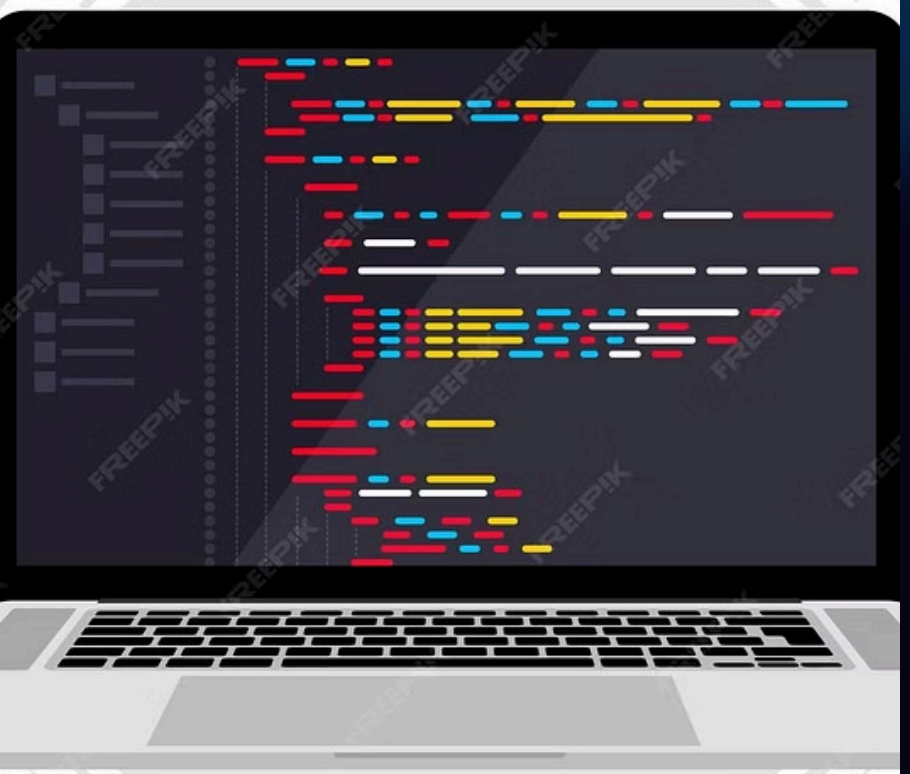


GitHub Copilot: Revolutionizing Code Writing

Welcome to the future of coding! GitHub Copilot is an AI-powered coding assistant that generates code suggestions in real-time, revolutionizing the way developers write code.

Introduction to GitHub Copilot

GitHub Copilot is an AI-powered coding assistant (AI Pair Programmer) developed by GitHub and OpenAI. It uses machine learning algorithms to provide developers with intelligent and context-aware code suggestions while they write code.

GitHub Copilot Background

Development History

- **Launched**: June 2021 (Technical Preview)
- **Developed by**: GitHub in collaboration with OpenAI
- **Powered by**: OpenAI Codex, a descendant of GPT-3
- **General Availability**: June 2022

Training Data

- Billions of lines of public code from GitHub repositories
- Natural language text from the internet
- Multiple programming languages and frameworks
- Open-source projects and documentation

Key Milestones

- **2021**: Initial launch as technical preview
- **2022**: General availability for individuals
- **2023**: GitHub Copilot X announced with chat and voice features
- **2023**: GitHub Copilot for Business launched
- **2024**: Enterprise features and enhanced security controls
- **2024:** Multi-model Copilot, Copilot Workspace, and Autofix announced at GitHub Universe
- **2024:** Free GitHub Copilot plan introduced for VS Code users
- **2025:** Copilot Agent Mode unveiled for autonomous, multi-step coding workflows
- **2025:** GitHub Copilot for Azure reached general availability with deep DevOps integration

Create GitHub Account & Activate Copilot

Step 1: Create a GitHub Account

1. Go to **github.com**
2. Click **Sign up** in the top right corner
3. Enter your **email address** and click **Continue**
4. Create a **strong password** and click **Continue**
5. Choose a **username** and click **Continue**
6. Verify you're not a robot by solving the puzzle
7. Click **Create account**
8. Enter the **verification code** sent to your email
9. Complete your profile setup (optional)

Step 2: Subscribe to GitHub Copilot

For Individual Users:

1. Sign in to your GitHub account at **github.com**
2. Click your **profile photo** in the top right → Select **Settings**
3. In the left sidebar, click **Copilot**
4. Click **Enable GitHub Copilot**
5. Choose your billing cycle:
   - **Monthly**: $10/month
   - **Yearly**: $100/year (save $20)
6. Add your **payment information**
7. Click **Save** to complete subscription

For Students & Teachers (Free):

1. Verify your academic status at **education.github.com**
2. Apply for **GitHub Student Developer Pack** or **GitHub Teacher Toolbox**
3. Once approved, GitHub Copilot is included **free**

Step 3: Verify Activation

1. Go to **github.com/settings/copilot**
2. Confirm your subscription status shows **Active**
3. Configure your preferences (optional)
4. Now install Copilot in your IDE!

Install GitHub Copilot - VS Code

Visual Studio Code (Windows)

Installation

1. **Open VS Code** → Click the **Extensions** icon in the sidebar (Ctrl+Shift+X)
2. **Search for "GitHub Copilot"** in the Extensions Marketplace
3. Click **Install** on the official GitHub Copilot extension
4. **Restart VS Code** if prompted

Authentication

1. After installation, click the **GitHub Copilot icon** in the status bar (bottom right)
2. Click **Sign in to GitHub**
3. Click **Copy and Open** to get the device code
4. **Paste the code** in your browser and click **Continue**
5. Click **Authorize GitHub Copilot Plugin**
6. Return to VS Code to start using Copilot

Install GitHub Copilot - IntelliJ IDEA

IntelliJ IDEA & JetBrains IDEs (Windows)

Installation

1. **Open IntelliJ IDEA** → Go to **File** → **Settings** (Ctrl+Alt+S)
2. **Navigate to Plugins** → Click **Marketplace**
3. **Search for "GitHub Copilot"** → Click **Install** on the official GitHub plugin
4. **Restart IDE** when prompted

Authentication

1. After restart, click **Tools** → **GitHub Copilot** → **Login to GitHub**
2. Click **Copy and Open** to get the device code
3. **Paste the code** in your browser and click **Continue**
4. Click **Authorize GitHub Copilot Plugin**
5. Click **OK** to start using Copilot

Install GitHub Copilot - Eclipse IDE

Eclipse IDE (Windows)

Installation Options

Option 1: Eclipse Marketplace

1. Go to **Help** → **Eclipse Marketplace**
2. **Search for "GitHub Copilot"**
3. Click **Install** → Select the plugin → Click **Confirm**
4. **Accept the license** and click **Finish**
5. **Restart Eclipse**

Option 2: Update Site

1. Go to **Help** → **Install New Software...**
2. Click **Add...** → Enter:
   - **Name**: GitHub Copilot
   - **Location**: `https://azuredownloads-g3ahgwb5b8bkbxhd.b01.azurefd.net/github-copilot/`
3. **Select the bundles** you want to install and click **Next**
4. **Follow the installation wizard** and restart Eclipse

Authentication

1. In the **bottom right corner**, click the **Copilot icon** → **Sign In to GitHub**
2. Click **Copy Code and Open** to get the device code
3. **Paste the code** in your browser and click **Continue**
4. Click **Authorize GitHub Copilot Plugin**
5. Click **OK** to begin using Copilot

Setup GitHub Codespaces

What is GitHub Codespaces?

GitHub Codespaces is a cloud-based development environment that lets you code directly in your browser or VS Code, with GitHub Copilot fully integrated.

Prerequisites

- Active GitHub account
- GitHub Copilot subscription (Individual, Business, or Enterprise)
- Repository access (your own or organization's)

Step 1: Enable Codespaces

1. Sign in to **github.com**
2. Go to **Settings** → **Codespaces**
3. Review your usage limits:
    - **Free tier**: 120 core hours/month + 15 GB storage
    - **Pro/Team**: Higher limits available
4. Configure default settings (optional):
    - Default editor (VS Code browser/desktop)
    - Region preference
    - Timeout settings

Step 2: Create a Codespace

From a Repository:

1. Navigate to your repository on GitHub
2. Click the **Code** button (green)
3. Select the **Codespaces** tab
4. Click **Create codespace on main** (or your branch)
5. Wait for environment to build (1-3 minutes)

From GitHub.com:

1. Go to **github.com/codespaces**
2. Click **New codespace**
3. Select repository and branch
4. Choose machine type (2-core, 4-core, 8-core)
5. Click **Create codespace**

Access & Manage GitHub Codespaces

Step 3: Access Your Codespace

Browser Access:

Codespace opens automatically in browser

- Full VS Code experience in your browser
- GitHub Copilot works immediately

VS Code Desktop:

1. Install **GitHub Codespaces** extension in VS Code
2. Click **Open in VS Code Desktop** in browser
3. Or: VS Code → Command Palette → **Codespaces: Connect to Codespace**

Step 4: Verify Copilot in Codespaces

1. Open any code file in your Codespace
2. Look for **Copilot icon** in status bar (bottom right)
3. Start typing code to see suggestions
4. Press **Tab** to accept suggestions
5. Use **Ctrl+I** (Cmd+I on Mac) to open Copilot Chat

Managing Codespaces

Stop

Click your codespace name → **Stop codespace**

Delete

Settings → Delete unused codespaces

Reconnect

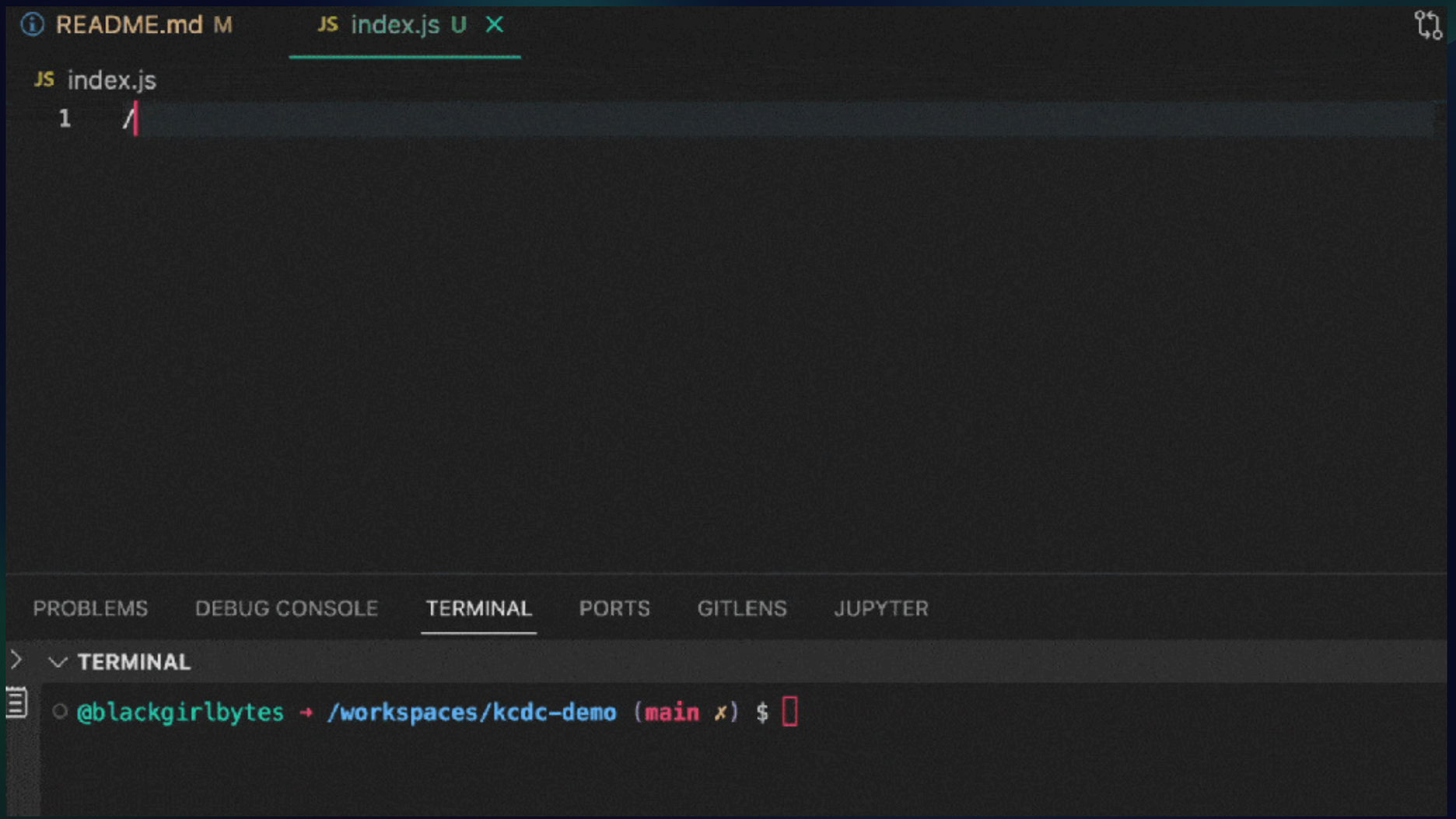
github.com/codespaces → Click your codespace

Export

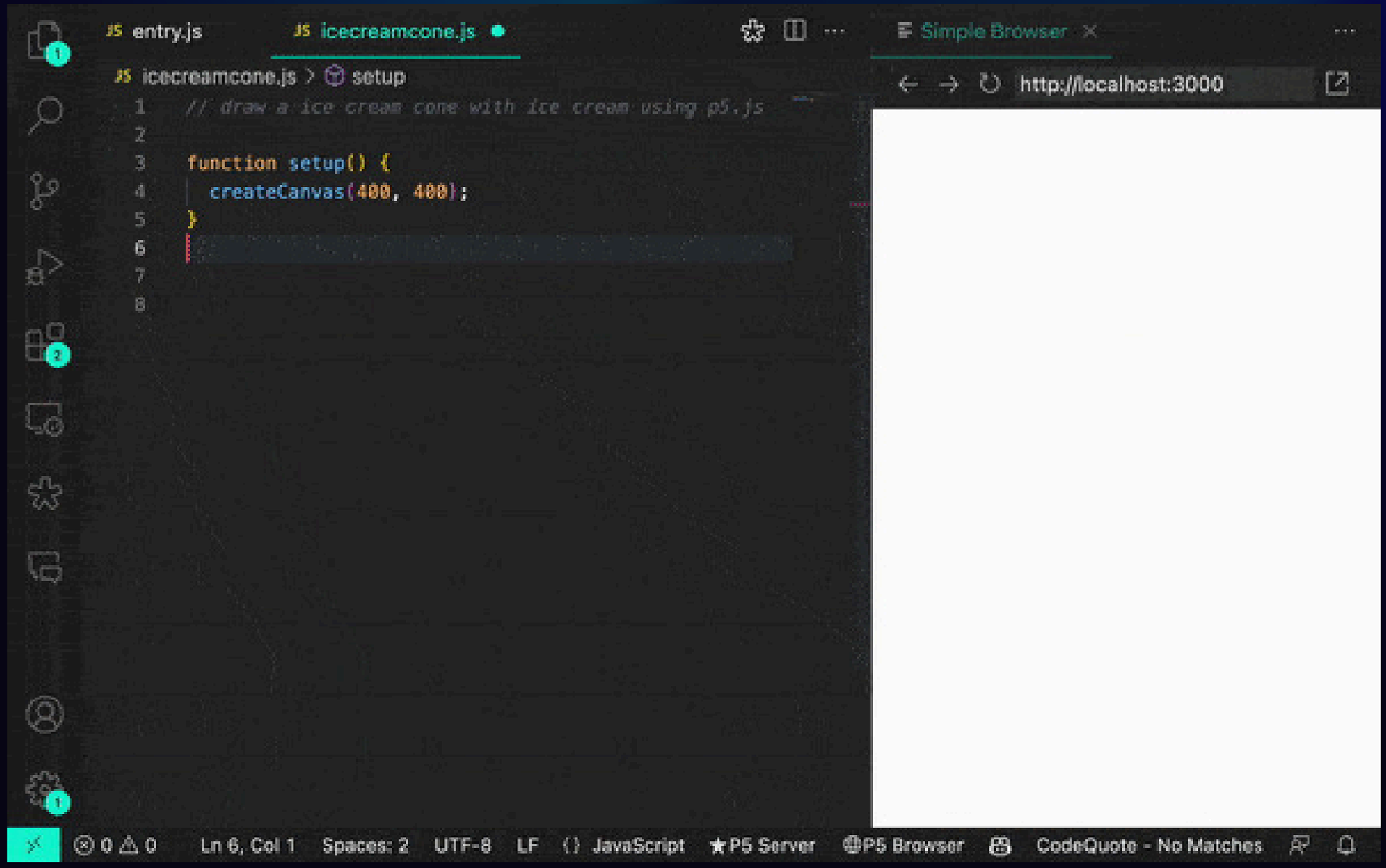
Download changes before deleting

How It Works

GitHub Copilot learns from billions of lines of code and developer patterns to offer intelligent code suggestions. It understands the programming context and generates code snippets that can be easily incorporated into your projects.



One More Example



Features of Copilot

1. Auto-completion of code snippets
2. Context-aware code generation
3. Support for multiple programming languages
4. Seamless integration with code editors
5. Assistance with code refactoring

Two Principles of Prompting

1. Write clear and specific instructions
2. Give copilot a time to think

Benefits and Advantages of Using Copilot

Increased Productivity

GitHub Copilot helps developers write code faster, reducing the time spent on repetitive tasks and increasing productivity.

Error Prevention and Code Quality

Copilot's intelligent suggestions help catch potential errors and improve overall code quality, resulting in more robust and maintainable projects.

Learning and Skill Development

Using Copilot exposes developers to a wide range of coding styles and best practices, accelerating their learning and skill development.

Potential Challenges and Limitations

1. **Contextual Understanding**

   Copilot may not always understand the specific context of your code, leading to inaccurate or irrelevant suggestions that need manual review.

2. **Dependency on Internet Connection**

   Copilot requires an active internet connection for model inference, meaning it might not be available in offline environments.

Keyboard Shortcuts (VS Code)

Core Inline Suggestions

- Tab - Accept inline suggestion
- Escape (Esc) - Reject/dismiss inline suggestion
- Alt +] - Navigate to next suggestion
- Alt + [- Navigate to previous suggestion
- Alt + / - Trigger inline suggestion manually
- Ctrl + Right Arrow - Accept suggestion at word level

Chat Features

Opening Chat

- Ctrl + Alt + I (Windows/Linux) / Cmd + Option + I (Mac) - Open Chat view
- Ctrl + I (Windows/Linux) / Cmd + I (Mac) - Start inline chat in editor or terminal
- Ctrl + Shift + Alt + L (Windows/Linux) / Shift + Option + Cmd + L (Mac) - Open Quick Chat

Chat Management

- Ctrl + N (Windows/Linux) / Cmd + N (Mac) - Start new chat session
- Ctrl + Shift + I (Windows) / Ctrl + Shift + Alt + I (Linux) / Shift + Cmd + I (Mac) - Switch to agent mode
- Ctrl + Alt + . (Windows/Linux) / Option + Cmd + . (Mac) - Show model picker

Voice Chat

- Ctrl + I (Windows/Linux) / Cmd + I (Mac)(hold) - Start inline voice chat

Advanced Features

- Ctrl + Enter - Show combined suggestions panel
- Ctrl + Shift + Alt + e - Show Labs feature list
- Ctrl + Shift + a - Toggle Copilot sidebar

Context and Tools When using chat, you can enhance your prompts with:

- / - Use slash commands for common tasks
- @ - Mention chat participants for domain-specific requests

4 Different Modes

- Ask Mode
- Edit Mode
- Plan Mode
- Agent Mode

When to use Ask Mode?

Learning and Understanding Code

Use cases:

- Explaining what existing code does
- Understanding complex algorithms
- Getting clarification on programming concepts
- Learning about frameworks or libraries
- Asking "What does this code do?"

Quick Questions and Guidance

Use cases:

- Asking for syntax help
- Getting best practices
- Understanding error messages
- Getting documentation summaries
- Asking "How do I..." questions

Code Review and Analysis

Use cases:

- Getting suggestions for improvements
- Identifying potential issues or code smells
- Finding edge cases
- Understanding security implications
- Asking "What's wrong with this code?"

When to use Ask Mode? (continued)

Debugging Assistance

Use cases:

- Understanding why code isn't working
- Getting explanations for error messages
- Troubleshooting logical flow issues
- Understanding the cause of exceptions
- Asking "Why is this failing?"

Quick Reference

Use cases:

- Looking up library usage examples
- Getting syntax reminders
- Understanding how to use APIs
- Learning about language features
- Asking "Show me an example of..."

Conceptual Questions

Use cases:

- Understanding differences between approaches
- Comparing frameworks or libraries
- Learning about performance implications
- Understanding when to use patterns
- Asking "What's the difference between..."

When to Use Plan Mode?

Project Planning & Architecture

Use cases:

- Breaking down large features into tasks
- Planning application structure
- Designing system architecture
- Creating implementation roadmaps
- Outlining refactoring steps

Collaborative Planning

Use cases:

- Creating plans for team review
- Documenting approach before development
- Getting alignment on strategy
- Breaking user stories into technical tasks
- Planning sprint work and effort estimation

Understanding Implementation

Use cases:

- Getting step-by-step plans before coding
- Identifying files to create or modify
- Understanding dependencies and prerequisites
- Evaluating different solution approaches
- Planning implementation order

Risk Assessment

Use cases:

- Identifying potential challenges early
- Understanding scope and impact of changes
- Discovering edge cases to handle
- Planning testing and validation needs
- Assessing technical debt

When to use Agent Mode?

Multi-File Operations

Use cases:

- Refactoring code architecture across files
- Migrating projects to new languages/frameworks
- Bulk changes across codebase
- Modernizing legacy code
- Updating namespaces and variables

End-to-End Development

Use cases:

- Building applications from scratch
- Implementing high-level requirements
- Creating complete CRUD operations
- Developing full workflows
- Integrating error handling and validation

Testing & Quality

Use cases:

- Generating comprehensive unit tests
- Creating integration tests
- Writing test suites for edge cases
- Implementing automated testing
- Testing multiple components

When to use Agent Mode? (continued)

Code Analysis & Bug Resolution

Use cases:

- Finding and fixing bugs in project structure
- Identifying security vulnerabilities
- Performing code quality analysis
- Resolving null reference exceptions
- Implementing security fixes

Documentation & Maintenance

Use cases:

- Generating comprehensive API docs
- Creating README files
- Adding XML documentation to methods
- Updating docs when code changes
- Writing technical documentation

Complex Problem Solving

Use cases:

- Debugging multi-file issues
- Optimizing performance bottlenecks
- Implementing complex algorithms
- Solving architectural challenges
- Iterative problem resolution

When NOT to Use Agent Mode

Agent mode is overkill for simple tasks. Use standard Copilot for:

- Simple code completion and autocomplete suggestions
- Writing individual functions or small code blocks
- Quick syntax help or simple questions
- Single-file modifications that don't require workspace context

Key Advantages of Agent Mode

Autonomous Operation: Unlike ask mode, agent mode doesn't stop after one response. It continues working until the goal is achieved or user input is needed.

Context Awareness: It understands your entire workspace, not just the current file, enabling more intelligent suggestions.

Error Detection and Correction: Agent mode can recognize when its changes cause errors and automatically iterate to fix them.

Tool Integration: It can run terminal commands, execute tests, and use various development tools as part of its workflow.

When to Use Ask Mode vs Agent Mode?

Use Ask Mode when:

- You want quick answers without code changes
- You need explanations or learning support
- You're exploring concepts or getting references
- You want to understand existing code
- You need simple code snippets or examples

Use Agent Mode when:

- You want Copilot to actually modify your files
- You need complex, multi-step implementations
- You want autonomous execution of tasks
- You need workspace-wide changes or analysis

When to Use Edit Mode?

Targeted Multi-File Refactoring

Use cases:

- Refactoring code using async/await patterns
- Converting synchronous to asynchronous code
- Implementing consistent error handling
- Modernizing code to newer language features
- Applying design patterns across files

Quick Specific Updates

Use cases:

- Adding error handling to functions
- Updating variable names consistently
- Adding logging statements to methods
- Implementing input validation
- Adding TypeScript types to JavaScript files

Code Quality Improvements

Use cases:

- Removing nested conditions
- Optimizing functions for performance
- Improving code readability and structure
- Breaking down complex functions
- Adding documentation and comments

When to Use Edit Mode? (continued)

Bug Fixes & Corrections

Use cases:

- Fixing null reference exceptions
- Correcting logic errors in functions
- Handling edge cases
- Fixing memory leaks or resource disposal
- Correcting API integration problems

Feature Additions

Use cases:

- Adding new properties to classes
- Implementing new methods in interfaces
- Adding unit tests for classes/functions
- Creating validation logic for forms
- Adding authentication checks to controllers

Refactoring & Cleanup

Use cases:

- Simplifying complex code structures
- Removing code duplication
- Extracting reusable components
- Improving naming conventions
- Standardizing code formatting

Key Advantages of Edit Mode

Granular Control: You choose exactly which files Copilot can modify, giving you precise control over the scope of changes.

Review-Ready Diffs: All changes are shown as inline diffs that you can review before accepting, ensuring you understand every modification.

Surgical Precision: Perfect for making specific improvements without affecting the broader system architecture.

Iterative Process: You can make requests, review changes, accept or reject them, and iterate until you get the desired result.

Checkpoint System: Some implementations offer rollback capabilities to revert to previous states if needed.

When to Use Edit Mode vs Other Modes?

Use Edit Mode when:

- You know exactly what needs to be changed and in which files
- You want to review every change before it's applied
- You need surgical precision without touching unrelated code
- You want to make "one commit's worth" of focused changes
- You're working in a brownfield application and need careful control

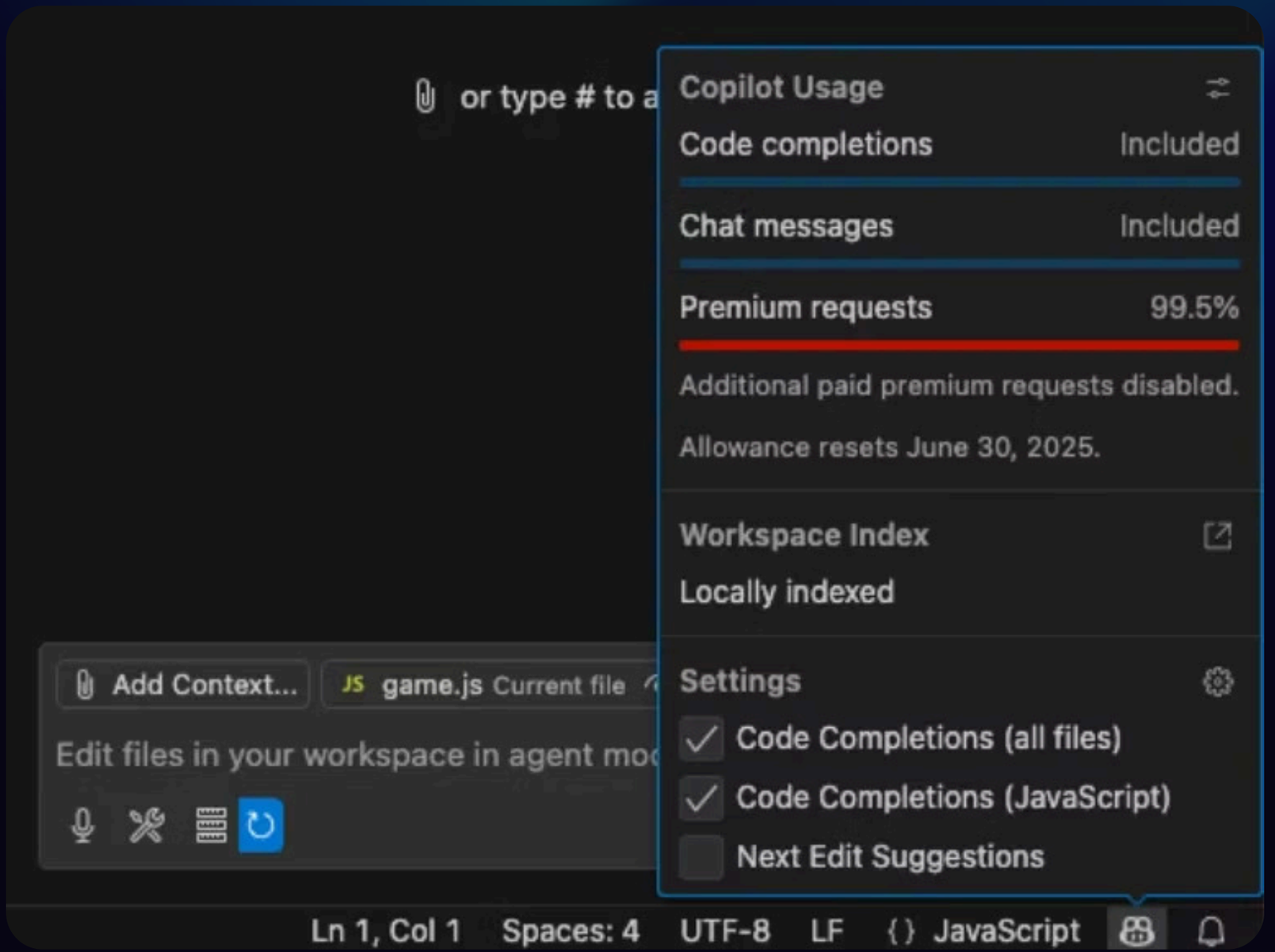
Use Ask Mode when:

- You need explanations or learning support without making changes
- You want code examples or guidance

Use Agent Mode when:

- You want autonomous execution across your entire workspace
- You need complex, multi-step implementations
- You're comfortable with Copilot making broader architectural decisions

Viewing Usage in IDE



Enterprise Considerations

GitHub Copilot for Business: Offers enhanced privacy protections:

- Does not train on private organizational code
- Provides better data isolation
- Includes additional security controls
- Offers enterprise-grade compliance features

Alternative Approaches:

- Use Copilot selectively for non-sensitive projects
- Implement air-gapped development environments for critical code
- Consider on-premises AI coding solutions for sensitive projects
- Maintain separate development workflows for different security levels

Chat Variables

Chat variables help you include specific context in your prompts by referencing code elements directly.

Syntax

Type `#` followed by the variable name in the chat prompt box.

Available Variables

Variable	Description
#block	Includes the current block of code
#class	Includes the current class
#comment	Includes the current comment
#file	Includes the current file's content
#function	Includes the current function or method
#line	Includes the current line of code
#path	Includes the file path
#project	Includes the project context
#selection	Includes the currently selected text
#sym	Includes the current symbol
#codebase	Includes broader codebase context
#git	Includes Git repository information
#editor	Includes editor context

Example Usage

```
#file:gameReducer.js #file:gameInit.js how are these files related
```

Chat Participants

Chat participants act as domain experts with specialized knowledge areas.

Syntax

Type `@` followed by the participant name to scope your prompt to a specific domain.

Common Participants

Participant	Specialty
`@workspace`	Workspace and project-wide questions
`@terminal`	Terminal and command-line operations
`@vscode`	VS Code-specific questions
`@github`	GitHub-related queries

Example Usage

```
@workspace how are notifications scheduled @terminal how to update an npm package
```

Slash Commands

Slash commands provide shortcuts for common coding tasks without writing complex prompts.

Syntax

Type `/` followed by the command name in the chat prompt box.

Common Commands

Command	Purpose
`/explain`	Explain selected code
`/fix`	Suggest fixes for problems
`/tests`	Generate unit tests
`/doc`	Generate documentation
`/optimize`	Optimize code performance
`/refactor`	Refactor code structure
`/review`	Review code for issues

Example Usage

```
/tests for the getCurrentUser function
/explain this sorting algorithm
/fix the authentication bug
```

GitHub Copilot in SDLC

GitHub Copilot can be integrated throughout the Software Development Life Cycle (SDLC) to enhance productivity and code quality at every stage.

SDLC Stages with Copilot

01

Planning & Requirements

Assist in clarifying requirements and generating initial code structures or documentation outlines.

02

Design & Architecture

Suggest design patterns, API structures, and architectural components based on best practices.

03

Development & Implementation

Generate boilerplate code, complete functions, and offer inline code suggestions to accelerate coding.

04

Testing & Quality Assurance

Write unit tests, suggest test cases, and help identify potential bugs or edge cases.

05

Deployment & Release

Help draft deployment scripts, configuration files, and release notes for smoother rollouts.

06

Maintenance & Support

Assist in understanding legacy code, refactoring, and generating fixes or new features during maintenance.

Each stage benefits from Copilot's AI-powered assistance to accelerate delivery and improve outcomes.

1. Planning & Requirements Stage

How Copilot Helps

Plan Mode for Project Planning:

- Break down user stories into technical tasks
- Create implementation roadmaps
- Identify technical dependencies
- Estimate effort and complexity

Practical Applications:

- Generate project structure templates
- Create technical specification documents
- Plan API endpoints and data models
- Identify potential technical challenges early

Ask Mode for Requirements Analysis:

- Clarify technical requirements
- Understand feasibility of features
- Get suggestions for technology stack
- Explore architectural patterns

Example Prompts:

"Plan the implementation of a user authentication system with JWT tokens"

"What are the technical requirements for building a real-time chat feature?"

2. Design & Architecture Stage

How Copilot Helps

Plan Mode for Architecture Design:

- Design system architecture and components
- Plan database schemas and relationships
- Create API design specifications
- Outline design patterns to use

Practical Applications:

- Generate class diagrams and structure
- Create database migration scripts
- Design RESTful API endpoints
- Plan microservices architecture

Ask Mode for Design Decisions:

- Compare different architectural approaches
- Understand design pattern implementations
- Get best practices for scalability
- Learn about security considerations

Example Prompts:

"Design a scalable e-commerce database schema with products, orders, and users"

"What's the best architecture pattern for a real-time notification system?"

3. Development & Implementation Stage

How Copilot Helps

Inline Suggestions:

- Auto-complete code as you type
- Generate boilerplate code quickly
- Suggest function implementations
- Complete repetitive patterns

Agent Mode:

- Build complete features from requirements
- Implement multi-file functionality
- Create CRUD operations
- Develop full workflows with error handling

Edit Mode:

- Update code across multiple files
- Apply consistent patterns
- Modernize legacy code
- Add error handling and validation

Practical Applications

- Write API endpoints and controllers
- Implement business logic
- Create data access layers
- Build UI components

Example Prompts

"Create a REST API endpoint for user registration with email validation"

"Implement a shopping cart feature with add, remove, and update quantity"

4. Testing & Quality Assurance Stage

How Copilot Helps

Agent Mode:

- Generate comprehensive unit tests
- Create integration test suites
- Write test cases for edge cases
- Implement automated testing workflows

Slash Commands:

- `/tests` - Generate unit tests
- `/review` - Review code for issues
- `/fix` - Suggest fixes for failing tests

Ask Mode:

- Understand testing best practices
- Learn about test coverage
- Get guidance on mocking
- Explore testing frameworks

Practical Applications

- Write unit tests for all functions
- Create integration tests for APIs
- Generate test data and fixtures
- Implement end-to-end test scenarios

Example Prompts

"Generate unit tests for the user authentication service with edge cases"

"Create integration tests for the payment processing API"

5. Deployment & Release Stage

How Copilot Helps

Agent Mode for Deployment Scripts:

- Generate CI/CD pipeline configurations
- Create Docker and Kubernetes files
- Write deployment automation scripts
- Build infrastructure as code

Practical Applications:

- Create Dockerfile and docker-compose files
- Generate GitHub Actions workflows
- Write deployment scripts for various environments
- Configure environment variables and secrets

Ask Mode for Deployment Guidance:

- Understand deployment best practices
- Learn about containerization strategies
- Get cloud platform recommendations
- Explore monitoring and logging setup

Example Prompts:

"Create a GitHub Actions workflow for deploying a Node.js app to AWS"

"Generate a Dockerfile for a Python Flask application with production settings"

6. Maintenance & Support Stage

How Copilot Helps

Agent Mode for Bug Fixes:

- Find and fix bugs across codebase
- Identify security vulnerabilities
- Implement performance optimizations
- Resolve production issues

Ask Mode for Troubleshooting:

- Debug production errors
- Understand error logs and stack traces
- Get suggestions for performance issues
- Learn about monitoring best practices

Edit Mode for Updates:

- Apply patches and updates
- Refactor legacy code
- Add new features to existing code
- Update dependencies and libraries

Agent Mode for Documentation:

- Generate API documentation
- Create README files
- Update technical documentation
- Add code comments and explanations

Practical Applications

- Fix reported bugs and issues
- Optimize slow-performing code
- Update deprecated dependencies
- Maintain comprehensive documentation

Example Prompts

"Find and fix the memory leak in the data processing service"

"Generate comprehensive API documentation for all endpoints"

Model Context Protocol (MCP)

What is MCP?

Model Context Protocol (MCP) is an open protocol that standardizes how applications provide context to Large Language Models (LLMs). It enables seamless integration between AI assistants like GitHub Copilot and external data sources, tools, and services.

Key Benefits

For Developers:

- Connect AI to any data source
- Build integrations once, use everywhere
- Standardized protocol for context sharing
- Reduced integration complexity

For AI Assistants:

- Access to real-time data
- Enhanced contextual understanding
- Improved code suggestions
- Better decision-making capabilities

How MCP Works with GitHub Copilot

- Access your company's internal documentation
- Connect to databases and APIs
- Retrieve project-specific context
- Integrate with development tools and services

This enables more accurate, context-aware code suggestions tailored to your specific environment and requirements.

GitHub Copilot MCP Integration

Connecting Copilot with MCP

GitHub Copilot can integrate with Model Context Protocol (MCP) servers to access external context sources, enabling more accurate and context-aware code suggestions tailored to your specific environment.

Integration Benefits

Enhanced Context Awareness:

- Company-specific documentation access
- Internal API specifications
- Database schemas and structures
- Knowledge bases and wikis
- Project-specific coding standards
- Internal library usage patterns
- Company best practices

Practical Use Cases:

- Query internal documentation
- Access proprietary API definitions
- Retrieve coding guidelines
- Connect to project management tools
- Access deployment configurations
- Integrate with testing frameworks
- Context-aware error handling

Setting Up MCP with Copilot

1. **Install or Connect to MCP Server:** Deploy your own MCP server or connect to a remote MCP server for your data sources
2. **Configure VS Code:** Add MCP server configuration to VS Code settings
3. **Define Resources:** Specify which data sources to expose via MCP
4. **Authenticate:** Set up secure authentication for MCP connections
5. **Test Connection:** Verify Copilot can access MCP resources
6. **Start Using:** Copilot will now use MCP context in suggestions

Example MCP Integrations

Project Management: Jira, Azure DevOps, Linear, Asana • **Documentation:** Confluence, Notion, SharePoint, internal wikis • **Code Repositories:** GitLab, Bitbucket, internal repos • **Databases:** PostgreSQL, MySQL, MongoDB schemas • **APIs:** OpenAPI specs, GraphQL schemas, REST doc

Security Considerations

Code Quality Risks

- **Vulnerable Code Generation**: May suggest insecure patterns (SQL injection, XSS, weak authentication)
- **Outdated Practices**: Could recommend deprecated libraries with known vulnerabilities
- **Insufficient Input Validation**: Generated code may lack proper security checks

Data Exposure Concerns

- **Secret Leakage**: 6.4% higher risk of exposing API keys, credentials, and tokens in code
- **Codebase Scanning**: Copilot reads entire workspace for context, potentially exposing proprietary logic
- **Third-Party Data Sharing**: Prompts may be sent to external services like Bing Search

Best Practices

- **Mandatory Code Review**: Treat all AI suggestions as untrusted code
- **Security Scanning**: Use automated tools like CodeQL and GitGuardian
- **Secret Management**: Eliminate plaintext credentials, use secure vaults
- **Testing Protocol**: Thoroughly test all AI-generated code before deployment

Enterprise Solutions

- **GitHub Copilot for Business**: Enhanced privacy protections and data isolation
- **Push Protection**: Enable secret scanning with automatic blocking
- **Air-Gapped Environments**: Use for highly sensitive projects
- **Policy Development**: Create clear AI usage guidelines and training

Risk Mitigation Strategies

Security Best Practices

Code Review Protocol:

- Treat all Copilot suggestions as untrusted code
- Implement mandatory security reviews for AI-generated code
- Use automated security scanning tools
- Test all suggestions thoroughly before deployment

Secret Management:

- Eliminate plaintext credentials from codebases entirely
- Use secure credential management systems
- Implement pre-commit hooks to detect secrets
- Deploy tools like GitGuardian for continuous secret scanning

Enable Security Features:

- Use GitHub's CodeQL for vulnerability detection
- Enable Copilot Autofix for automated vulnerability remediation
- Activate secret scanning with push protection
- Implement dependency vulnerability scanning

Legal Protection Measures

Policy Development:

- Create clear guidelines for AI tool usage
- Establish code review requirements for AI-generated content
- Define acceptable use policies for different project types
- Train developers on intellectual property considerations

License Compliance:

- Maintain awareness of open-source license requirements
- Implement license scanning tools
- Document code sources and attributions
- Consider legal review for commercial projects using AI-generated code

Important Security Settings in github.com

- Ask Copilot to find and fix vulnerabilities

  > Go to your repository → Security tab → Code scanning alerts, then click "**Copilot Autofix**" checkbox next to any vulnerability for AI-powered analysis and fix suggestions.

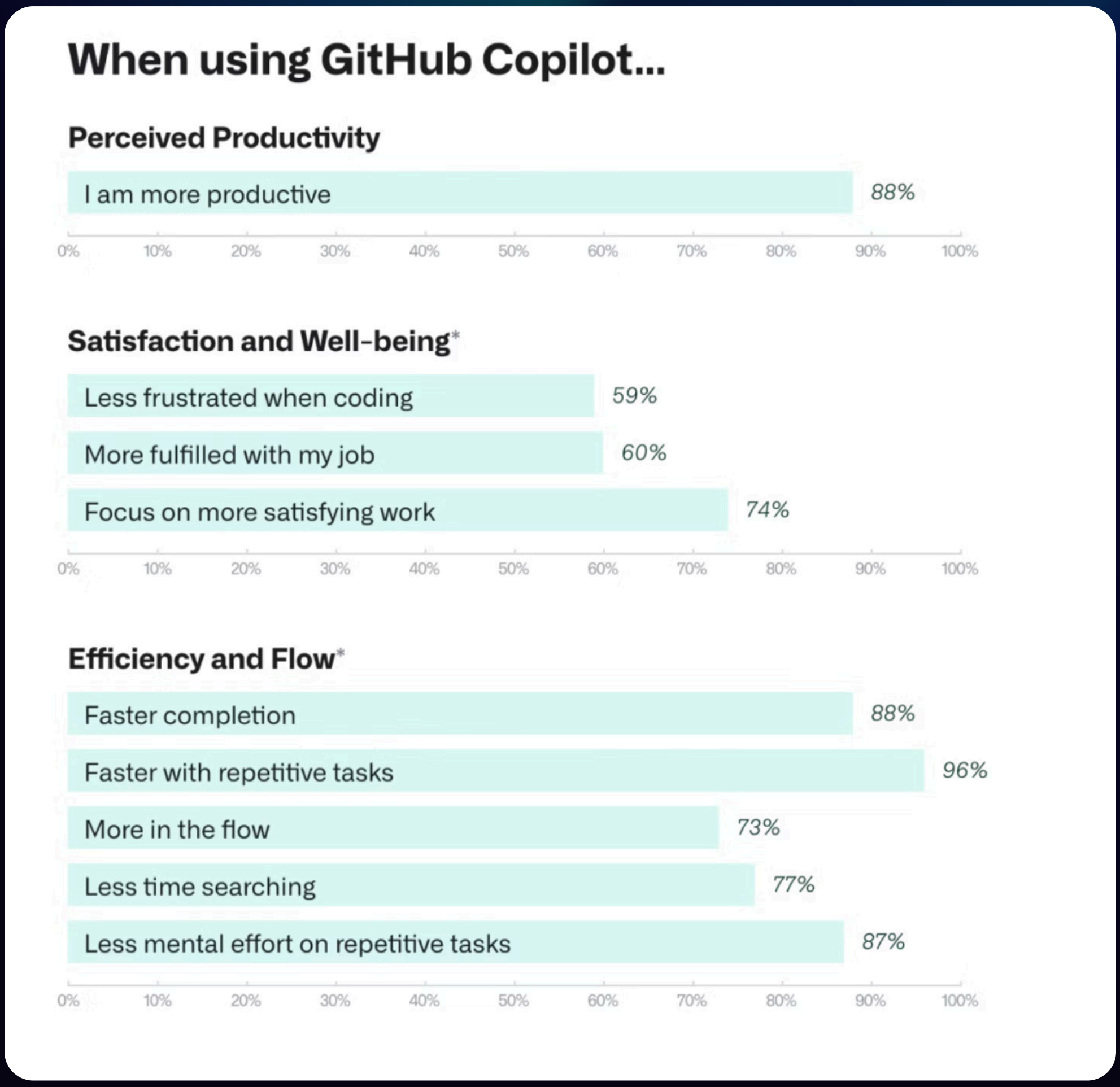
- Ask Copilot for more details when problems are found

  > In any security alert or pull request, use the Copilot chat icon to ask follow-up questions like "How serious is this vulnerability?" or "What are the potential impacts?"

- Enable Dependabot, code scanning, and secret scanning

  > Navigate to Settings → Security & analysis in your repository and toggle on "Dependency graph", "Dependabot alerts", "Code scanning", and "Secret scanning" with one-click setup.

Real-World Examples and Success Stories



Refer: https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/

Future Developments and Improvements

1. **Enhanced Language Support**

   GitHub Copilot aims to support an ever-increasing number of programming languages, making it accessible to a wider range of developers.

2. **Improved Contextual Understanding**

   Continued development of Copilot's machine learning capabilities will enhance its ability to understand complex code contexts and generate more accurate suggestions.

3. **Collaborative Coding**

   GitHub Copilot envisions enabling seamless collaboration among developers, allowing them to work together on code snippets and projects using the power of AI.

GitHub Copilot Examples

Angular or React

NodeJS or Python or Java or .Net

Playwright or Cypress or Selenium

MCP Integration

FAQ - Copilot

- What data has GitHub Copilot been trained on?

  GitHub Copilot is powered by a generative AI model developed by GitHub, OpenAI, and Microsoft. It has been trained on natural language text and source code from publicly available sources, including code in public repositories on GitHub.

- Will GitHub Copilot help me write code for a new platform?

  GitHub Copilot is trained on public code. When a new library, framework, or API is released, there is less public code available for the model to learn from. That reduces GitHub Copilot's ability to provide suggestions for the new codebase.

- Will my code be shared with other users?

  No. We follow responsible practices in accordance with our **Privacy Statement** to ensure that neither your Prompts or Suggestions will be shared or used as suggested code for other users of GitHub Copilot.

- Can GitHub Copilot introduce insecure code in its suggestions?

  Public code may contain insecure coding patterns, bugs, or references to outdated APIs or idioms. When GitHub Copilot synthesizes code suggestions based on this data, it can also synthesize code that contains these undesirable patterns. This is something we care a lot about at GitHub, and as of March 2023 we launched an **AI-based vulnerability prevention system** that blocks insecure code patterns in real-time to make GitHub Copilot suggestions more secure.

- Does GitHub own the code generated by GitHub Copilot?

  GitHub Copilot is a tool, like a compiler or a pen. GitHub does not own the suggestions GitHub Copilot provides to you. You are responsible for the code you write with GitHub Copilot's help. We recommend that you carefully test, review, and vet the code before pushing it to production, as you would with any code you write that incorporates material you did not independently originate.

Conclusion and Key Takeaways

GitHub Copilot is changing the way developers write code by providing intelligent and context-aware suggestions. Embrace this powerful tool to boost your productivity, improve code quality, and enhance your coding skills.

Thank you