies DB. Deleted ${mockDeletedCount} items, finished at ${expect.any(Date)}`,
    );
  });

  it('should handle errors and log them', async () => {
    const mockError = new Error('Something went wrong');
    jest.spyon(service, 'getAllRecords').mockRejectedValue(mockError);
    jest.spyon(logger, 'error').mockImplementation();
    jest.spyon(logger, 'trackEvent').mockImplementation();
    await service.cleanCopies();
    expect(service.getAllRecords).toHaveBeenCalled();
    expect(logger.error).toHaveBeenCalledWith(mockError);
    expect(logger.trackEvent).toHaveBeenCalledWith('CriticalUnhandledErrorOccurred', mockError);
  });
});
```

Coverage improvements

By scanning the code structure and logic, Copilot can identify areas where test coverage is insufficient. It then generates additional tests to fill these gaps. To enhance test coverage, you should adhere to a common approach, providing precise prompts. A good prompt example is: “Add one more test when API fails, and the result should throw an exception.”

Using overly broad prompts may result in useless suggestions, failing to enhance the quality of test coverage. A bad example that illustrates this would be: “Add more unit tests, including negative to improve test coverage as much as possible.”

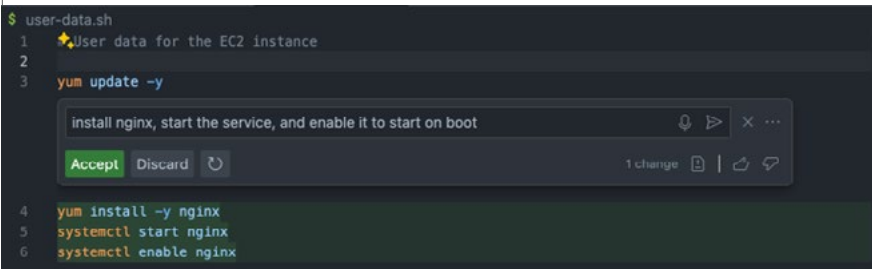
AI for DevOps

While GitHub Copilot Business excels in generating code and programming components, it faces challenges in DevOps and infrastructure-related functions. As a discipline, DevOps requires a deep understanding of project architecture, security measures, and infrastructure optimization—factors that can be too intricate for automation using standard code templates. Moreover, DevOps solutions rely heavily on context and security needs, which vary widely among projects. All these factors make Copilot helpful only in the hands of an experienced DevOps Engineer.

Practical coding scenarios

As a DevOps Engineer, using a tool like GitHub Copilot Business can significantly enhance your productivity and efficiency. Let’s explore the tasks and types of code you can write or automate using Copilot.

Scripting for automation: You can use Copilot for writing scripts in various languages, for example, Bash, PowerShell, or Python. These scripts can automate deployments, manage servers, or handle cloud resource provisioning.

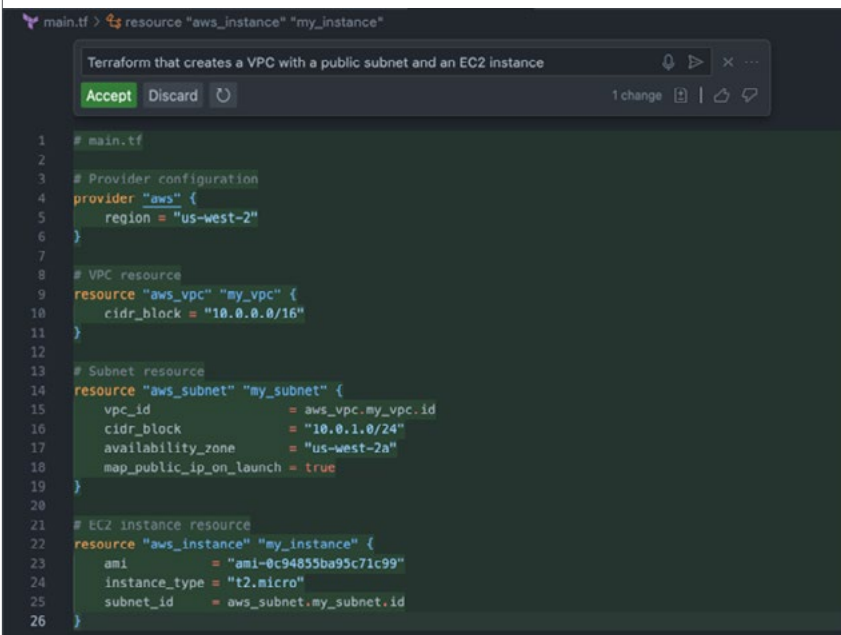


```
$ user-data.sh
1  ✨ User data for the EC2 instance
2
3  yum update -y
   install nginx, start the service, and enable it to start on boot
   Accept Discard 1 change
4  yum install -y nginx
5  systemctl start nginx
6  systemctl enable nginx
```

Known limitations.

- **Accuracy and context:** Copilot might not always understand the exact context or the specific needs of your automation task. Generated scripts may require significant review and adjustment.
- **Security concerns:** Automatically generated code might inadvertently include security vulnerabilities, especially if it suggests patterns or practices that are outdated or insecure.

Infrastructure as Code (IaC): Use Copilot to help you write IaC for tools such as Terraform, AWS CloudFormation, or Ansible. It's particularly useful in defining cloud infrastructure resources, configuration files, and Kubernetes manifests.



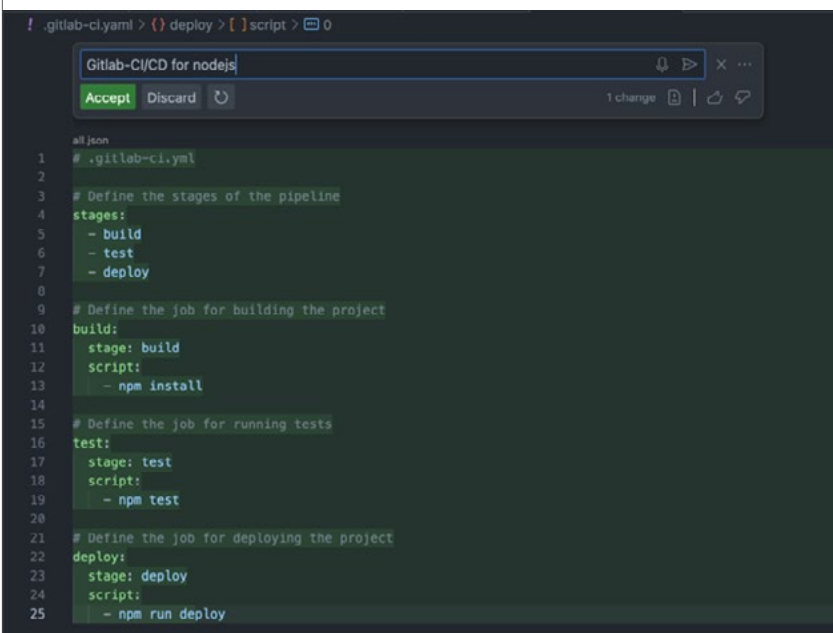
```
main.tf > resource "aws_instance" "my_instance"
Terraform that creates a VPC with a public subnet and an EC2 instance
Accept Discard 1 change

1  # main.tf
2
3  # Provider configuration
4  provider "aws" {
5    region = "us-west-2"
6  }
7
8  # VPC resource
9  resource "aws_vpc" "my_vpc" {
10   cidr_block = "10.0.0.0/16"
11 }
12
13 # Subnet resource
14 resource "aws_subnet" "my_subnet" {
15   vpc_id            = aws_vpc.my_vpc.id
16   cidr_block        = "10.0.1.0/24"
17   availability_zone  = "us-west-2a"
18   map_public_ip_on_launch = true
19 }
20
21 # EC2 instance resource
22 resource "aws_instance" "my_instance" {
23   ami           = "ami-0c94855ba95c71c99"
24   instance_type = "t2.micro"
25   subnet_id     = aws_subnet.my_subnet.id
26 }
```

Known limitations.

- **Complexity and specificity:** For complex infrastructure setups, Copilot's suggestions may not fully grasp the nuances of your architecture, leading to potentially incorrect or inefficient configurations.
- **Up-to-date practices:** The suggestions might not always reflect the most current best practices or features of IaC tools, depending on the training data it was built on.

Continuous integration and deployment: Employ Copilot for writing and optimizing CI/CD pipeline configurations. Whether you're working with GitHub Actions, GitLab CI, Jenkinsfiles, or other systems, Copilot can provide configuration suggestions and scripts.



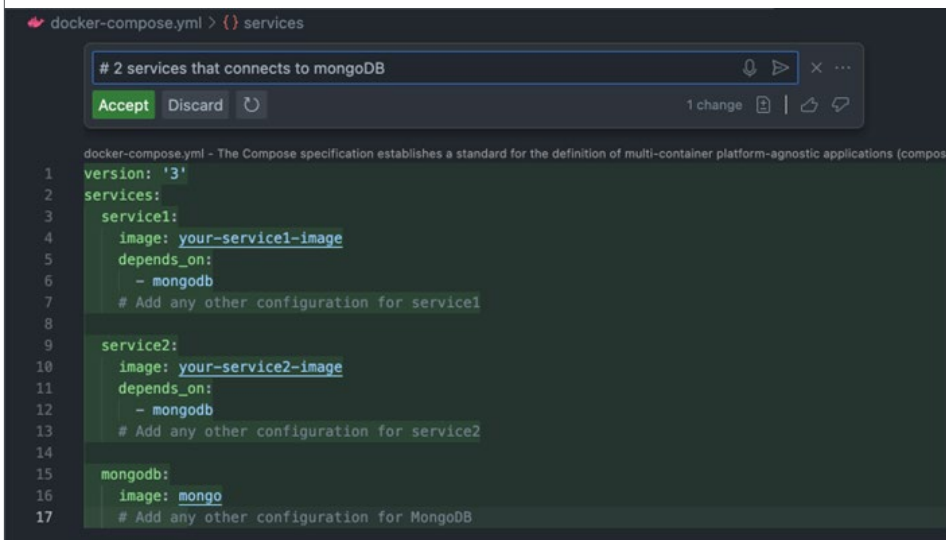
```
! .gitlab-ci.yml > {} deploy > {} script > 0
.gitlab-ci.yml for nodejs
Accept Discard 1 change

1  all:
2    # .gitlab-ci.yml
3
4    # Define the stages of the pipeline
5    stages:
6      - build
7      - test
8      - deploy
9
10   # Define the job for building the project
11   build:
12     stage: build
13     script:
14       - npm install
15
16   # Define the job for running tests
17   test:
18     stage: test
19     script:
20       - npm test
21
22   # Define the job for deploying the project
23   deploy:
24     stage: deploy
25     script:
26       - npm run deploy
```

Known limitations.

- **Tool-specific knowledge:** While Copilot can suggest configurations, understanding the intricacies of each CI/CD tool and its ecosystem is crucial for effective implementation, which might not be fully covered by Copilot's suggestions.
- **Custom requirements:** Custom or complex pipeline requirements might not be accurately captured by Copilot, requiring manual configuration.

Containerization: Use Copilot in crafting Dockerfiles or docker-compose files. It can offer suggestions for optimizations and best practices in containerization.



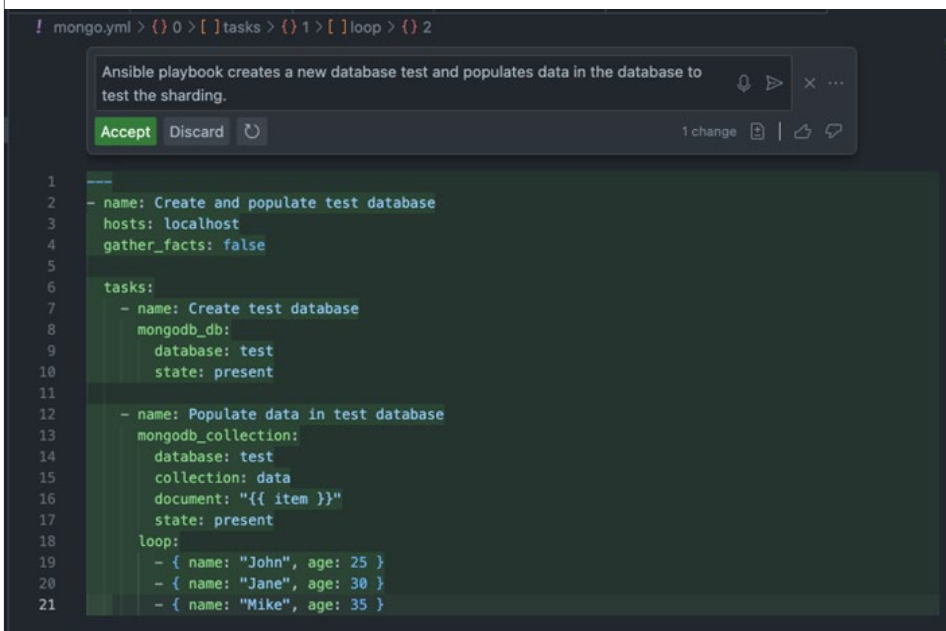
The screenshot shows a VS Code editor with a file named `docker-compose.yml`. A Copilot suggestion box is open, showing a snippet of YAML code for two services that connect to MongoDB. The code is as follows:

```
# 2 services that connects to mongoDB
version: '3'
services:
  service1:
    image: your-service1-image
    depends_on:
      - mongodb
    # Add any other configuration for service1
  service2:
    image: your-service2-image
    depends_on:
      - mongodb
    # Add any other configuration for service2
  mongodb:
    image: mongo
    # Add any other configuration for MongoDB
```

Known limitations.

- **Optimization and best practices:** Copilot might not always suggest the most optimized practices for containerization, particularly for advanced or very specific use cases.
- **Security:** Generated Dockerfiles or compose files might not adhere to security best practices, requiring thorough review and adjustments.

Configuration management: With tools like Ansible, Chef, or Puppet, Copilot can help you write playbooks, recipes, and manifests, while also suggesting improvements or alternative configuration methods.



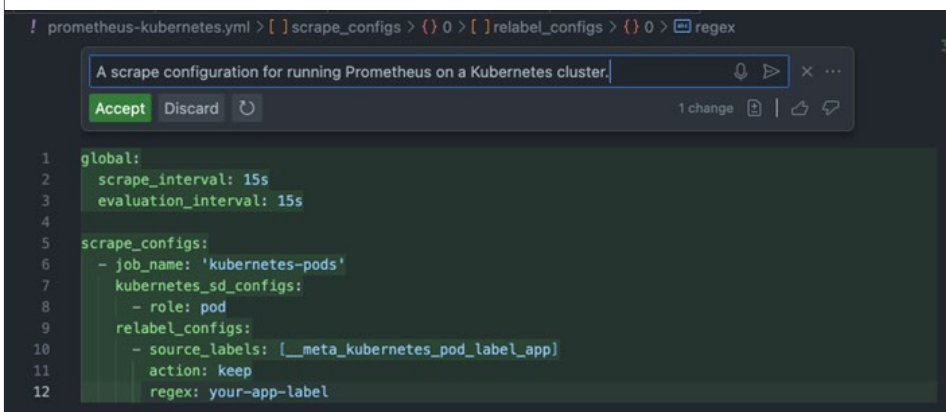
The screenshot shows a VS Code editor with a file named `mongo.yml`. A Copilot suggestion box is open, showing a snippet of an Ansible playbook. The code is as follows:

```
! mongo.yml > {} 0 > [] tasks > {} 1 > [] loop > {} 2
Ansible playbook creates a new database test and populates data in the database to
test the sharding.
Accept Discard 1 change
---
- name: Create and populate test database
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Create test database
      mongodb_db:
        database: test
        state: present
    - name: Populate data in test database
      mongodb_collection:
        database: test
        collection: data
        document: "{{ item }}"
        state: present
      loop:
        - { name: "John", age: 25 }
        - { name: "Jane", age: 30 }
        - { name: "Mike", age: 35 }
```

Known limitations.

- **Specific tool capabilities:** Each configuration management tool has its own set of capabilities and best practices, which Copilot might not fully encapsulate in its suggestions.
- **Dynamic environments:** Adapting to highly dynamic or specific environmental conditions might require a more nuanced understanding than Copilot can provide.

Monitoring and logging scripts: Copilot can streamline the process of writing scripts for monitoring and logging your infrastructure and applications.



The screenshot shows a VS Code editor with a file named `prometheus-kubernetes.yml`. A Copilot suggestion box is open, showing a snippet of a Prometheus scrape configuration. The code is as follows:

```
! prometheus-kubernetes.yml > [] scrape_configs > {} 0 > [] relabel_configs > {} 0 > regex
A scrape configuration for running Prometheus on a Kubernetes cluster.
Accept Discard 1 change
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'kubernetes-pods'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: your-app-label
```

Known limitations.

- **Custom metrics and logs:** Tailoring scripts to monitor and log custom metrics or specific system behaviors might go beyond the general suggestions of Copilot, requiring manual work.

Testing: Enhance your testing workflow with Copilot by generating unit tests, integration tests, and other testing scripts for infrastructure and applications

Known limitations.

- **Coverage and complexity:** Generating meaningful tests that cover edge cases or complex scenarios might be challenging for Copilot, potentially leading to inadequate test coverage.
- **Tool-specific syntax and practices:** Understanding the nuances of different testing frameworks and their best practices might require more than what Copilot can suggest.

```
test.py > ...

I need assistance in writing a Python script that makes a GET request to an API
Gateway endpoint and validates the response. The script should send a request to a
specific URL, and then verify that the response matches expected criteria, such as a
200 status code and certain JSON payload structure. It should handle different
scenarios like successful responses and potential error cases (like 404 or 500
errors). Can you help me create this script with appropriate error handling and
response validation?

Accept Discard 1 change

1 import requests
2
3 def make_request(url):
4     try:
5         response = requests.get(url)
6         response.raise_for_status() # Raise an exception for non-2xx status codes
7         return response.json()
8     except requests.exceptions.RequestException as e:
9         print(f"An error occurred: {e}")
10        return None
11
12 def validate_response(response):
13     if response is None:
14         return False
15
16     # Check if the response has a 200 status code
17     if response.status_code != 200:
18         return False
19
20     # Check if the response has the expected JSON payload structure
21     if "key1" not in response.json() or "key2" not in response.json():
22         return False
23
24     return True
25
26 # Replace 'your_api_endpoint' with the actual API Gateway endpoint URL
27 api_endpoint = 'your_api_endpoint'
28
29 # Make the request
30 response = make_request(api_endpoint)
31
32 # Validate the response
33 if validate_response(response):
34     print("Response is valid!")
35 else:
36     print("Response is not valid.")
```

Cloud-specific services: For work involving cloud providers like AWS, Azure, or GCP, Copilot can assist in writing SDK calls and automation scripts tailored to these platforms.

Known limitations.

- **Service complexity:** The vast array of services and their configurations that cloud providers offer might not be fully covered or accurately suggested by Copilot.
- **API changes:** Cloud services evolve rapidly, and Copilot's suggestions might not always align with the latest changes or best practices.

```
athena.py > ...

provide a python script using AWS CDK to set up an Amazon Athena query, an AWS
Glue crawler, and an S3 bucket

Accept Discard 1 change

1 from aws_cdk import (
2     core,
3     aws_athena as athena,
4     aws_glue as glue,
5     aws_s3 as s3,
6 )
7
8 class AthenaGlueS3Stack(core.Stack):
9     def __init__(self, scope: core.Construct, id: str, **kwargs) -> None:
10        super().__init__(scope, id, **kwargs)
11
12        # Create an S3 bucket
13        bucket = s3.Bucket(self, "MyBucket")
14
15        # Create an AWS Glue crawler
16        crawler = glue.CfnCrawler(
17            self, "MyCrawler",
18            name="MyCrawler",
19            role="arn:aws:iam:123456789012:role/service-role/AWSGlueServiceRole",
20            targets={
21                "s3Targets": [
22                    {
23                        "path": bucket.bucket_arn,
24                    }
25                ]
26            },
27        )
28
29        # Create an Amazon Athena query
30        database = athena.CfnDatabase(
31            self, "MyDatabase",
32            name="MyDatabase",
33        )
34
35        table = athena.CfnTable(
36            self, "MyTable",
37            database_name=database.name,
38            table_input={
39                "name": "MyTable",
40                "description": "MyTable",
41                "table_type": "EXTERNAL_TABLE",
42                "parameters": {
43                    "classification": "csv",
44                    "typeOfData": "file",
45                    "columnsOrderedBy": "name",
46                },
47                "storage_descriptor": {
48                    "location": bucket.bucket_arn,
49                    "input_format": "org.apache.hadoop.mapred.TextInputFormat",
50                    "output_format": "org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat",
51                    "columns": [
52                        {
53                            "name": "column1",
54                            "type": "string",
55                        }
56                    ],
57                }
58            },
59        )
```

Security automation: Automate security tasks with Copilot's help in scripting for vulnerability scanning, certificate renewal, or security policy enforcement.

Known limitations.

- **Comprehensive security practices:** Copilot might not be able to fully automate comprehensive security practices, especially those that require deep understanding of the security landscape and threat modeling. DO not delegate your security to AI entirely!

Documentation: Copilot can also help you write documentation, providing sample commands, boilerplate text, and configuration examples.

Known limitations.

- **Accuracy and completeness:** While Copilot can document code, ensuring accuracy, completeness, and alignment with your specific implementations requires manual review and enhancement.

```
!_gitlab-ci.yml> () njsscan > image
all json
1 # .gitlab-ci.yml
2
3 # Define the stages of the pipeline
4 stages:
5   - build
6   - test
7
8   create a GitLab CI/CD pipeline stage for running njsscan
9   Accept Discard 1 change
10
11 - njsscan
12 - deploy
13
14 # Define the stage for running njsscan
15 njsscan:
16   stage: test
17   image: node:latest
18   script:
19     - npm install -g njsscan
20     - njsscan
```

```
① Readme.MD > # Terraform instructions
write a documentation on installing and launching Terraform locally on a
Windows and MacOS systems
Accept Discard 1 change

1 # Terraform instructions
2
3 # Step 1: Installing Terraform
4 # For Windows:
5 # 1. Download the Terraform binary for Windows from the official website (https://www.terraform.io/downloads.html).
6 # 2. Extract the downloaded ZIP file.
7 # 3. Add the extracted folder to the system's PATH environment variable.
8
9 # For MacOS:
10 # 1. Install Homebrew if not already installed. Open Terminal and run the following command:
11 #   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
12 # 2. Install Terraform using Homebrew. Run the following command in Terminal:
13 #   brew install terraform
14
15 # Step 2: Verifying the Installation
16 # To verify the installation, open a new Terminal or Command Prompt window and run the following command:
17 # terraform version
18 # If the installation was successful, you should see the Terraform version information.
19
20 # Step 3: Launching Terraform
21 # 1. Create a new directory for your Terraform project.
22 # 2. Navigate to the project directory using Terminal or Command Prompt.
23 # 3. Create a new Terraform configuration file with the extension ".tf" (e.g., main.tf).
24 # 4. Write your Terraform configuration in the created file.
25 # 5. Initialize the Terraform project by running the following command:
26 #   terraform init
27 # 6. Apply the Terraform configuration to create or modify resources by running the following command:
28 #   terraform apply
29
30 # Note: Make sure to provide the necessary credentials and configuration details in your Terraform files.
31
32 # Example Usage:
33 # ---
34 # provider "aws" {
35 #   access_key = "YOUR_ACCESS_KEY"
36 #   secret_key = "YOUR_SECRET_KEY"
37 #   region     = "us-west-2"
38 # }
39
40 # resource "aws_instance" "example" {
41 #   ami           = "ami-0c94855ba95c71c99"
42 #   instance_type = "t2.micro"
43 # }
44 # ---
45
46 # This code provides a basic guide on installing and launching Terraform on Windows and MacOS systems.
```

AI-driven transformation in DevOps

AI is becoming more and more a part of the DevOps world. Let's explore its manifold benefits for DevOps Engineers, the latest breakthroughs, the most popular AI tools, potential risks, and future trends.

Breakthroughs	AI is making big changes in DevOps, especially in how tasks are automated and managed. First, it's making routine tasks in the CI/CD pipeline, such as building and deploying software, faster and more automated. This speeds up the whole process of software delivery. Next, AI helps us manage incidents by using predictive analytics. It can guess when problems might happen, letting teams solve issues before they cause any downtime. Lastly, AI is enabling systems that can fix themselves. This means the system can handle common problems on its own, keeping it stable and running smoothly without needing people to step in all the time.
Toolset	We're seeing a range of DevOps tools powered by AI that are changing the way teams work. First, there are monitoring tools that use machine learning. They can look through huge amounts of data and give useful insights about the health of the system, even predicting issues before they happen. Then, there are AI-powered chat systems, known as ChatOps platforms. These make it easier for teams to work together, share information, and get automated help when they need it. Finally, there's a growing use of AI in policy enforcement. This means using AI tools to make sure that the code follows certain rules and security measures throughout its development. All these tools are making a big difference in how DevOps teams do their jobs.
Benefits	AI is bringing a lot of advantages to DevOps. It's automating everyday tasks, which means DevOps Engineers have more time to focus on the big picture and creative parts of their work. Also, AI helps us make better decisions. With the analytics and insights it provides, teams can use their resources more effectively and improve how their systems perform. Another big plus is how AI helps teams collaborate. It's breaking down barriers between different groups: those who develop and those who operate the systems, encouraging everyone to communicate and collaborate more. These benefits are really changing how DevOps teams operate and succeed.
Risks	Using AI in DevOps also brings some risks that need attention. A big concern is the safety and privacy of data, especially when sensitive information is used in AI algorithms. It's really important to have strong security measures in place to protect this data. Then, there's the issue of depending too much on automation. While AI is helpful, it's crucial to keep human oversight, especially for important decisions. Lastly, integrating AI tools into existing DevOps processes can be tricky. Finding ways to smoothly blend these tools into current practices is a challenge that teams need to work on. Addressing these risks is key to making the most of AI in DevOps.
Future	Looking ahead, AI is set to play an even bigger role in DevOps. We expect to see more advanced integration of AI, bringing in new functions and better ways of working. Another exciting development could be AI making workflows more personalized. This means AI could adapt to how each person works and their preferences, making everything more efficient and user-friendly. Along with these advancements, it's important to think about the ethics of using AI. We need to make sure that AI is developed and used in a way that's responsible and considers its impact on daily routines in DevOps. Keeping an eye on these future trends is important as we continue to evolve with AI in the field. While GitHub Copilot Business serves as a powerful tool for enhancing productivity across a range of DevOps tasks, its limitations necessitate a careful and considered approach. Relying on it requires a balance between leveraging its AI-driven capabilities and applying human oversight to ensure accuracy, security, and alignment with best practices.

AI-driven no-code and low-code platforms

AI has emerged as a transformative technology, empowering businesses to automate processes, gain insights from data, and drive innovation. However, developing AI applications traditionally requires extensive coding knowledge and expertise. With the advent of low-code and no-code AI-driven platforms, which have democratized AI development, individuals with varying to no technical backgrounds can also harness the power of AI.

Low-code AI platforms

Low-code AI platforms provide an environment that allows users to build AI applications using visual interfaces and minimal coding. These platforms offer pre-built components, templates, and drag-and-drop functionality, reducing the complexity of coding tasks. With low-code platforms, developers can accelerate the development cycle, integrate with existing systems, and focus on business logic rather than on infrastructure.

Low-code platforms bridge the gap between developers and business users, empowering them to collaborate on AI projects. They enhance productivity by abstracting complex coding tasks, enabling faster application development and deployment. Such platforms are beneficial for building AI applications with standard use cases and moderate complexity. Developers with coding expertise can extend the capabilities of low-code platforms by incorporating custom code when needed.

No-code AI platforms

No-code AI platforms take the concept of simplicity a step further by eliminating the need for coding. These platforms focus on enabling users with minimal technical skills to build and deploy AI applications through intuitive interfaces, leveraging pre-built components and automation.

No-code platforms provide simplicity and accessibility, enabling users to focus on problem-solving rather than coding implementation. Such platforms are suitable for building AI applications with standard use cases and limited customization requirements. Developers with coding expertise can still benefit from no-code platforms by using them as prototyping tools and building MVPs.

Comparing low- and no-code platforms

Both low- and no-code platforms have distinct advantages and trade-offs. While low-code solutions require basic coding skills but offer greater flexibility, no-code platforms eliminate the need for coding but have some limitations in customization.

Metric	Low-code	No-code
Technical skill requirement	Basic coding skills are required, but coding complexity is significantly reduced.	No coding skills are required, enabling non-technical users to develop AI applications.
Flexibility and customization	Offers more flexibility and customization options through the inclusion of custom code.	Limited flexibility, as customization options are often pre-defined and constrained by the platform.
Development speed	Accelerates development speed by abstracting complex coding tasks and providing pre-built components.	Rapid application development with minimal time spent on coding, thanks to visual interfaces and pre-built components.
Use case complexity	Suitable for applications with moderate complexity and custom requirements.	Ideal for standard use cases with minimal customization needs.
User collaboration	Facilitates collaboration between developers and business users, leveraging visual modelling and shared development environments.	Enables broader user collaboration, allowing non-technical users to actively participate in AI application development.

Most rated low- and no-code platforms

Low-code platforms strike a balance between coding and simplicity, providing flexibility and customization options. The choice between low-code and no-code AI platforms depends on the complexity of the use case, the level of customization required, the pricing offering, and the technical skills available within the development team.

 Amazon SageMaker	Amazon SageMaker is a comprehensive machine learning platform provided by AWS. It offers both low-code and traditional coding options for building, training, and deploying AI models. SageMaker provides pre-built algorithms, automatic model tuning, and managed infrastructure to simplify the development process. It supports a wide range of use cases, such as recommendation systems, fraud detection, and time series forecasting.	Get started for free and request a pricing quote.
	Microsoft Lobe is a no-code AI platform that allows users to build machine learning models using visual interfaces and simple interactions. It supports image classification, object detection, and text classification tasks. Lobe provides an intuitive drag-and-drop interface for data labeling, model training, and model export. It enables users to deploy models in various formats. The best part is you can download and install it on your local machine.	Use for free.
	Apple CreateML is a no-code AI platform specifically designed for developers targeting Apple devices. It enables users to train machine learning models using drag-and-drop interfaces and graphical tools. CreateML focuses on computer vision tasks, natural language processing, and tabular data analysis. It integrates seamlessly with Apple's Core ML framework, allowing developers to deploy models directly on iOS, macOS, watchOS, and tvOS.	Use for free on macOS.
 Google's AutoML	Google AutoML is a suite of AI tools that provides both low-code and no-code options for building custom machine learning models. It simplifies the model development process by automating tasks such as data pre-processing, architecture selection, and hyperparameter tuning. Google AutoML offers specific products for vision, natural language, translation, and tabular data analysis, making it accessible for developers with varying levels of technical expertise.	Get started for free and request a pricing quote.
	Google Teachable Machine is a no-code AI platform that focuses on training models for image, sound, and gesture recognition. It uses a simple interface where users can upload their own datasets and create custom machine learning models without writing code. Teachable Machine is ideal for educational purposes, prototyping, and small-scale applications.	Use for free.
	Akkio is a low-code AI platform that focuses on automating machine learning workflows. It allows users to build and deploy AI models without extensive coding knowledge. Akkio offers a user-friendly interface with drag-and-drop functionality for data preparation, model training, and deployment. It supports a wide range of applications, including predictive analytics, natural language processing, and image recognition.	Check pricing details.
 DataRobot	DataRobot is an automated machine learning platform that combines low-code and no-code approaches. It offers a visual interface for data preparation and model building, as well as an automated feature engineering process. DataRobot automates repetitive tasks like model selection, hyperparameter tuning, and deployment. It supports a wide range of industries and use cases, including finance, healthcare, and marketing.	Use 30-day trial for free and check pricing details.
	ObviouslyAI is a low-code AI platform focused on automated predictive analytics. It simplifies the process of building and deploying predictive models by providing a visual interface and automated machine learning capabilities. Users can connect their data sources, select target variables, and let ObviouslyAI handle feature selection, model training, and deployment. It is suitable for business analytics, sales forecasting, and customer segmentation.	Use for free.

Key takeaways

- ▶ GitHub Copilot Business **offers personal consultation on technical matters**, generates technical documentation, and writes commit messages based on recent code changes.
- ▶ Copilot provides comprehensive summaries and comparisons of technical options, **shortening research time and enhancing decision-making**.
- ▶ Copilot **improves project documentation quality and test coverage** with minimal effort from developers.
- ▶ While Copilot is beneficial for many DevOps tasks, its **effectiveness in infrastructure-related functions is limited** due to the complexity required.
- ▶ Low-code and no-code platforms allow users with varying technical skills to **build AI applications through visual interfaces and pre-built components**, accelerating development while balancing flexibility and simplicity.

In the next chapter, let's explore best practices for the use of AI in pair programming, code review, quality assurance, and knowledge sharing.

Collaborative development with AI

“

*The art of programming is the skill of
controlling complexity.*

– Marijn Haverbeke

Pair programming with AI

Imagine yourself facing the challenge of developing a new feature for our Interview Portal. The feature aims to facilitate hiring decisions and consists by visualizing interview scores of candidates. The conceptual representation of what such a visualization might look like is shown below: candidates are scored during interviews, and the portal defines and visualizes the candidate's job level depending on the scores received.



What are your next steps as a software developer? Probably, you will start googling, diving into dozens of articles to select the best technologies for the task. After selecting a tool stack, you will create a proof of concept and get familiar with the chosen technologies. This may involve hours, if not days, of deciphering complex technical documentation and experimenting with examples that may not precisely fit your needs. Eventually, you might discover that the initially selected tool stack is inadequate for the task. While continuing your exploration, each technology might have its own challenges and uncertainties. But what about trying to save time and complete the task with the help of GitHub Copilot Business?

need to create advanced visualizations for employee performance metrics using React. What tool should I use and which pros and cons it have?



Used 1 reference >

For creating advanced visualizations in React, you might consider using D3.js or Recharts.

1. D3.js

- Pros:
 - Highly flexible and powerful, allowing for the creation of almost any kind of visualization.
 - Large community and extensive resources for learning.
- Cons:
 - Steeper learning curve compared to other libraries.
 - Integration with React can be complex because both manipulate the DOM.

2. Recharts

- Pros:
 - Easier to use and learn than D3.js.
 - Built specifically for React, so it integrates well.
 - Provides a set of modular charting components.
- Cons:
 - Less flexible than D3.js, as it's not possible to create any kind of visualization.
 - Smaller community and fewer resources for learning compared to D3.js.

Remember, the best tool depends on your specific needs, such as the complexity of the visualizations you need to create, your team's familiarity with these libraries, and the time you have to learn a new tool.

❖ What are some popular React libraries for handling form validation?

Ask Copilot a question or type "/" for commands



Sounds cool! With Copilot, we will try to bypass hours of tedious research and gain a comprehensive understanding of the available options and their benefits. Since we anticipate that the charts might be complicated, flexibility is our first priority, which makes D3 our first choice over Recharts. So, let's ask Copilot to explain what D3 entails, assuming we are not familiar with it.

Can you please provide a high level overview of D3 and its capabilities?

 Github Copilot

Used 1 reference >

D3.js, or Data-Driven Documents, is a JavaScript library that allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. It's primarily used for creating complex, dynamic, and interactive data visualizations in web browsers.

Here are some of its key capabilities:

1. **Data Visualization:** D3.js provides a wide range of visualization capabilities. You can create bar charts, pie charts, scatter plots, and much more. It also supports more complex types of visualizations like chord diagrams, treemaps, and force-directed graphs.
2. **Data Binding:** D3.js allows you to bind data to DOM elements. This means you can create elements and attributes based on your data.
3. **Transitions:** D3.js supports transitions, which means you can animate changes in your data over time. This is particularly useful for creating dynamic, interactive visualizations.
4. **Interactivity:** D3.js supports user interactions. You can create visualizations that respond to user input, like mouse clicks or keyboard input.
5. **Flexibility:** D3.js doesn't provide pre-built charts like some other libraries. Instead, it provides the building blocks to create custom visualizations. This means it's more flexible and powerful, but it also has a steeper learning curve.
6. **Integration:** D3.js can be integrated with other JavaScript frameworks and libraries, like React and Angular, although this can sometimes be complex due to both D3 and these frameworks wanting to control the DOM.
7. **Large Community:** D3.js has a large and active community, which means there are plenty of resources for learning and getting help. There are also many examples of D3.js visualizations available online.

💡 What are some popular React libraries for creating data visualizations?

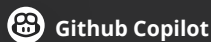
Ask Copilot a question or type '/' for commands



A promising start! The library seems to meet our goals. And here comes the most difficult part: how should we start the development? Should we read the D3 documentation? Should we google some examples and code snippets? Or, should we try writing something without understanding exactly how it works? No need—let's ask Copilot to help us generate a starting point.



Please, provide a code example using react and d3 with the following requirements: generate a horizontal bar chart with the employee performance levels (0-250 - bad performance, 250-500 - average performance, 500-750 - great performance, 750-1000 - outstanding performance). I should be able to hover at "zone of performance" to see the tooltip with the detailed description of each level.



Used 1 reference >

Here's a basic example of how you might create a horizontal bar chart with D3.js and React. This example uses the `react-tooltip` library for the tooltips.

First, install the necessary dependencies:

```
npm install d3 react-tooltip
```

Then, create your chart component:

```
import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';
import ReactTooltip from 'react-tooltip';
const data = [
  { performance: 'Bad Performance', range: [0, 250] },
  { performance: 'Average Performance', range: [250, 500] },
  { performance: 'Great Performance', range: [500, 750] },
  { performance: 'Outstanding Performance', range: [750, 1000] },
];

function BarChart() {
  const ref = useRef();

  useEffect(() => {
```

💡 How can I customize the colors of the bars in the D3.js horizontal bar chart?

Ask Copilot a question or type '/' for commands



We have received some code! Let's review what Copilot has provided.



Well, not a bad start, but definitely not what we want to have in the end!
Let's ask Copilot to refine the implementation further.



Update the `#file:Chart.tsx` with the following requirements:

1. Add xAxis with the levels of performance with the step === 100
2. Add yAxis with the name of employee (let it be "Anna Brown" for this case)
3. Make this axis labels white color



Used 1 reference >

Here's how you can update your `Chart.tsx` file to include the x-axis with performance levels and the y-axis with the employee's name:

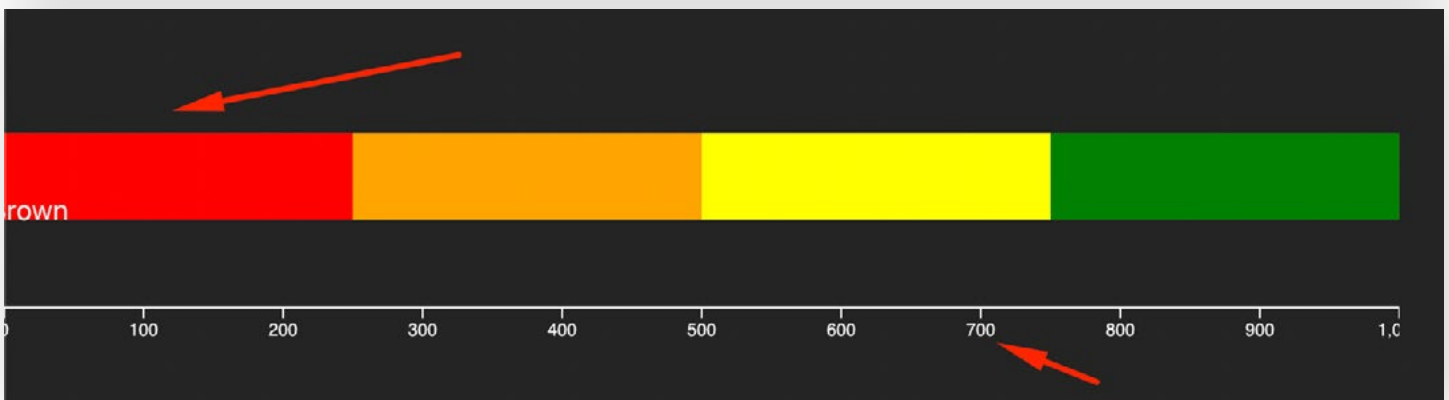
```
import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';
import ReactTooltip from 'react-tooltip';
const data = [
  { performance: 'Bad Performance', range: [0, 250], color: 'red' },
  { performance: 'Average Performance', range: [250, 500], color: 'orange' },
  { performance: 'Great Performance', range: [500, 750], color: 'green' },
  { performance: 'Outstanding Performance', range: [750, 1000], color: 'blue' },
];

function BarChart() {
  const ref = useRef<any>();

  useEffect(() => {
    const svg = d3.select(ref.current as any)
      .attr('width', 800);
  });
}
```

How can I add animation to the bars in React?

Ask Copilot a question or type '/' for commands



Now it's better, but still not ideal. Let's continue our session with Copilot and add a few more requirements:

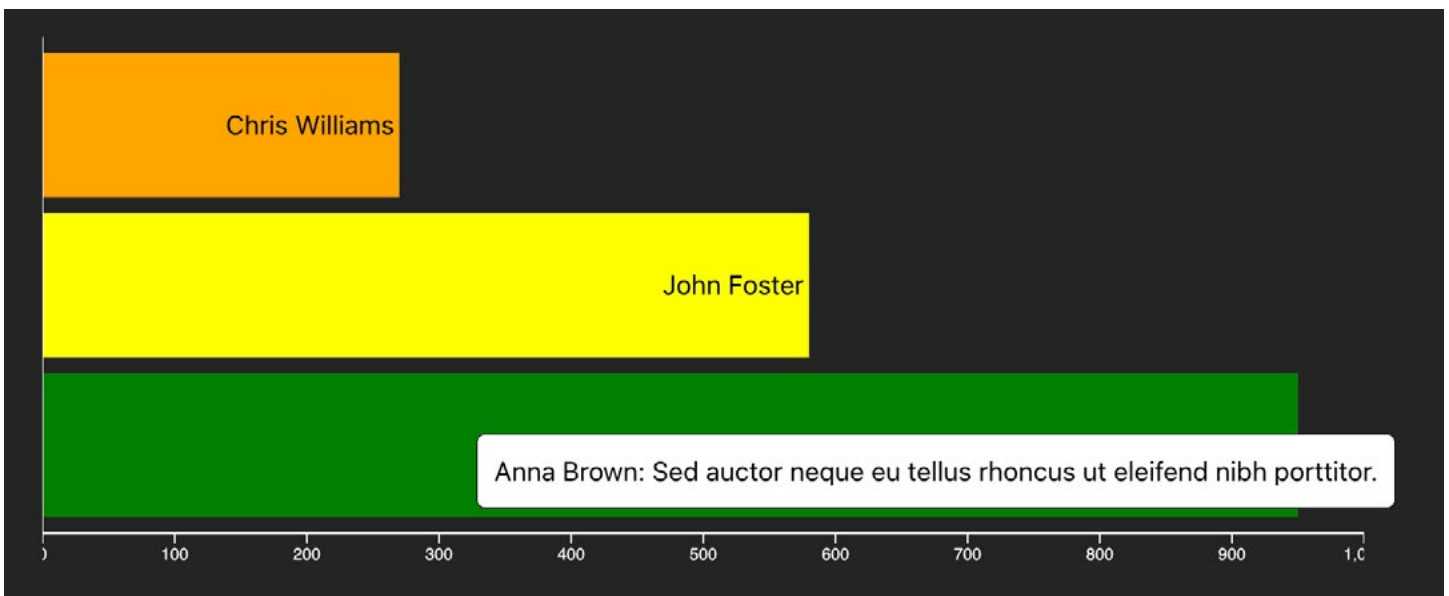
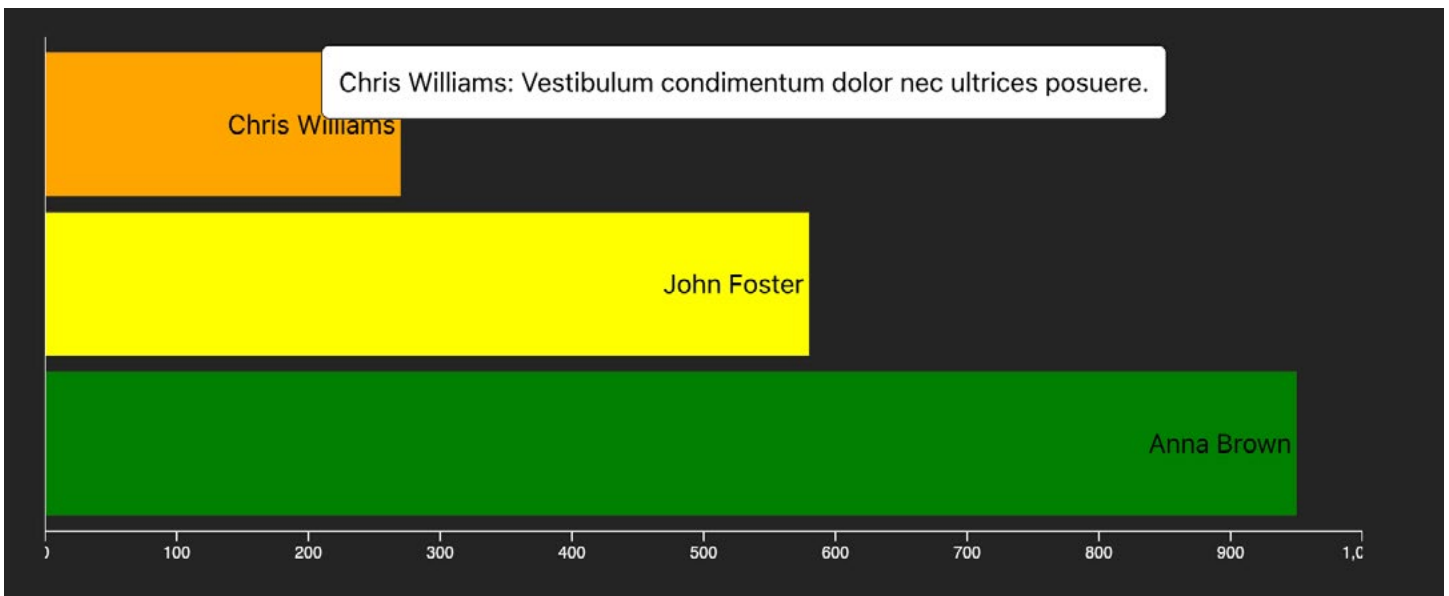


Let's update the implementation: `#file:Chart.tsx`

1. Add margin at the left, so the "Anna Brown" text is fully visible.
2. Add one more bar with for "John Foster", where performance equals 580 and "Chris Williams", where performance equals 270.
3. Sort the employees in ascending order.

make tooltip hoverable using plain d3 (no react-tooltip lib), with different lorem ipsum text depending of performance level
`#file: Chart.tsx`

What are we going to get from Copilot?



Here comes the code generated by Copilot. It is not ideal, so you need to carefully review it and update some ugly parts, such as TS `any` usages or nullish coalescing.

```
import React, { useEffect, useRef } from 'react';
import * as d3 from 'd3';

const data = [
  // Your data here
];

data.sort((a, b) => a.performance - b.performance);
// Sort the data in ascending order

function getColor(performance: number) {
  // Function to get color based on performance
}

function getTooltipText(performance: number) {
  // Function to get tooltip text based on performance
}

function BarChart() {
  const ref = useRef<any>();

  useEffect(() => {
    const svg = d3.select(ref.current as any)
      .attr('width', 800)
      .attr('height', 330)
      .style('margin-left', '100px');

    const xScale = d3.scaleLinear()
      .domain([0, 1000])
      .range([0, 800]);

    const yScale = d3.scaleBand()
      .domain(data.map(d => d.name))
      .range([0, 300])
      .padding(0.1);

    const xAxis = d3.axisBottom(xScale).ticks(10);
    const yAxis = d3.axisLeft(yScale);

    svg.append('g')
      .attr('transform', 'translate(0, 300)')
      .call(xAxis)
      .selectAll('text')
      .attr('fill', 'white');

    svg.append('g')
      .call(yAxis)
      .selectAll('text')
      .attr('fill', 'black');

    const tooltip = d3.select('body').append('div')
      .style('position', 'absolute')
      .style('color', 'black')
      .style('visibility', 'hidden')
      .style('background', 'white')
      .style('border', 'solid')
      .style('border-width', '1px')
      .style('border-radius', '5px')
      .style('padding', '10px');

    svg.selectAll('rect')
      .data(data)
      .enter()
      .append('rect')
      .attr('x', 0)
      .attr('y', d => yScale(d.name))
      .attr('height', yScale.bandwidth())
      .attr('width', d => xScale(d.performance))
      .attr('fill', d => getColor(d.performance))
      .on('mouseover', function (event, d) {
        tooltip
          .on('mousemove', function (event) {
            //
          })
          .on('mouseout', function () {
            //
          });
      })

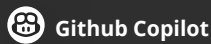
    svg.selectAll('text.bar')
      .data(data)
      .enter()
      .append('text')
      .attr('class', 'bar')
      .attr('x', d => xScale(d.performance) - 3)
      .attr('y', d => yScale(d.name) + yScale.bandwidth() / 2)
      .attr('dy', '.35em')
      .text(d => d.name)
      .attr('text-anchor', 'end')
      .attr('fill', 'black');

    return (
      <div>
        <svg ref={ref} />
      </div>
    );
  }, []);
}
```

Let's review the code once again and request Copilot to add some enhancements: color-code "zones" with interview scores and add some tweaks to the UI.



can you please add a few zones with different colors: if the performance is between 0 and 100 - Failed with red background, 100 and 175 - Associate level (pink background), 175 and 350 - Junior (yellow background), 350 and 550 - Middle (light green background), 550 and 675 - Strong Middle (green background), 675 and 800 - Senior (dark green background), 800 and 1000 - Principal (turquoise color).



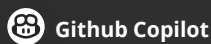
Used 1 reference >

Sure, you can add different zones with different colors by appending rectangles to the SVG before the bars. Here's how you can do it:

```
const zones = [
  { from: 0, to: 100, color: 'red', label: 'Failed' },
  { from: 100, to: 175, color: 'pink', label: 'Associate level' },
  { from: 175, to: 350, color: 'yellow', label: 'Junior' },
  { from: 350, to: 550, color: 'lightgreen', label: 'Middle' },
  { from: 550, to: 675, color: 'green', label: 'Strong Middle' },
  { from: 675, to: 800, color: 'darkgreen', label: 'Senior' },
  { from: 800, to: 1000, color: 'turquoise', label: 'Principal' },
];
```



Also, add a vertical dividers with gray dashes lines, corresponding to the borders of each zone



Used 1 reference >

To add vertical dividers at the borders of each zone, you can append lines to the SVG. Here's how you can do it:

```
useLayoutEffect(() => {
  const svg = d3.select(ref.current as any)

  // Add zones
  zones.forEach(zone => {
    svg.append('rect')
      .attr('x', xScale(zone.from))
      .attr('y', 0)
```

Ask Copilot a question or type '/' for commands



After adjusting the colors to match the design specifications, fonts, and alignment, we finally achieve the intended result.

It's amazing! We have fully-functional bar charts displaying tooltips with additional insights on candidates. While this example could be further elaborated, it clearly illustrates that Copilot can enhance the productivity of a software developer. Overall, the level of support you can get from Copilot marks a significant milestone in the software development industry. You always have an assistant at your disposal ready to help you with any task. Experimenting with D3 without any previous experience could involve weeks spent trying to grasp how it works, let alone implementing anything meaningful.

Now, you need an additional 30 minutes to finalize the feature by asking Copilot the right questions and providing clear instructions on what needs to be done.

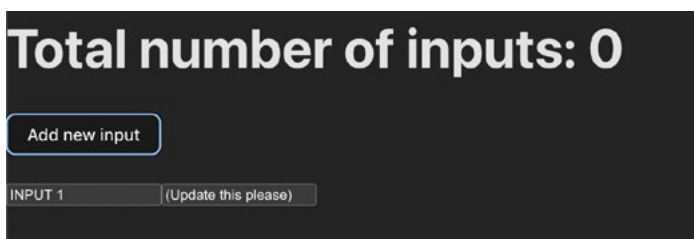
Code review and quality assurance

When preparing a big pull request introducing a new feature, software developers review their code to make sure it is free from extraneous development artifacts such as console logs, unused imports or variables, and unrefactored code segments.

As code review and quality assurance practices are tailored to each programming language and framework, let's explore how well Copilot can help with self-review of a component written in React.

Having taken a closer look at the code below, we spot a couple of problems:

- ▶ Extremely poor typing. It's missing altogether.
- ▶ Violations of hook rules, as hooks should not be conditionally invoked. Dependencies array is empty for `useEffect`.
- ▶ Key for mapped items is missing.



```
import React, { useEffect, useState } from "react";

const INITIAL_TEXT = '(Update this please)';

export const BadComponent: React.FC = (props) => {
  const [items, setItems] = useState([]);
  const [itemsCounter, setItemsCounter] = useState(0);

  if (props.hidden) {
    return null;
  }

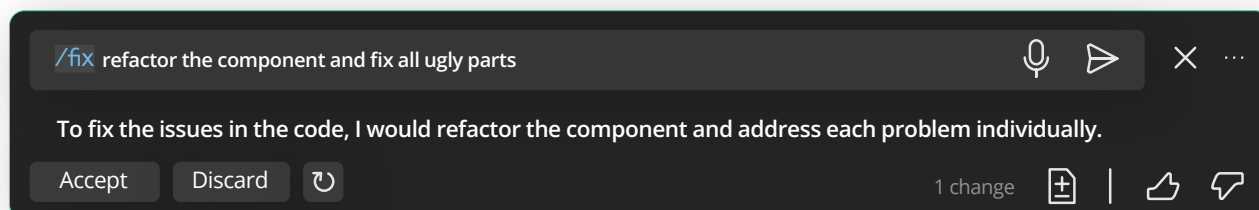
  useEffect(() => {
    setItemsCounter(items.length);
  }, []);

  const onAddButtonClick = () => {
    setItems([...items, {id: items.length ? items[items.length - 1].id + 1 : 0, name: INITIAL_TEXT }]);
  };

  const onUpdate = (updatedItem: any) => (e: any) => {
    setItems(items.map((item) => (updatedItem.id === item.id ? { ...item, name: e.target.value } : item)));
  };

  return (
    <>
      <h1>Total number of inputs: {itemsCounter}</h1>
      <button onClick={onAddButtonClick}>Add new input</button>
      <br />
      <br />
      {items.map((item) => {
        return <input value={item.name} onChange={onUpdate(item)} />;
      })}
    </>
  );
};
```

Having identified these issues before creating a merge request, let's see what improvements Copilot suggests while refactoring this component:



/fix refactor the component and fix all ugly parts



GitHub Copilot

To fix the issues in the code, I would refactor the component and address each problem individually.

Accept

Discard



Accept or Discard 1 change.

```
import React, { useEffect, useState } from 'react';

const INITIAL_TEXT = '(Update this please)';

interface Props {
  hidden?: boolean;
}

interface Item {
  id: number;
  name: string;
}

export const BadComponent: React.FC<Props> = ({ hidden }) => {
  const [items, setItems] = useState<Item[]>([]);
  const [itemsCounter, setItemsCounter] = useState<number>(0);

  useEffect(() => {
    setItemsCounter(items.length);
  }, [items.length]);

  const onAddButtonClick = () => {
    setItems((prevItems) => [
      ...prevItems,
      { id: prevItems.length ? prevItems[prevItems.length - 1].id + 1 : 0, name: INITIAL_TEXT }
    ]);
  };

  const onUpdate = (updatedItem: Item) => (e: React.ChangeEvent<HTMLInputElement>) => {
    setItems((prevItems) =>
      prevItems.map((item) =>
        updatedItem.id === item.id ? { ...item, name: e.target.value } : item
      )
    );
  };

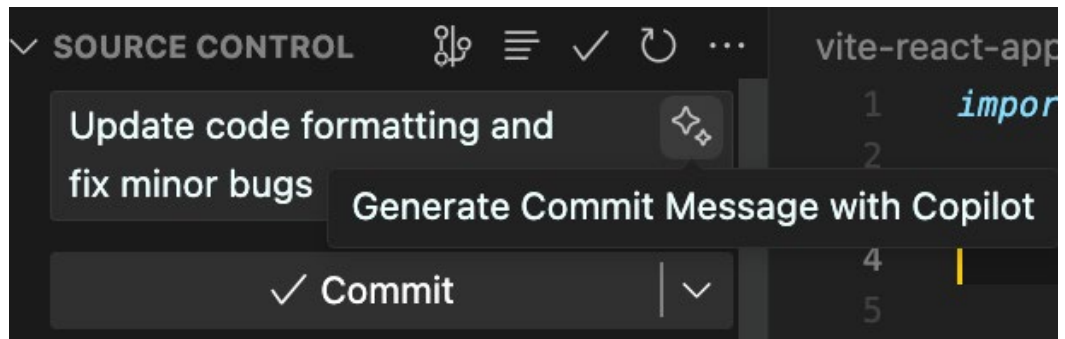
  if (hidden) {
    return null;
  }

  return (
    <>
      <h1>Total number of inputs: {itemsCounter}</h1>
      <button onClick={onAddButtonClick}>Add new input</button>
      <br /><br />
      {items.map((item) => (
        <input key={item.id} value={item.name} onChange={onUpdate(item)} />
      ))}
    </>
  );
};
```

As we can see, Copilot managed to identify and fix all the mentioned issues. So, a “self-review” for any code smells works well in practice and should be done every time you are ready to create a pull request. On top of “self-reviews”, Copilot has something more to offer you and your team: within the context of Git flow, Copilot extends its functionalities to the following:

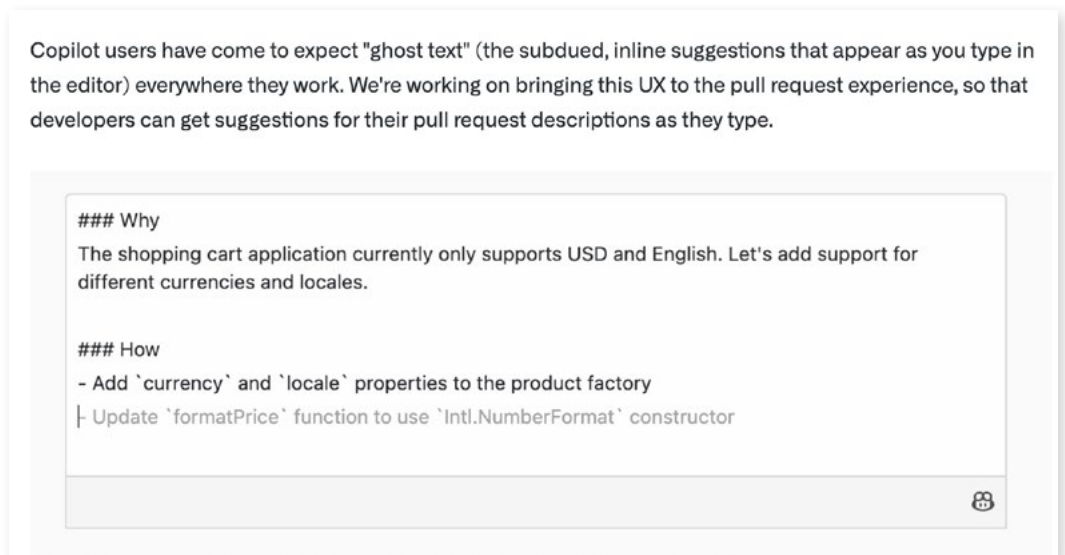
1

- Writing commit messages, accurately describing the changes made.



2

- Following a recent update on GitHub, Copilot Enterprise now supports pull requests, offering comprehensive support and analytics. It furnishes various features for describing pull request changes, providing insights into the code, and highlighting potential issues.



3

- Copilot can understand your project's rules and best practices, suggesting solutions that adhere to your project's conventions and standards.

Knowledge sharing and documentation

For large projects with a huge code base, it is crucial to have rich documentation and onboarding capabilities for new engineers joining the team. Even here Copilot can assist you and your teams with the following features.

Auto-generated documentation

One of the most significant advantages of using an AI copilot is its ability to automatically generate documentation based on the code base. This includes not only comments within the code but also external documentation such as API guides, usage examples, and technical specifications. By analyzing the structure and functionality of the code, AI copilots can produce accurate and detailed descriptions, saving developers considerable time and effort. This auto-generated documentation can then be reviewed and refined by the team, ensuring clarity and completeness.

Let's take a look at the code below. It handles API requests and shows different loaders depending on the request type (MobX example):

```
class ApiLoader {
  _loadingRecordIds: LoadingRecord[] = [];

  @computed
  get loadingInfo(): {
    isLoading: boolean;
    activeRequestTypes: Set<LoadingRecordType>;
  } {
    return {
      isLoading: this._loadingRecordIds.length > 0,
      activeRequestTypes: new Set(
        this._loadingRecordIds.map((item) => item.type),
      ),
    };
  }

  addLoadingRequest(key: string, type: LoadingRecordType): string {
    const id = `${key}__${new Date().getTime()}`;
    this._loadingRecordIds = [...this._loadingRecordIds, { id, type }];
    return id;
  }

  removeRequest(id: string) {
    this._loadingRecordIds = this._loadingRecordIds.filter(
      (item) => item.id !== id,
    );
    return this._loadingRecordIds;
  }

  constructor() {
    makeAutoObservable(this);
  }
}
```

You can just use a Copilot prompt and select /doc command to document the selected code:



```
/**
 * Represents a class that manages loading requests and provides loading information.
 */
class ApiLoader {
  _loadingRecordIds: LoadingRecord[] = [];

  @computed
  get loadingInfo(): {
    isLoading: boolean;
    activeRequestTypes: Set<LoadingRecordType>;
  } {
    return {
      isLoading: this._loadingRecordIds.length > 0,
      activeRequestTypes: new Set(
        this._loadingRecordIds.map((item) => item.type),
      ),
    };
  }

  /**
   * Adds a loading request to the loader.
   * @param key - The key associated with the loading request.
   * @param type - The type of the loading request.
   * @returns The unique identifier of the loading request.
   */
  addLoadingRequest(key: string, type: LoadingRecordType): string {
    const id = `${key}__${new Date().getTime()}`;
    this._loadingRecordIds = [...this._loadingRecordIds, { id, type }];
    return id;
  }

  /**
   * Removes a loading request from the loader.
   * @param id - The unique identifier of the loading request to remove.
   * @returns The updated array of loading requests.
   */
  removeRequest(id: string) {
    this._loadingRecordIds = this._loadingRecordIds.filter(
      (item) => item.id !== id,
    );
    return this._loadingRecordIds;
  }

  constructor() {
    makeAutoObservable(this);
  }
}

export default new ApiLoader();
```

Real-time documentation updates

As projects evolve, keeping documentation aligned with the latest changes can be quite a challenge. In this arena, AI copilots shine by providing real-time updates to documentation as code changes. This dynamic approach ensures that documentation reflects the current state of the project, reducing discrepancies and misunderstandings. Getting real-time documentation updates also facilitates the onboarding process, as newcomers can substantially reduce the learning curve and become high-performing contributors much sooner.

In-line documentation of code

AI copilots can enrich the code with in-line documentation, offering explanations and insights directly within the code base. This in-line commentary can clarify complex algorithms, explain the purpose of specific code segments, and highlight the potential impacts of future changes. Such contextual information enhances understanding and facilitates more effective collaboration and maintenance.

Knowledge base development

In addition to project-specific documentation, AI copilots can contribute to the development of a knowledge base. By aggregating and organizing information sourced from various projects, AI tools can help create a centralized repository encompassing best practices, common patterns, and solutions to recurring issues. This resource proves invaluable for developers seeking guidance or inspiration, promoting a culture of knowledge-sharing and continuous learning.

Interactive documentation and tutorials

AI copilots can also create interactive documentation and tutorials, offering a more engaging learning experience. These resources include code snippets, interactive examples, and step-by-step guides tailored to the developers' current tasks and projects. By providing a hands-on learning environment, AI helps developers delve deeper into concepts and apply them more effectively in their work.

Key takeaways

- ▶ GitHub Copilot significantly **reduces the time and effort** required to explore new technologies and develop features by providing immediate code suggestions.
- ▶ Copilot assists in **refining initial code** implementations by generating improvements based on specific requirements, such as adding axes to charts or adjusting design elements, ensuring the final output meets the desired standards.
- ▶ Copilot can **identify and correct common code issues**, such as poor typing, hook violations, and missing keys in mapped items, enabling developers to perform thorough self-reviews before creating pull requests.
- ▶ AI copilots facilitate the **creation and maintenance of up-to-date documentation**, including in-line comments, API guides, and technical specifications, thereby enhancing knowledge sharing and onboarding for new team members.

In the next chapter, let's delve into how Intellias mitigates privacy and security risks while building AI-powered products.

05

Secure use of AI



“

Security is always excessive until it's not enough.”

– Robbie Sinclair

Privacy and security risks

In the world of software development, where innovation drives progress, the use of AI poses some unique security risks. All interactions, such as entering or copying data from an AI system, create data that becomes an integral part of the system, 'enriching' subsequent responses with this new knowledge. Therefore, the most prominent risks associated with the use of AI in software development involves personal, sensitive, confidential, and corporate data, as well as intellectual property.

Let's explore each risk in more detail:

- ▶ **Data leakage** is one of the most significant risks for corporate use of AI. AI tools are continuously collecting and storing data to teach their models, which means confidential information, including Intellectual property, may be leaked by a trained model output.
- ▶ **Privacy** is another big concern related to personal and sensitive data, which may be used without appropriate consent. The risk of this data being leaked may have a significant impact on a company and its employees, sub-contractors, and contractors.
- ▶ **AI outputs** may often contain factual errors., which should be anticipated. For example, AI may suggest insecure code if your existing code base contains security vulnerabilities. Also, both algorithm design and the choice of training datasets for AI models can cause inadvertent bias due to unrepresentable training datasets. AI tools imitate learned patterns or use available context without providing judgment.
- ▶ AI tools may reuse **open-source code**. Pulled from restrictive license code bases, AI-generated code can introduce Public Software (OSS) and Free Public Software (FOOS) risks to Intellias' or our clients' code repositories.
- ▶ **Regulatory compliance risk:** The USA and EU are drafting **AI legislation**, which may restrict or limit the use of AI in some areas.

Disclaimer

In your security practices, never rely solely on AI. AI can assist, but human oversight is essential to ensure comprehensive protection.

AI risks mitigation

Understanding the risks and possible consequences, our goal is not only to harness the power of AI tools but also to mitigate the associated risks. Let's delve into some essential strategies on how to mitigate risks when using AI tools in a corporate environment.

- ▶ **Avoid the exposure of sensitive corporate data to public AI platforms.** This includes confidential documents, client information, source code, and non-anonymized user and test data. Opt for secured proprietary AI solutions such as GitHub Copilot Business or proprietary versions of ChatGPT.
- ▶ **Use secured proprietary AI tools when writing code with AI.** In proprietary AI tools, prompts used to generate a suggestion are not retained, including the code and other context. Prompts are discarded once a suggestion is returned and are not used for training models, which provides an added layer of security.
- ▶ **Thoroughly review code generated by AI tools.** This review should include verification of architectural patterns and a security assessment assessment to identify and rectify potential vulnerabilities.
- ▶ **Develop clean code and adhere to security coding guidelines for the development team.** We recognize that generative AI coding assistants lack an understanding of code semantics and solely mimic patterns from their training data. Thus, it is paramount to provide clear role models and adhere to established guidelines.
- ▶ **Regularly review the licensing terms** of libraries, frameworks, and components used to generate code, as well as the licensing terms of AI tools and models.
- ▶ **Get approval from the client before incorporating any open-source software** (OSS) or free and open-source software (FOSS) into project environments to mitigate potential legal and compliance risks.
- ▶ **Introduce corporate policies on the appropriate use of AI tools.** These policies should highlight key risks and mitigation strategies specific to the organization's context, fostering a culture of responsible and secure AI use.

EU Artificial Intelligence Act (AIA)

Following in the footsteps of the General Data Protection Regulation (GDPR), a new legislation on artificial intelligence – the EU Artificial Intelligence Act (AIA) – will be introduced in Europe. The implementation phase, estimated to span two to three years contingent upon progress in EU institutional negotiations, suggests that the EU Artificial Intelligence Act may be enacted in 2026 or beyond.

This legislation targets companies that develop AI solutions, mandating compliance with rules structured around a four-tiered risk framework:

Risk tier	System types
Unacceptable risk	Systems engaging in prohibited practices such as government-led social scoring, as observed in China, biometric categorization and instant remote recognition based on sensitive characteristics, subliminal persuasion to influence political preferences without consent, exploiting vulnerabilities that lead to harmful actions, evaluation of an individual's emotional state, indiscriminate collection of facial images for database expansion from the internet or video surveillance recordings.
High risk	Systems mainly used in critical sectors such as healthcare, transportation, and law enforcement. These systems are assessed for accuracy and protection from cyber threats to reduce any associated risks.
Limited risk	Less risky systems than the high-risk ones, so they don't have as many rules to follow. But, even though they do not need to be assessed as closely, they still have to be clear about how they work to keep them accountable and trustworthy.
Minimal risk	Systems where stakeholders, whether based within or outside the EU, are urged to develop codes of conduct.



Development of an AI solution compliant with the EU AI Act will require a thorough distribution of responsibilities among specialized roles. For example, the Compliance team should assess the system against the EU AI Act Compliance Checker, while the Data Science team should verify the implementation. Furthermore, the development environment should be strengthened with tools to assess crucial aspects such as ethics and privacy, transparency and bias, fairness, AI explainability, and more.

Feel free to contact the Intellias Sales Department for a demonstration, licensing information, and potential discounts to experience firsthand how Compl-AI can transform your regulatory compliance challenge.

Key takeaways

- ▶ The use of AI tools in software development **introduces significant risks of data leakage**, including the exposure of confidential and sensitive information, potentially impacting companies and individuals.
- ▶ AI-generated outputs can **contain inaccuracies and biases**, which may lead to the suggestion of insecure code or propagate biases present in training datasets. Human oversight is essential to mitigate these risks.
- ▶ AI tools may inadvertently introduce open-source software and free and open-source software risks due to the reuse of restrictive licensed code, necessitating thorough **review and client approval**.
- ▶ Effective risk mitigation includes **avoiding exposure of sensitive data to public AI platforms**, using secured proprietary AI tools, adhering to security coding guidelines, regularly reviewing licensing terms, and establishing corporate policies for AI use.
- ▶ The upcoming **EU Artificial Intelligence Act will impose stringent compliance requirements** on AI systems, particularly those in high-risk sectors. Organizations must prepare for these regulations by implementing comprehensive compliance and review processes.

In the next chapter, we will discuss two key ways of using AI, highlight the importance of building trust in AI by design, and address biases in AI models.



06

Building AI products from business and user perspectives

“

*Trust is the glue of life. It's the
foundational principle that holds all
relationships.”*

– Stephen Covey

Having explored various aspects of the development and capabilities of AI Copilot, as well as the potential security risks, we are now halfway through our book. However, this is only part of what product owners, business analysts, and the development team need to consider. In this chapter, we will discuss various issues and biases that your product may encounter in its final stages if they are not addressed early in the development and design process.

AI product development inherently carries higher risks compared to traditional, non-AI technology products. These risks extend beyond concerns about data privacy, spanning critical issues such as **algorithmic transparency** and **fostering trust between the product and its users**. Let's explore these facets in depth to understand the unique challenges posed by AI technologies.

To begin, the handling of private data by AI systems raises significant privacy concerns. AI's reliance on vast datasets for training means that sensitive information is often at risk of exposure or misuse, posing profound implications for users' privacy.

Moreover, the opacity of many AI algorithms—often referred to as “black boxes”—complicates matters further. The lack of transparency and potential algorithmic biases can obscure the decision-making process, making it challenging for users and regulators to understand how decisions are made. This opacity poses a significant challenge in establishing trust, which is a cornerstone of the user-product relationship.

Building trust in AI systems is a multifaceted challenge. It entails ensuring privacy and transparency while demonstrating reliability, fairness, and accountability in the system's operations. Users need to feel confident that AI products will consistently act in their best interests, without introducing unexpected biases or errors.

Addressing these risks is complex and requires a concerted effort from developers, regulators, and users alike. Effective navigation of these challenges involves fostering open dialogue, implementing ethical guidelines, and pursuing technological advancements that enhance transparency and trust. This section delves into these issues, shedding light on the intricate balance between innovation and responsibility in developing AI products from both business and user standpoints.

Two main ways of using AI

The rapid evolution of AI is reshaping the landscape of work, introducing two major shifts: automation and augmentation. Automation involves AI performing repetitive tasks instead of people. On the other hand, augmentation entails AI helping people perform their jobs more effectively. Let's examine how these changes are unfolding, their implications for jobs, and the potential future outcomes.

AI excels at learning routine tasks that follow predictable patterns. It thrives in roles where consistency and speed are paramount, such as sorting through large volumes of data or controlling machinery in manufacturing settings. AI's proficiency in recognizing patterns improves with exposure to more examples, enhancing its capabilities over time.

In certain business scenarios, AI replaces human labor. The aim is to execute tasks faster and more accurately, resulting in cost savings and improved efficiency. Examples include autonomous vehicles, customer service chat bots, and automated financial analysis tools. Here, AI strives to outperform humans, particularly in tasks involving rapid information processing or decision-making.

However, **some tasks require AI to comprehend complex situations or solve problems like humans.** This demands a deeper understanding, beyond mere pattern recognition. For example, understanding nuances in language, humor, or cultural references, as well as adapting to dynamic environments, presents formidable challenges. Achieving human-like cognition remains an ongoing pursuit for AI researchers.

In these scenarios, AI and human capabilities complement each other, working on common goals as one team. AI may serve as a valuable tool to augment people. Rather than replacing us, AI assists in performing tasks more effectively. This includes aiding doctors in medical diagnoses, suggesting content for writers, or analyzing large datasets to facilitate decision-making. Such collaboration aims to tackle challenges that may be too complex or time-consuming for us alone.

Automation reduces dependency on human labor

Automation, powered by AI, eliminates the need for a person to step in. This includes both simple tasks and more complex ones, such as reading documents or recognizing objects in pictures. Let's explore how automation is changing various industries, expediting certain tasks and reducing the reliance on human labor.

Our exploration navigates the advantages and intricacies inherent in this shift, particularly its implications for employment. AI broadens the horizon of automation possibilities, spanning tasks that range from computer vision to natural language processing.

A McKinsey study examining 800 different occupations revealed that 60% of them entail activities with a potential automation rate exceeding 30%. These activities encompass both physical tasks within structured environments and tasks revolving around data collection and processing.

Collaboration of AI and people through augmentation

Augmentation, in contrast to automation, entails the collaboration of AI and people. It's like having a smart assistant who can analyze data or propose creative solutions at lightning speed. Rather than displacing jobs, augmentation aims to enhance them, rendering tasks more manageable and effective.

The benefits of human-AI collaboration include the combination of complementary skill sets and maintaining human oversight of the process. This collaboration manifests in numerous applications across various domains, showcasing its versatility and impact. Let's walk through some notable examples that underscore AI's potential to augment human capabilities across various sectors:

Triage



Radiology



Insurance underwriting

Decision support



Medical diagnosis



Investing

- ▶ **Triage in healthcare settings.** AI helps medical professionals prioritize patient care based on severity and urgency.
- ▶ **Radiology.** AI assists in the analysis of medical images to detect abnormalities and make diagnosis.
- ▶ **Insurance underwriting.** AI algorithms help insurance companies assess risks and determine appropriate coverage for clients.
- ▶ **Decision-making.** AI provides insights and recommendations to support strategic and tactical decision-making processes.
- ▶ **Medical diagnosis.** AI assists healthcare providers in diagnosing diseases and conditions based on patient data and symptoms.
- ▶ **Investment.** AI algorithms analyze financial data and market trends to facilitate investment decisions and optimize portfolio performance.

Trust in AI by design

As AI becomes a part of our lives, making sure it's trustworthy is super important. Trust by design means making AI in the way that people feel safe and confident using it. It's about being clear on how AI makes decisions and making sure it treats everyone fairly. Let's explore how to build AI that people can trust, focusing on being open and fair.

Building trust within your model

In the pursuit of trust, it's essential to acknowledge the inevitability of AI model imperfections. By building transparency and clear communication, users can trust AI systems, despite their flaws. Here's how to foster trust and confidence in your AI model:

- ▶ **Acknowledge that your model will make mistakes.** No AI model is perfect, and it's important to be upfront about this. By setting realistic expectations, users are less likely to be disillusioned when errors occur.
- ▶ **Communicate its performance clearly.** Be open about how well your model performs. Share statistics or success rates that highlight its reliability in understandable terms. This helps users gauge when they can rely on it and for what purposes.
- ▶ **Show how confident it is in its own decisions.** Whenever your model makes a prediction or a decision, also provide a confidence score. This tells users how sure the AI is about its output, helping them make informed decisions on whether to follow the AI's advice.
- ▶ **Explain how it comes to its conclusions.** People trust what they understand. If possible, offer simple explanations for the model's decisions. This could be a breakdown of the factors that influenced a decision or a straightforward explanation of the steps it took.
- ▶ **Be honest about its limitations.** Every model has its weak points. By openly discussing these, you encourage users to use the AI where it's strongest and avoid areas where it might not perform well.
- ▶ **Keep people in the loop.** For critical decisions or when the AI's confidence is low, having a human review process can boost trust. This shows users that there's a safety net to catch errors the AI might make.

Communicating AI performance metrics

In understanding the impact of AI models, it's crucial to grasp both the overarching objectives and the specific outputs they generate. Let's explore how outcome and output metrics play into the evaluation of AI performance.

Outcome metrics encompass the broader goals driving the use of AI within your business or project. Think of them as the ultimate objectives that define success, whether it's increasing sales, cutting costs, or enhancing customer satisfaction. These metrics encapsulate the overarching benefits AI is expected to deliver, often directly tied to financial outcomes.

On the other hand, **output metrics** provide a granular assessment of the AI model's performance in executing its designated tasks. Similar to a report card, these metrics offer insights into how effectively the AI is accomplishing the specific objectives set for it. For instance, if the AI's function involves image recognition, output metrics might quantify its accuracy in identifying images correctly.

Providing explanations

When the model is off, transparency into model output helps users understand why it differs from reality. Whenever your model makes a prediction or decision, it's helpful to provide a brief explanation that sheds light on why it made that choice. This can be something like identifying the key factors or data points that had the biggest impact on the decision. For example, if your AI recommends a specific type of shoe to a customer, it could say it made that recommendation because the customer often buys that brand or has searched for similar styles recently.

Along with the explanation, show how confident the model is in its decision by using a percentage. But don't stop there—also share what other options were considered and how confident the model was in those options as well. For example, if the AI is 80% confident in its first choice, it might also show that there was a second choice it was 60% confident in. This gives users a sense of the model's certainty and allows them to see alternatives it considered but deemed less likely.

Acknowledging limitations

Admit when the model doesn't have the answer. It's really important to know that AI isn't perfect. There will be times when your AI model can't make a reliable prediction or decision. In those cases, it's better to be honest and say the model doesn't know, rather than guessing and possibly getting it wrong.

Instead of taking a wild guess and risking a mistake, it's sometimes wiser not to give an answer at all. This way, you avoid giving users incorrect information, which could lead to confusion or mistakes based on bad info.

When the AI can't provide an answer, it doesn't mean you leave the user with nothing. You can suggest other ways for them to get the information or help they need. This could be directing them to a human expert, recommending they check out certain resources, or using another tool better suited for their specific question.

In simple terms, if your AI doesn't know the answer, it's best to say so. Then, try to be helpful by pointing the user towards someone or something that can help. This honesty not only builds trust but also ensures that users are not misled by inaccurate information.

Human-in-the-loop

Human-in-the-loop is a key concept in AI development, involving people in the validation of AI-generated outcomes. It's particularly crucial during the initial stages of AI implementation when the model is honing its capabilities and prone to mistakes.

We should think of people checking AI's work like having a teacher look over a student's homework. After your AI model makes a decision or gives an answer, a person checks to make sure it's correct. This helps catch any mistakes before they cause problems for users. If a person spots a mistake, it's an opportunity to either correct the error directly or find another way to help the user. For example, if the AI can't answer a question correctly, the person checking can provide the right answer or guide the user to where they can find what they need.

It's important not to rely on guesswork to decide if the AI is right or wrong. The person checking the AI's work should use clear guidelines and knowledge. This way, the process is systematic and doesn't add confusion by guessing.

Accountability

Accountability is key for ensuring responsible and ethical use of AI. It entails clarity around who oversees the AI's actions, mechanisms for users to address issues, and adherence to established rules and values.

Just like someone is responsible for making sure a car is safe to drive, there should also be clear responsibility for the AI's actions and decisions. Knowing who's in charge means there's a person or a team you can point to **who makes sure the AI works right and does what it's supposed to do**. If someone using the AI finds a mistake or an issue, they should have a way to report it and get it fixed. It's like knowing there's a customer service line or a help desk you can call when something's not working.

Let's explore the top three considerations for accountability in AI. First of all, it should be clear who is responsible for making sure the AI does its job well and ethically. Is it the people who made it, the ones who use it, or someone else?

Another consideration is **what rules and values guide the AI**. Every AI system should follow certain rules and values, which could include fairness, privacy, and not causing harm. These should be based on the laws of the place it's used and the moral standards expected by society.

And last but not least, accountability in AI **involves providing users with a course of action if the AI isn't acting appropriately**. If the AI makes a mistake or acts against its set rules and values, users need to have a way to fix the situation. This could mean correcting the mistake, reporting the issue to the authorities, or even getting compensation for any harm caused.

By addressing these key considerations, we ensure that AI systems operate ethically, guided by principles of fairness, privacy, and harm prevention, and that users have avenues for recourse when issues arise.

User onboarding

User onboarding is a critical phase in the implementation of any new system, especially when it involves AI. **Proactive user education** plays a key role in successful change management, ensuring that users not only understand the value proposition of the AI system but also embrace its integration into their workflows.

By highlighting the **"what's in it for me"** aspect, users are more likely to perceive the AI system as a valuable asset that will enhance their daily lives rather than a disruptive force. Additionally, anticipating and addressing concerns or fears that users may have about the new

technology is crucial for acceptance and reducing resistance during the onboarding process.

Through **clear communication** of the value that the AI system will create and thoughtful consideration of user needs, organizations can facilitate a smooth transition to AI-powered solutions and maximize their impact.

Transparency

Transparency in the use of data and AI model functioning is key for ensuring user trust and confidence. When users have visibility into how their data is being used and understand the inner workings of the models involved, they feel more reassured in their interactions with the system. Let's delve into some effective strategies for achieving transparency in data-driven processes:

- ▶ **Using models that users can easily grasp**, like decision trees or linear models, helps them understand how data is used and decisions are made.
- ▶ **Breaking down complex algorithms** into simpler terms helps users understand without getting lost in technical details.
- ▶ **Showing which input factors** are most important in model decisions helps users see what influences outcomes.
- ▶ **Describing how changes in input data affect model predictions** helps users understand how the model works in different situations.

**Interpretable
models**

**Counterfactual
explanations**

**Feature
importance**

**Simplified
approximations**

Bias in AI models

Bias in AI is a big problem. It can lead to unfair decisions or make some people worse off. We need to be really careful about how AI is trained and used to avoid these issues.

Let's explore where these biases come from, how they can affect us, and what we can do to make AI fairer for everyone. Bypassing biases will help you create a better AI product, solve the business challenges of early product deployment, and reduce the associated risks.

Historical bias

The data an AI collects reflects the biases that exist in society at the time the data was collected. For example, facial recognition technologies trained on datasets predominantly composed of lighter-skinned individuals tend to misidentify or fail to recognize people with darker skin tones, illustrating the bias present in the dataset's composition.



Algorithmic bias

AI systems are widely regarded as objective tools. However, they can inadvertently amplify human biases, leading to unfair outcomes. These biases can manifest in various ways, often impacting marginalized groups disproportionately. Here are the main sources through which biases can impact AI systems:

- ▶ The personal **beliefs and biases of the developers** can unintentionally influence the design and function of AI systems. This might occur during the coding process or through subjective decisions about which features of the data are most important.
- ▶ This bias may occur depending on how **data is collected, selected, or used** to train AI. If the data reflects historical inequalities or lacks representation from certain groups, the AI will likely replicate these issues.
- ▶ Sometimes, AI systems are **deployed in contexts that were not anticipated** by the developers, leading to unexpected biases. This can happen when an AI system designed for one purpose is used for another without considering the different requirements or impacts on various user groups.



Understanding these sources of bias is crucial for developing more equitable AI technologies. Addressing algorithmic bias involves ongoing efforts from developers, regulators, and users to ensure AI systems perform reasonably across all segments of society.

Representation bias

This issue arises when the data used to train an AI system doesn't fully capture the diversity of the group it's meant to serve. This often leads to certain segments of the population being under-represented.

For example, if a speech recognition system is mostly trained on voice data from native English speakers, it might struggle to accurately recognize accents or dialects from non-native speakers.

Another example is a facial recognition system trained predominantly with images of men, which may then perform less accurately in identifying women, particularly if it has not been exposed to a balanced gender dataset during its training phase.

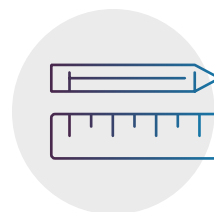


Measurement bias

Measurement bias refers to inaccuracies that occur when the metrics or indicators chosen to represent a concept do not effectively capture it, or when these measurements vary inconsistently across different groups.

This bias can lead to misleading conclusions. An example of this would be using body mass index as a proxy for an individual's health or fitness level. Body mass index might not accurately reflect a person's overall health because it doesn't distinguish between muscle and fat mass, nor does it account for distribution of fat or muscle. As a result, athletes or individuals with high muscle mass might be incorrectly classified as overweight or obese.

Another example could be the use of arrest records as a measure of criminal activity in different neighborhoods. This can be biased if police patrols or enforcement intensity varies by area, leading to disproportionate crime rates in more heavily patrolled neighborhoods, rather than an accurate reflection of actual crime rates across all areas.

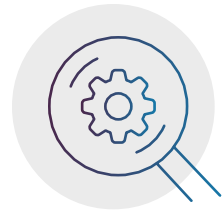


Learning bias

Learning bias arises when the methodology and choices made in the development of a model exacerbate performance discrepancies among different demographic groups. This is often a result of a model's cost function optimizing for overall performance without ensuring fairness and consistency across these groups, leading to unequal outcomes.

For example, a loan approval algorithm may disproportionately favor applicants from certain socioeconomic backgrounds over others, not because of their actual creditworthiness, but due to the data and parameters emphasized in the model.

Another example could be in language translation models. They may perform well on widely spoken languages but poorly on dialects or languages with fewer speakers. This occurs because the model was optimized for the majority, thus sidelining the linguistic nuances and accuracy needed for these less represented languages.



Deployment bias

Deployment bias occurs when there is a gap between the intended and actual use of a technology, often because its creators did not fully account for the real-world context in which it would operate.

An example of this is a facial recognition system designed for security purposes, intended to identify persons of interest in crowded public spaces. However, if this system is deployed in a manner that subjects certain demographic groups to increased surveillance and scrutiny, its use deviates significantly from its original security-enhancing purpose.

This shift not only raises ethical concerns but also demonstrates how technology can be repurposed in ways that magnify biases and inequalities, highlighting the consequences of not considering the broader implications of technology deployment.



Feedback loop bias

Feedback loop bias happens when the system's setup causes it to influence its own training data, leading to skewed model outputs over time.

An example of this is in predictive policing, where law enforcement uses algorithms to predict crime hot spots. If a system directs more police patrols to areas it predicts as high-risk, and the increased police presence leads to more reported crimes in those areas. This occurs

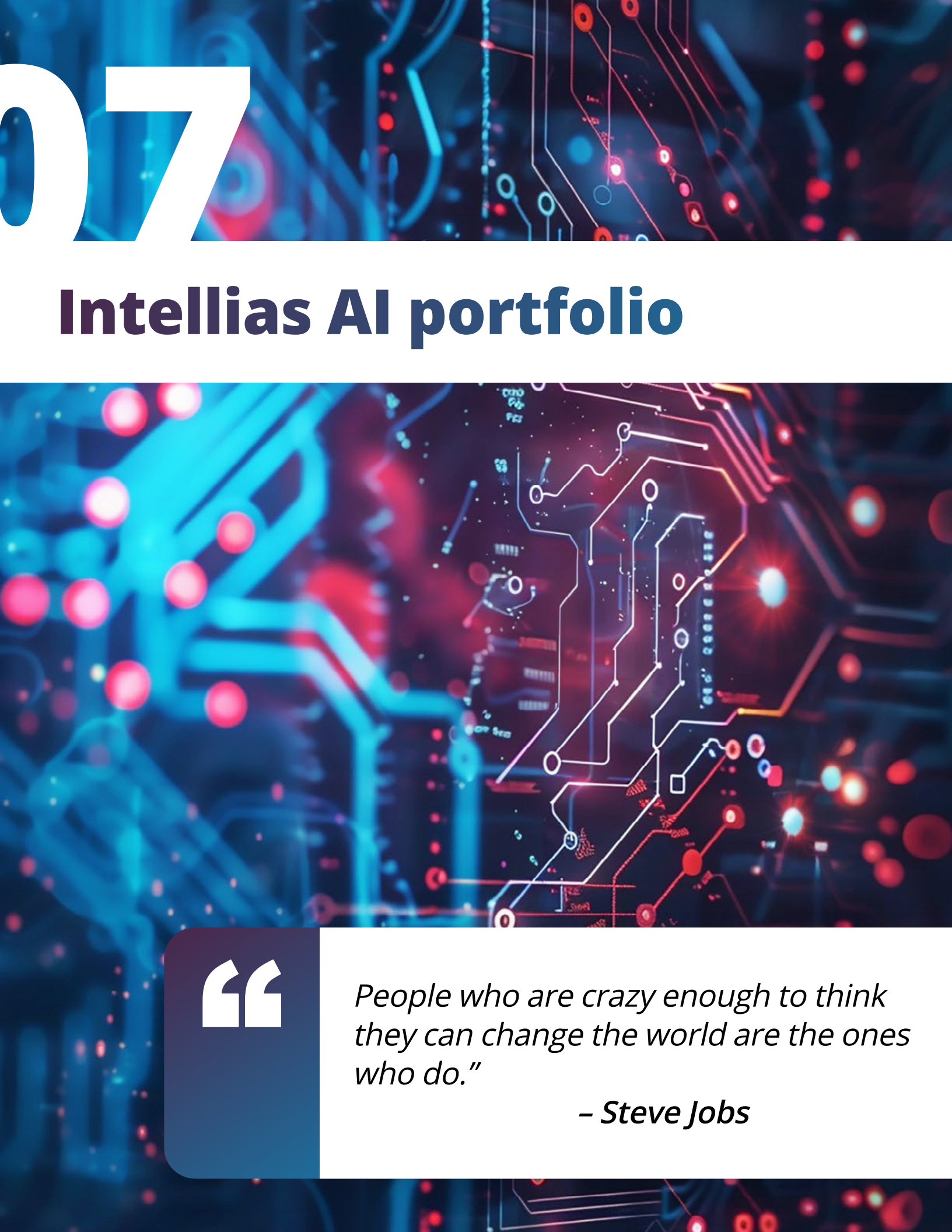


simply due to higher surveillance (rather than an actual increase in crime rate), as the system receives data that confirms its predictions. This, in turn, causes the algorithm to further highlight these areas as high-risk, creating a self-reinforcing cycle. The result is a disproportionate focus on certain neighborhoods, amplifying initial biases and potentially neglecting other areas that might also need attention.

Key takeaways

- ▶ AI product development requires careful handling of private data and addressing the opacity of AI algorithms to **build user trust**. Ensuring algorithmic transparency and demonstrating how decisions are made are crucial for fostering confidence in AI systems.
- ▶ AI reshapes work by **automating routine tasks** and **augmenting human capabilities**. While automation reduces reliance on human labor, augmentation enhances job performance by combining AI's data analysis and pattern recognition with human creativity and decision-making.
- ▶ Establishing trust involves **acknowledging AI's imperfections**, communicating performance metrics clearly, explaining decision-making processes, and being transparent about limitations. This helps users understand and have confidence in AI's capabilities and reliability.
- ▶ **Identifying and mitigating biases in AI models** is essential to avoid unfair outcomes. This includes recognizing historical, algorithmic, representation, measurement, learning, deployment, and feedback loop biases to ensure equitable performance across all user groups.
- ▶ **Incorporating human oversight in AI processes** ensures accuracy and ethical use. Clear accountability structures, ethical guidelines, and mechanisms for addressing issues are vital for responsible AI deployment and maintaining user trust.

Let's explore our expertise in building an AI portfolio that addresses the diverse business needs of mid-sized companies and enterprises.



07

Intellias AI portfolio

“

People who are crazy enough to think they can change the world are the ones who do.”

– Steve Jobs

Compl-AI, a solution for ensuring regulatory compliance of business processes

Markets such as financial services and insurance, energy, pharmaceuticals, and automotive operate within a highly regulated environment. In recent years, the proliferation of laws, rules, and regulations has been growing exponentially. In 2024, over 50 new regulations have appeared for banks alone.

Apart from the increasing number of regulations, businesses often face scenarios where regulations from different sectors intersect, necessitating multiple review processes. For international entities like banks, the compliance burden is compounded by the need to navigate the distinct regulatory frameworks of each country. This escalating complexity presents a significant challenge, particularly for smaller companies that struggle to afford the expertise and tools required for comprehensive regulatory compliance.

At Intellias, we are now piloting Compl-AI, an AI-based solution that automatically analyzes regulations and assesses their impact on business processes. Startups and mid-sized businesses can benefit from these capabilities to streamline and enhance their compliance efforts:

- ▶ **Regulatory analysis.** Businesses can upload their process descriptions to receive tailored regulatory assessments. The solution provides insights into how specific regulations impact their operations, enabling proactive compliance management.
- ▶ **Monthly updated library of regulations.** Our clients gain access to a regularly updated library of legal texts, serving as a resource for regulatory analysis. This ensures that businesses stay informed about the latest regulatory developments and can adapt their compliance strategies accordingly.
- ▶ **Monthly newsletter.** Subscribers receive a regular newsletter containing updates on new regulations, current analysis results, and benchmark comparisons. The newsletter helps businesses stay ahead of regulatory changes and benchmark their compliance efforts against industry standards.
- ▶ **Cloud-based installation.** The solution is seamlessly integrated into the cloud infrastructure of businesses, ensuring seamless accessibility, scalability, and security. Cloud deployment facilitates

easy implementation and maintenance, minimizing IT overhead.

[Get in touch](#) with the Sales team to benefit from the early adoption of Compl-AI and enhance your regulatory compliance strategy.

IntelliAssistant, a platform for enhancing corporate productivity and employee experience

As AI becomes integral for handling complex operational and technology challenges, its adoption is no longer a choice but a necessity to safeguard the future of any business. AI assistants can reduce employee learning curves and minimize operational overhead, freeing up resources to focus on strategic initiatives. We incorporated our vision of AI-powered assistants in [IntelliAssistant](#), our propitiatory platform for building custom digital assistants and AI-powered chatbots tailored to specific needs.

IntelliAssistant's ready-to-go infrastructure allows companies to swiftly launch a digital assistant and integrate it into corporate systems via the cloud, deploying a frontend, backend, and LLM model, while building specific functionality that best serves specific business needs.

Our platform is an all-in-one AI assistant that boosts productivity across the organization. It the tool that can enhance coding quality and speed, reduce administrative burdens, and mitigate sensitivities due to its advanced capabilities:

- ▶ Pre-trained model on client-specific content for superior efficiency
- ▶ Copilot functionality that speeds up coding tasks and everyday cross-functional assistance capabilities
- ▶ Automated personalized interactions
- ▶ Compliant and secure ChatGPT functionality with rich communication features
- ▶ Cross-functional adaptability, including IT, talent management, sales, marketing, and more
- ▶ Responsible data governance and ethical use of AI technology

IntelliAssistant in Numbers

2,000+

enterprise-level users

6x

faster development

40%

cost savings on deployment

We piloted the use of IntelliAssistant with our 3500 employees, demonstrating its effectiveness across these use cases:

- ▶ **Smarter enterprise knowledge management.** IntelliAssistant centralizes access to an updated company knowledge repository using AI, providing consistent, security-compliant insights from trusted corporate-wide sources.
- ▶ **Personalized upskilling/reskilling.** Employees get personalized learning recommendations based on their skills gaps, career aspirations, and company goals. IntelliAssistant facilitates mentor-mentee connections, using integrated knowledge repositories for all the functions.
- ▶ **AI-powered crisis management assistance.** With emergency assistance modules, Intellias manages crisis and force majeure situations across the enterprise, sends timely alerts to employees based on event criteria, and gathers safety data for proactive HR assistance, ensuring effective crisis communication.
- ▶ **AI for sales assistance and productivity.** Our Sales team benefits from AI-driven collateral provided by IntelliAssistant, which offers curated content, real-time data analytics, and proficiency insights from the enterprise-wide knowledge repository.

[Connect with us](#) to create a custom AI assistant that will automate your workflows.

AI Copilot, a digital assistant for software engineering

With the increasing complexity of software projects and the demand for rapid development cycles, we recognized that traditional coding practices needed a transformative boost. Inspired by the capabilities of GitHub Copilot Business, we developed our own AI-powered coding assistant to meet and even exceed these capabilities, all coming at a more favorable price. As with many of our proprietary products that we develop in-house to meet our specific needs, we named this tool IntelliCopilot, signifying its role as a development companion for Intellis engineers.

The primary goals of IntelliCopilot were twofold: first, to match the capabilities and convenience of GitHub Copilot Business for Intellis engineers, and second, to offer these benefits at a lower price.

Within just four months, we rolled out an MVP for IntelliCopilot, significantly exceeding our initial goals. The result is a convenient AI coding assistant that offers secure, on-demand access to the ChatGPT-4 Turbo model, hosted on Azure. Impressively, IntelliCopilot proved to be 3.8 times more cost-effective while maintaining a comparable level of capabilities and convenience to its top-ranking counterpart. Unlike a traditional subscription, IntelliCopilot's pricing is based on the number of tokens used, offering a more flexible and economical approach.

Feature	IntelliCopilot	GitHub Copilot Business
Chat interface	★ ★ ★ ★ ★	★ ★ ★ ★ ★
Code completions	★ ★ ★	★ ★ ★ ★ ★
Code highlights	★ ★ ★ ★ ★	★ ★ ★ ★ ★
Code insertion	★ ★ ★ ★ ★	★ ★ ★ ★ ★
Contextual understanding	★ ★ ★ ★	★ ★ ★ ★ ★
Employee authentication	★ ★ ★ ★ ★	★ ★ ★ ★ ★
Helpers on context menu	★ ★ ★ ★ ★	★ ★ ★ ★ ★

IntelliCopilot is equipped with several **advanced features**:

- ▶ **Integrated IDE tool.** Unlike ChatGPT, IntelliCopilot is built into the developer's IDE, making it convenient for work and immediately available within the same environment. The copilot is available for the three most widely used IDEs: Visual Studio, Visual Studio Code, and IntelliJ IDEA.
- ▶ **Contextual file submission.** Engineers can ask questions, request text generation, or refactor code based on the content of one or multiple files.
- ▶ **Data privacy and security.** Deployed on our Azure service, IntelliCopilot operates within a secure sandbox environment. This ensures that no Intellis data or client data is used for training the model, maintaining a high level of data privacy and security.
- ▶ **Model-agnostic flexibility.** IntelliCopilot features a model-agnostic design, allowing it to integrate with any popular AI model offered by various AI technology vendors, including Anthropic, Ollama, and Mistral. This flexibility ensures that IntelliCopilot can adapt to the latest advancements in AI technology, providing continuous improvement and staying at the forefront of AI-assisted coding.

[Get in touch](#) with us to boost productivity across your engineering teams with a secure, model-agnostic, and cost-effective AI coding assistant.

Key takeaways

- ▶ Intellis **is taking proactive steps** to develop AI tools to address various business needs.
- ▶ Intellis **has expanded its AI portfolio**, which now includes Compl-AI for regulatory compliance, IntelliAssistant for enhancing corporate productivity and the employee experience, and IntelliCopilot for improving software engineering efficiency.

By monitoring future AI trends and finding new areas for AI application, Intellis will continue to grow its AI product portfolio to deliver innovative solutions that meet evolving market demands and contribute to technological advancement.



08

The future of AI in software development

“

The future belongs to those who see possibilities before they become obvious.”

– John Sculley

Emerging trends

Forward-thinking companies embrace emerging trends to lead them into the future. This might be a future where AI not only augments the capabilities of software developers but also transforms the entire software development lifecycle, making it more efficient, accessible, and secure. With the evolution of AI technologies, staying ahead of these trends will be crucial for development teams and businesses aiming to leverage the full potential of AI.

Relying on decades of experience in software development and the adoption of new emerging technologies, we foresee that AI evolution will manifest in a multitude of ways.

Enhanced code generation and auto-completion

Without a doubt, advancements in AI will change the way we write and optimize code. Moving beyond traditional auto-completion features, future AI copilots will offer **enhanced code generation and auto-completion capabilities**. These AI assistants will be not merely reactive but proactive, leveraging context awareness to comprehend project-specific nuances.

Being aware of the developer's intent and the existing code base styles, future AI copilots will generate code that aligns closely with the task at hand. They will deeply understand the context within which they operate. This means AI copilots will be able to grasp the intricacies of the project they will assist with and anticipate developer needs, making code suggestions that seamlessly integrate with the code base.

As technology continues to evolve, we will see an **expansion of the languages and frameworks supported by AI**. This expansion will not only cover mainstream languages and frameworks but will also accommodate niche or emerging ones. By embracing a wider spectrum of languages and frameworks, AI copilots will cater to the diverse needs of developers, irrespective of the specific technologies. Consequently, developers will leverage these AI copilots across a broader range of projects, enhancing efficiency and productivity in software development.

AI-powered testing and debugging

We believe that AI-powered testing and debugging will transform quality control in software development. Through **automated bug detection**, AI will predict and identify potential bugs and vulnerabilities in real-time, even before code execution. By analyzing patterns and historical data, AI algorithms will proactively detect issues, providing developers with early warnings and opportunities for preemptive action.

Another emerging trend is **test case generation by AI**. AI copilots will automate test case generation, ensuring comprehensive coverage while saving significant time for developers. The copilots will adaptively learn from past test outcomes, enabling continuous improvement in future test case generation strategies. Such iterative learning will enhance the effectiveness of testing procedures, ultimately leading to better software products.

Natural language to code conversion

Natural language to code conversion has become a huge leap forward in programming capabilities. Due to **advancements in Natural Language Processing (NLP)**, developers will describe functionalities in plain language, while AI copilots will translate these descriptions into syntactically correct and efficient code. This breakthrough will make programming more accessible, non-developers will be able to articulate ideas in their native language, reducing the barrier to entry for coding novices.

We assume that the impact of this capability will extend beyond the traditional development cycle. By bridging the gap between natural language and code, non-developers can contribute directly to the development workflow. This **broader application scope** will democratize software creation, empowering individuals from diverse backgrounds to seamlessly contribute ideas and insights. As a result, the collaborative potential of software development will be greatly enhanced, leading to more inclusive and innovative outcomes.

AI-driven project management and planning

Given our current understanding, AI-driven project management and planning will redefine software development methodologies. Leveraging historical data, AI will enable **predictive project management**, allowing teams to forecast project timelines accurately, anticipate potential bottlenecks, and optimize resource allocation for maximum efficiency.

With the pace of innovation, AI copilots will facilitate **real-time adaptation** within project workflows. By continuously monitoring progress and external factors, AI copilots will offer timely suggestions for adjusting project scopes and timelines. This adaptive approach will ensure that projects remain agile and responsive, ultimately leading to greater success rates and higher levels of client satisfaction.

Seamless integration of AI into development environments

From our current perspective, it appears probable that integration of AI into development environments will transform the way software is created. AI copilots will play a central role in this transformation. They will seamlessly merge into development environments, providing contributors with a **unified development experience**. This integrated approach will blend coding, testing, and deployment tasks into a cohesive workflow, streamlining the entire development process.

We believe that another emerging trend will be the flexibility to tailor AI assistants to suit the individual preferences of software developers. These **customizable AI copilots** will help developers craft personalized development environments. By aligning with specific workflow preferences and project needs, these customized assistants will enhance productivity and foster a more efficient and effective development process.

Ethical and secure coding practices

Ethical and secure coding practices have always been a priority in software development. **AI's role in security** will also expand, with AI-driven tools being increasingly used to enforce best practices. These tools will automatically review code, identifying security vulnerabilities, and ensuring adherence to industry standards. By integrating AI into the security process, developers will enhance the robustness of their applications and mitigate potential risks more effectively.

As AI assumes a more significant role in coding, there will be a growing emphasis on **ethical considerations**. Ensuring that AI-generated code aligns with ethical guidelines is crucial. This will entail implementing AI tools that prevent the creation of biased or discriminatory software. By prioritizing ethical code generation, developers will contribute to building more inclusive and equitable technology ecosystems.

Collaborative AI development

Peering into the future, we envision that through **enhanced collaboration tools**, AI will usher in a new era of seamless teamwork. These tools will enable real-time code collaboration, intelligent merging of code changes, and suggestions for conflict resolution. By integrating AI into a collaboration framework, software development will become more efficient, with smoother processes for managing and integrating contributions from multiple team members.

Perhaps most importantly, AI will bridge the gap between different disciplines within development teams. By facilitating **cross-disciplinary collaboration** between software developers, data scientists, designers, and domain experts, AI will break down silos and foster a more integrated approach to project development. This holistic collaboration will ensure that diverse perspectives are leveraged effectively, leading to more innovative solutions and enhanced project outcomes.

Preparing for what's next

The underlying **AI and machine learning technologies** that power enhanced code generation and auto-completion are **advancing rapidly**. As a result, these technologies are continuously improving in terms of understanding context, predicting developer intentions, and generating more accurate and efficient code. As AI and machine learning technologies become more sophisticated, the utility and effectiveness of code generation tools will only increase.

The trend towards supporting a **broader range of programming languages and frameworks** will make enhanced code generation and auto-completion tools applicable to a wider array of software projects. This means that regardless of the specific technology stack, developers will benefit from AI assistance, making this trend impactful across the entire software development industry.

Key takeaways

- ▶ AI will **accelerate writing code**, providing proactive, context-aware code suggestions that integrate seamlessly with project-specific requirements and styles.
- ▶ AI will **transform quality control** by identifying bugs in real time and automating test case generation, which will lead to higher quality software products and more efficient testing procedures.
- ▶ NLP advancements will allow **developers and non-developers to describe functionalities in plain language**, making software engineering more accessible.
- ▶ AI will **enable predictive project management** by forecasting timelines, anticipating bottlenecks, and optimizing resource allocation, while facilitating real-time adaptation within project workflows for greater agility.

Let's proactively watch these trends and identify areas that can be enhanced with [AI solutions developed by Intellias](#) as well as other market leaders to get real-time guidance, find inspiration, and be more productive.

Afterword

In closing, our AI journey at Intellias has been like following a series of intricate recipes—each step contributing to continuous growth and innovation. As we conclude this book, we reflect on the strides we’ve made in enhancing our development processes and internal systems with AI capabilities, much like perfecting a complex dish. Our commitment remains unwavering—to help our clients achieve the utmost success in their development endeavors, continuously identifying areas where AI can bring profound value, just as a master chef discovers new ingredients to enhance a recipe.

We extend our sincerest gratitude to you, our readers, for joining us on this exploration and experiencing firsthand the transformative potential of AI outlined within these pages.

As we turn the final page of this book, we invite you to stay tuned for future editions, where we will continue to delve deeper into the ever-evolving landscape of AI. We will share new insights, discoveries, and advancements, much like unveiling new recipes in a beloved cookbook. Until the next edition of the book comes out, may the possibilities of AI continue to inspire and propel us toward a future of endless innovation and growth!

Whether you're a Software Developer or a business owner interested in increasing productivity through AI, Coding with AI cookbook is your companion in exploring the power of AI for faster, smarter, and better coding. The book summarizes practical insights of Intellias Technology Office, who researched into AI capabilities and tools on real-life projects.

- ▶ Discover ways to optimize coding tasks of Software Developers, Test Automation and DevOps Engineers
- ▶ View on-the-project code snippets for Java, JavaScript, and .NET stacks
- ▶ Find out which AI tool is better: GitHub Copilot Business, Tabnine or Amazon CodeWhisperer
- ▶ Explore strategies for automating processes like documentation, debugging, and code optimization
- ▶ Gain insights into AI-related ethical considerations and security measures
- ▶ Learn about [Intellias AI offerings](#) for ensuring regulatory compliance and enhancing coding productivity and employee experience.