ChatGPT

Comments as Context for AI Assistants

Modern AI-powered editors like Cursor rely heavily on **code context** to generate accurate suggestions. In this setting, well-written comments serve as *explicit signals of developer intent*, providing information that the code alone may not encode. As one expert observes, "code can only tell you how the program works; comments can tell you why" [1]. AI tools use comments to "grasp the purpose and constraints that might not be immediately apparent from the code structure alone" [2]. In practice, when an AI assistant reads code with clarifying comments, it gains access to the original author's thought process and domain knowledge, improving its ability to answer questions or generate relevant edits [2] [1].

- **Context window:** Within a coding session, LLMs have a limited context window (e.g. 8K–32K tokens). Comments add tokens but enrich context. Tools like Cursor automatically pull in relevant files and use techniques (retrieval, embeddings) to feed context to the model [3] [4]. Well-placed comments (especially at a file's top or around complex logic) are more likely to fall within the model's window. Research finds that LLMs focus on the *beginning* of code: faults in the first quarter of a file are detected with far higher accuracy than those later on [5]. This suggests that comments placed early may have an outsized impact on AI comprehension.

- **AI understanding:** Contrary to the notion that AI can "just read the code," studies show LLMs actually use comments as semantic cues. For example, injecting *misleading* comments into code significantly degraded GPT-4's debugging accuracy, because the model "tries to extract a code's semantic information from the comments in addition to the code itself" [6]. In other words, **accurate comments help AI get the right answer, while incorrect or irrelevant comments can mislead it**.

- **Immediate benefits:** In the short term, comments improve AI tooling by providing context for code completion and suggestions. An LLM with comments explaining *why* something is done can generate more aligned and useful code. As noted by an AI tooling expert, "When these systems have access to commented code explaining intent and rationale, they can generate suggestions that align more closely with the developer's goals" [7]. In Cursor specifically, indexed code (including comments) is embedded for retrieval; each file's embedding captures all textual content [3]. Thus, meaningful comments can make Cursor's search and tab-completion smarter by encoding design rationale in its embeddings.

- **Long-term effects:** Over multiple sessions, comments enhance **AI "memory" and indexing**. Tools like Cursor support persistent memory structures (e.g. *Rules* and *Memory Banks* [8] [9]) that capture project context. Comments are automatically captured as part of a project's files: for example, Pieces.ai (a memory add-on) explicitly records "file names, project structure, and even your comments" into long-term memory [9]. This means comments become searchable facts in future AI interactions. Similarly, code-search tools using vector embeddings will index comments along with code, making it easier to retrieve relevant code snippets by meaning or intent. In effect, thoughtful comments can be "remembered" by AI and surfaced later during maintenance or audits.

LLM Code Comprehension and Comments

Research and developer experience align on the value of comments. A 2024 study of LLM debugging found that subtle changes like misleading comments drastically reduce LLM performance, implying that the models do read comments as part of understanding code [6]. The same work notes that LLMs tend to weight *non-functional* elements (comments, dead code) equally in attention, which can mislead them if comments don't match the code [10]. In practice, this means:

- **Accurate comments aid AI:** When comments truly reflect the code's intent, they guide the AI. For example, a correct explanation of a non-obvious algorithm helps the model reason about it.
- **Misleading comments confuse AI:** If a comment is wrong or outdated, it can "misdirect" the model. GPT-4 was fooled in a grid-based game example by a bogus comment about grid initialization [6].

In developer forums, experts warn that letting an AI generate comments *after the fact* is risky: the AI may guess intent incorrectly and produce misleading comments [11]. This underlines that comments should ideally be written by someone (or some process) that truly knows *why* code was written that way. Comments written at development time (human-authored or reviewed) are more reliable and preserve knowledge that the code doesn't contain.

Cursor IDE and Code Context

Cursor's design emphasizes context retrieval and memory. Its **Codebase Indexing** feature embeds each file (code+comments) for search [3], so any information in comments can be retrieved as needed. Cursor's "@ symbols" let you explicitly include specific code files or symbols in a query [12]. If you annotate code with clear comments or JSDoc/Python docstrings, these docblocks can be fetched as context. Moreover, Cursor's emerging "Rules" and "Memory Bank" system explicitly stores project knowledge (project brief, tech context, etc.) [13]. While these are usually written separately, you could incorporate key comments or extracted insights into those memory files for future sessions. For example, a high-level comment about a module's architecture could be summarized in `systemPatterns.md` in a Cursor memory bank.

- **Immediate context:** In an active coding session, any comment you write above a function becomes part of the file content the model sees. Cursor will use the open buffer and related files as context, so inline comments directly feed into the LLM prompt. If comments explain *why* a function exists or what edge cases it handles, the LLM can leverage that when refactoring or fixing bugs in that function.

- **Persistent memory:** Because Cursor can index and recall project files, your comments live beyond a single session. For open-source projects, Cursor on a new machine can re-index the repo (including comments) and "remember" them next time it runs. In personal/private projects, enabling privacy mode or memory banks means your comments (if stored in code) contribute to the project's stored context. Tools like Pieces.ai explicitly save comments into AI memory [9]. Thus comments act as a form of documentation that aids both immediate Q&A and future searches.

Personal vs. Open-Source Context

- **Personal/Private code:** For a personal project not shared publicly, your AI assistant's only "knowledge" of your code comes from what's loaded or stored. Well-commented code helps the AI (and future you) recall why certain approaches were taken. In private repos, AI models (especially closed models like GPT) won't have seen your code in training, so comments are the sole source of embedded insight. Using Cursor's memory or rules, you can even craft project-briefs or include important comments to prime the model each session.

- **Open-source code:** If your code is open source, large LLMs may have *seen* parts of it during training, but often not the latest version or branch. Even then, comments still add value. They explain design decisions that the LLM cannot infer from code alone. Moreover, many AI tools (like GitHub Copilot or ChatGPT with a repository plugin) retrieve the actual repository contents (including comments) when answering questions. Good comments in open codebases help any AI (or human) visitor quickly grasp complex parts. They also improve developer experience (e.g. IDE tooltips, docs generation).

In both cases, comments should be viewed not as obsolete but as **a complementary layer to AI knowledge**. With LLMs, comments become part of the rich context those models use. Proper comments can prevent hallucinations (by constraining the AI's assumptions) and reduce context-solicitation (the AI asking clarifying questions) by proactively providing needed info.

Best Practices & Potential Pitfalls

To maximize benefits, follow sound commenting practices (many of which align with traditional advice):

- **Write purposeful comments (focus on *why*):** Don't just restate what code does – AI can parse that. Explain *intent, design rationale, and non-obvious behavior*. As experts advise, comments should explain *why* certain approaches were taken [14] [1]. For example, note algorithmic assumptions, business rules, or reasons for an unusual workaround. These insights feed AI reasoning.

- **Keep comments accurate and up-to-date:** Outdated or trivial comments can mislead AI. A recent analysis showed that incorrect comments cause LLMs to make errors [6]. Regularly review comments when code changes. When fixing bugs, add or update comments to capture your reasoning (StackOverflow's best-practices note this too [15]).

- **Avoid redundant or trivial comments:** Comments that duplicate code ("i = i+1; // add one to i") clutter the prompt without adding meaning. They waste tokens and, according to industry lore, "can become out-of-date" [16]. In the AI era, such comments add noise and even risk confusing the assistant (one anti-pattern guide warns that superfluous comments "create noise that can potentially confuse AI systems" [17]). Instead, focus on explaining nuanced aspects of the code.

- **Leverage structured doc comments:** Where language support exists (Javadoc, Python docstrings, JSDoc), use them. Cursor and other tools can present these in hover tooltips or use them in analysis. A clear function signature comment not only helps AI but also populates IDE completions for other humans.

- **Use comments for domain and architectural context:** Larger design notes (e.g. at top of file or module) can be extremely useful. For instance, a module-level comment summarizing its role or a function's intended use-case provides context that AI might otherwise lack. Cursor's memory banks even encourage a "project brief" or "tech context" to be written separately [18]; embedding key comments into those documents is a great idea.

Conclusion

In the age of LLM code assistants, **comments have regained importance as a form of context-rich documentation**. They help AI tools like Cursor deliver smarter, more accurate suggestions by encoding developer knowledge that code alone does not convey [1] [2]. Both immediate code editing sessions and long-term code maintenance benefit. Comments enrich the context window, guide code generation, and become part of persistent memory/indexes that future AI sessions use [6] [9].

However, simply "relying on AI to explain code" and abandoning comments is ill-advised. Research shows LLMs do not magically infer hidden intent – they rely on cues from comments just as humans do [19]. The best practice remains: write clear, concise comments that explain *why* and preserve domain-specific insights, and avoid stale or superfluous commentary [14] [16]. By thoughtfully commenting code now, you help today's AI assistants – and tomorrow's developers – understand and build on your work.

Sources: Authoritative forums, AI tooling documentation, and recent research all emphasize that well-crafted comments boost AI code comprehension [19] [2] [6], while rote or incorrect comments can hinder it [6] [16]. These insights, combined with Cursor's context and memory features [3] [9], show that balanced commenting is more valuable than ever.

---

[1] [2] [7] [14] [17] The Importance of Code Comments for Modern AI Coding Assistants | by Dmytro Chystiakov | Medium
https://medium.com/@iZonex/the-importance-of-code-comments-for-modern-ai-coding-assistants-fbfced948a28

[3] Cursor – Codebase Indexing
https://docs.cursor.com/context/codebase-indexing

[4] [12] Cursor – Working with Context
https://docs.cursor.com/guides/working-with-context

[5] [6] [10] How Accurately Do Large Language Models Understand Code?
https://arxiv.org/html/2504.04372v1

[8] [13] [18] How to Supercharge AI Coding with Cursor Rules and Memory Banks | Lullabot
https://www.lullabot.com/articles/supercharge-your-ai-coding-cursor-rules-and-memory-banks

[9] Add AI Memory to vscode.dev - Pieces for Developers — Long term memory for your developer workstream
https://pieces.app/ai-memory/vscode-dev

[11] [19] documentation generation - Are comments obsolete in favor of Generative AI? - Software Engineering Stack Exchange
https://softwareengineering.stackexchange.com/questions/453931/are-comments-obsolete-in-favor-of-generative-ai

15 16 Best practices for writing code comments - Stack Overflow
https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/