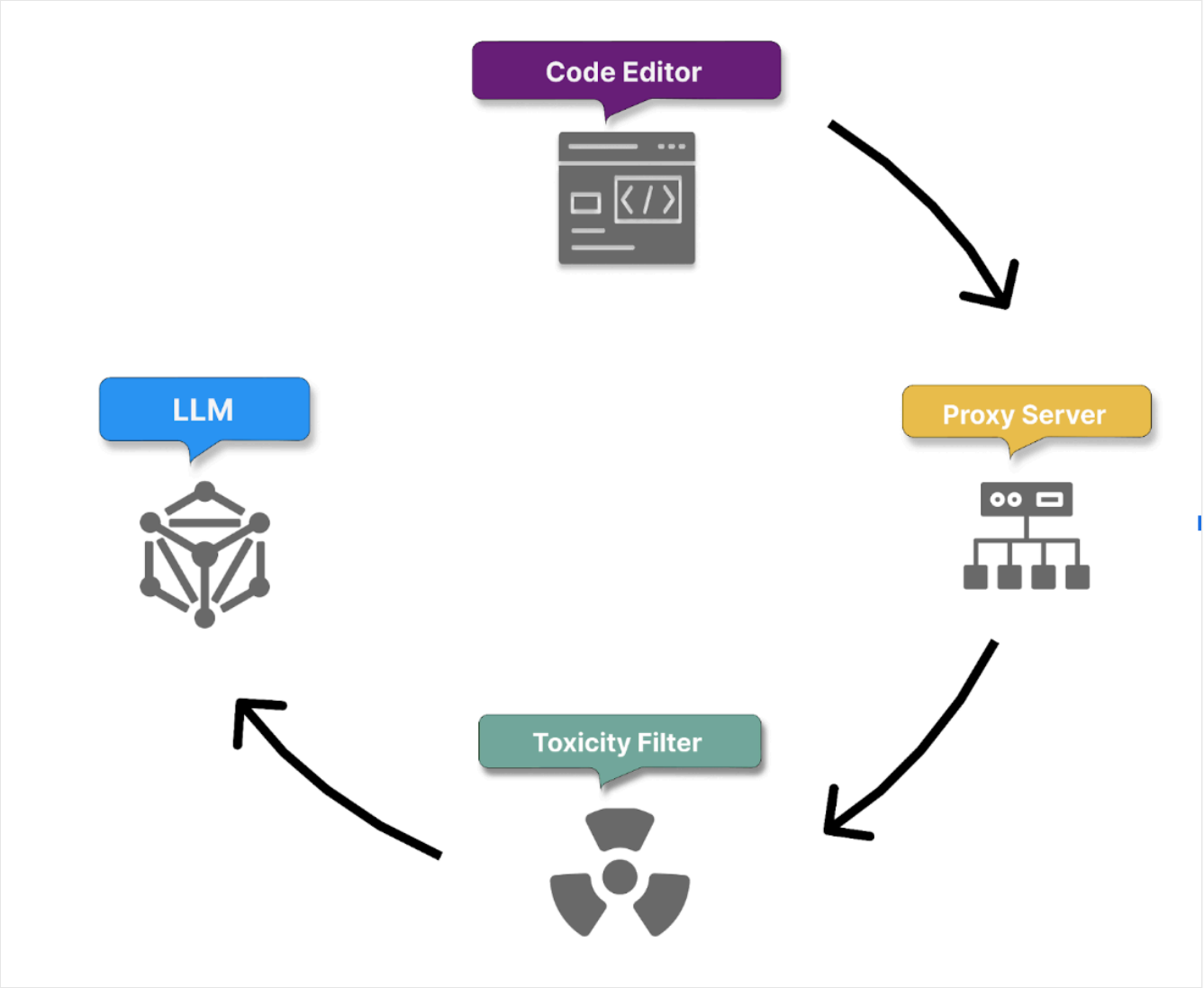
100 XP

GitHub Copilot user prompt process flow

5 minutes

In this unit, we'll break down how GitHub Copilot turns your prompts into smart, usable code. Generally, GitHub Copilot receives prompts and returns code suggestions or responses in its data flow. This process suggests an inbound and outbound flow.

Inbound flow:



Let's walk through all the steps Copilot takes to process a user's prompt into a code suggestion.

1. Secure prompt transmission and context gathering

The process begins with the secure transmission of the user prompt over HTTPS. This ensures that your natural language comment is sent to GitHub Copilot's servers securely and confidentially, protecting sensitive information.

GitHub Copilot securely receives the user prompt, which could be a Copilot chat or a natural language comment provided by you within your code.

Simultaneously, Copilot collects context details:

- Code before and after the cursor position, which helps it understand the immediate context of the prompt.
- Filename and type of the file being edited, allowing it to tailor code suggestions to the specific file type.
- Information about adjacent open tabs, ensuring that the generated code aligns with other code segments in the same project.
- Information on project structure and file paths
- Information on programming languages and frameworks
- Pre-processing using Fill-in-the-Middle (FIM) technique to consider both the preceding and following code context, effectively expanding the model's understanding allowing Copilot to generate more accurate and relevant code suggestions by leveraging a broader context.

These steps translate the user's high-level request into a concrete coding task.

2. Proxy filter

Once the context is gathered and the prompt is built, it passes securely to a proxy server hosted in a GitHub-owned Microsoft Azure tenant. The proxy filters traffic, blocking attempts to hack the prompt or manipulate the system into revealing details about how the model generates code suggestions.

3. Toxicity filtering

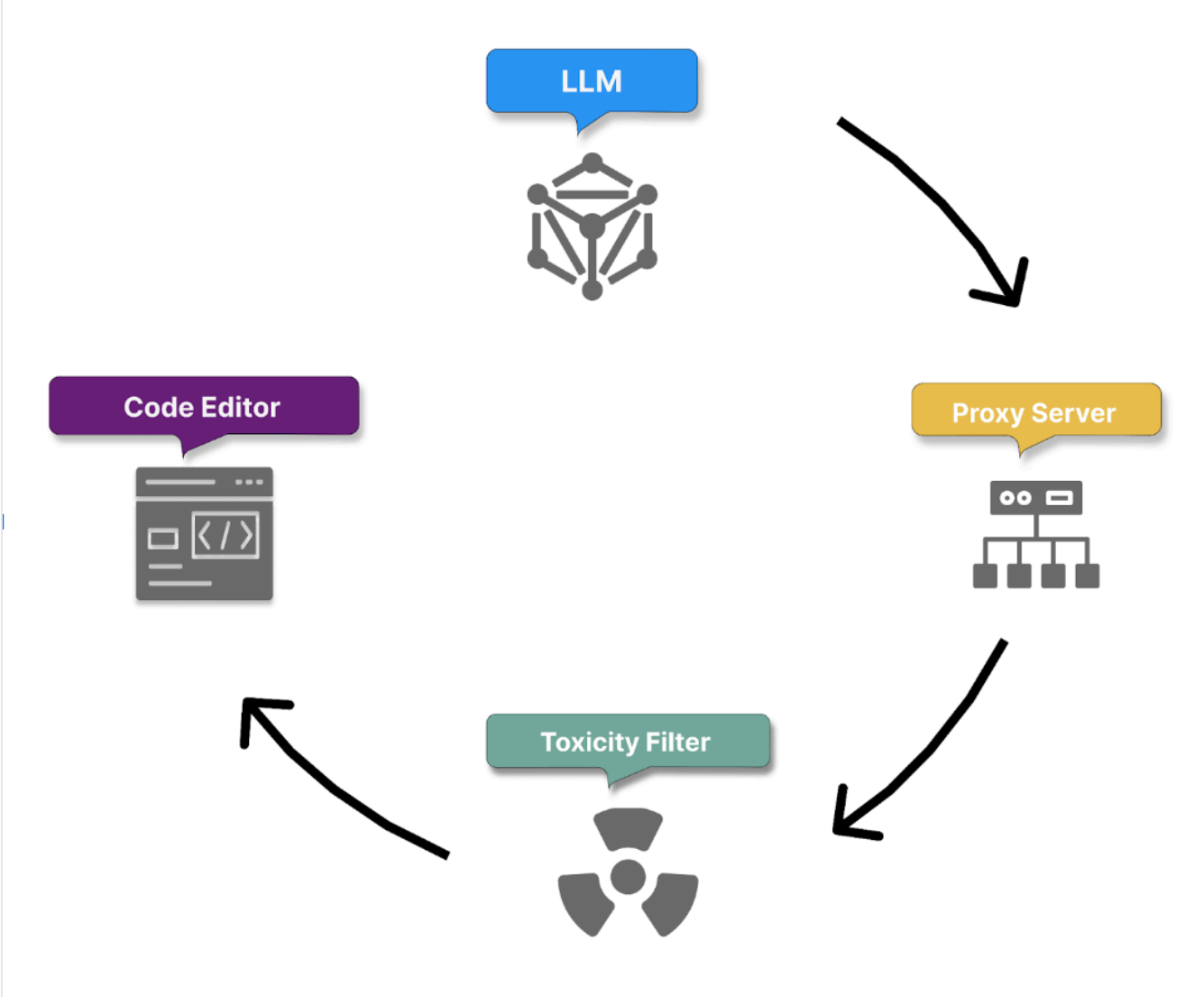
Copilot incorporates content filtering mechanisms before proceeding with intent extraction and code generation, to ensure that the generated code and responses don't include or promote:

- **Hate speech and inappropriate content**: Copilot employs algorithms to detect and prevent the intake of hate speech, offensive language, or inappropriate content that could be harmful or offensive.
- **Personal data**: Copilot actively filters out any personal data, such as names, addresses, or identification numbers, to protect user privacy and data security.

4. Code generation with LLM

Finally, the filtered and analyzed prompt is passed to LLM Models, which generate appropriate code suggestions. These suggestions are based on Copilot's understanding of the prompt and the surrounding context, ensuring that the generated code is relevant, functional, and aligned with project-specific requirements.

Outbound Flow:



5. Post-processing and response validation

Once the model produces its responses, the toxicity filter removes any harmful or offensive generated content. The proxy server then applies a final layer of checks to ensure code quality, security, and ethical standards. These checks include:

- **Code quality**: Responses are checked for common bugs or vulnerabilities, such as cross-site scripting (XSS) or SQL injection, ensuring that the generated code is robust and secure.

- **Matching public code (optional)**: Optionally, administrators can enable a filter that prevents Copilot from returning suggestions over ~150 characters if they closely resemble existing public code on GitHub. This prevents coincidental matches from being suggested as original content. If any part of the response fails these checks, it is either truncated or discarded.

6. Suggestion delivery and feedback loop initiation

Only responses that pass all filters are delivered to the user. Copilot then initiates a feedback loop based on your actions to achieve the following:

- Grow its knowledge from accepted suggestions.
- Learn and improve through modifications and rejections of its suggestions.

7. Repeat for subsequent prompts

The process is repeated as you provide more prompts, with Copilot continuously handling user requests, understanding their intent, and generating code in response. Over time, Copilot applies the cumulative feedback and interaction data, including context details, to improve its understanding of user intent and refine its code generation capabilities.

Next unit: GitHub Copilot data

Previous Next