**Introduction:**
The manual process of code commenting in software development presents several difficulties that cause inefficiencies and obstacles in the creation and upkeep of projects. The main problems are inaccurate comments, time-consuming manual commenting, inadequate documentation, and challenges in providing meaningful context. The suggested system uses cutting-edge technologies, such as Natural Language Processing (NLP) and Large Language Models (LLMs), to automate the code commenting process to overcome these difficulties.

**Challenges Addressed:**
The problem of insufficient documentation in software projects is addressed by the tool. The lack of thorough documentation that frequently follows manual commenting makes it difficult for developers to comprehend code snippets and prevents productive teamwork. Moreover, it is difficult to give significant context through comments, which makes it harder to understand the relationships between variables and functions within the codebase. Errors and inconsistencies that could result from manual commenting are avoided by the tool, which corrects inaccurate comments. Moreover, it lessens the time-consuming nature of the manual process, particularly in big projects where it can be tiresome and easy to overlook to provide thorough explanations for every code snippet.

**Tool's Approaches:**
By automatic code commenting, the suggested system ensures accuracy and context relevance by utilizing LLMs and NLP. As a solution to the problem of a time-consuming process, the tool automates the commenting process, greatly reducing the time and effort needed for manual commenting. The inclusion of LLMs improves the readability of the codebase by enabling more precise and thorough comments. The tool's focus on reusability is online with the objective of producing well-documented code, which promotes efficient reuse across various project components. As demonstrated by the scenarios, the tool's integration into the development workflow allows for smooth team collaboration and makes code documentation an effective and developer-friendly part of the software development process.

We used GPT LLM APIS and trained the model which helps in giving more accurate comments than the already existing approach.

**how the implement tool features address the challenge**
**Incorporating Large Language Models (LLMs):**
To automatically generate code comments, the tool makes use of sophisticated LLMs such as ChatGPT. These models are optimized for coding tasks, meaning that they can comprehend code snippets and offer pertinent commentary.
Employing Natural Language Processing (NLP) Techniques:

NLP techniques are used for code snippet analysis, language comprehension, and precision measurement. The tool's algorithms effectively parse and comprehend code, extracting details like variable and function names to produce insightful comments.

**Deciphering Code Context:**
To comprehend the context of code snippets, the system is built to be intelligent. Variables, function names, and their functions within the codebase as a whole are considered so that the generated comments make sense in the context.

**Refining LLMs:**
The tool recognizes that optimizing LLMs for coding tasks is difficult. By continuously fine-tuning the process based on data and user feedback, it guarantees that the generated comments are accurate and relevant within the context.
Designing User-Friendly Interfaces:
To facilitate easy interaction between developers and the tool, an intuitive interface has been designed. Developers have control over the documentation process because they can edit or modify the comments that the models produce.

**Ensuring Scalability:**
The system places a strong emphasis on scalability, guaranteeing that it can manage sizable codebases with ease—a crucial factor for large-scale software projects. This dedication to scalability guarantees the tool's continued efficacy and responsiveness in large and complex code repositories, skillfully meeting the changing needs of large-scale software development projects.

**Usability**

Using a short sample of Python code as input, developers worked with the Automatic Comment Generation Extension in Visual Studio Code.
First, we need to download the extension and select the code we want to generate the comments then we need to select the generate comments from Ctrl + shift +P.
We added a feature where we can add custom comments according to the developer and the comments will be generated.

Project_demo.mp4

**Positive Aspects:**

**Efficiency Boost:** The tool's quick generation of comments for the supplied Python code was well-received by developers. The time usually spent on manual commenting was greatly decreased by the automated process, increasing the efficiency of development.
**Accurate Comments:** Participants observed that the generated comments correctly reflected the code's functionality. Contextually relevant comments resulted from a nuanced understanding of the code context, which was ensured by the integration of advanced language models.
**User-Friendly Interface:** The Visual Studio Code tool's user interface was positively received. The developers discovered that it was easy to use and permitted fluid communication. The flexibility to edit or personalize comments right from the interface was greatly valued by users.

**Negative Aspects:**
**Over-Generality in Comments:** Although the automatic comments were accurate in most cases, some developers noticed that they were often very generic. This resulted in remarks that were technically accurate but lacked the clarity that developers frequently seek for a deeper comprehension.
**Handling Complex Code Structures:** The program's ability to produce accurate comments was limited in some situations involving complex code structures. The tool's inability to fully capture the intricacy of the code led to comments that were not as detailed as desired, according to the developers.
**Limited Handling of Edge Cases:** Participants noted situations in which the tool had trouble managing edge cases, producing remarks that were sometimes inappropriate or even deceptive. This demonstrated the necessity of ongoing improvement, particularly when managing various coding scenarios.

These usability study findings offer important new perspectives on how developers view and use the Automatic Comment Generation Extension. Although the tool showed promise in many areas, including handling complex code structures, reducing overgeneralization, and enhancing performance in edge cases, there is still opportunity for improvement. Developer input will direct future improvements so that the tool better fits a wider range of coding requirements.