Can We Trust AI to Write Our Code? A Developer’s Perspective

By Tyson Williams

It is recommended to view this blog post here https://tysonwilliams.dev/#/blog

AI tools such as ChatGPT and GitHub Copilot are revolutionising software and web development. With the ability to generate entire code blocks from prompts, these tools promise faster development cycles and fewer bugs. But at what cost?

As a student developer, I’ve often wondered, “Should I trust AI to write code I cannot fully explain myself?” In this project, I explore the quality, clarity, and maintainability of AI-generated code compared to my own manually written code for the same task.

In a professional setting, maintainable and readable code is critical for team collaboration, handovers, and long-term support. If developers are deploying AI-written code without fully understanding it, this poses potential risks to project sustainability and ethical integrity.

To explore this issue, I developed two versions of a simple project: one written entirely by me, and one generated using ChatGPT. I then compared both based on code readability, maintainability, and my own level of understanding.

This concern is not mine alone; several industry leaders and researchers have raised alarms about overreliance on AI tools for production code, especially in junior teams.

GitClear's "Coding on Copilot" Report (GitClear, 2024) analysed over 150 million lines of code and found that AI-assisted development with tools like GitHub Copilot can lead to a decrease in code quality. Specifically, the report highlights an increase in code churn and a decline in code reuse, suggesting that AI-generated code may be less maintainable and more prone to issues over time. These findings raise concerns about the long-term sustainability and readability of AI-produced code.

The Rise of AI-Powered Coding Tools in Modern Development Workflows

In recent years, AI-powered coding tools have transitioned from experimental novelties to integral components of modern software development. Platforms like GitHub Copilot, Replit, and Cursor are now commonplace, assisting developers in writing, debugging, and optimising code more efficiently. This shift is driven by the demand for faster development cycles and the democratisation of coding knowledge, enabling individuals with varying levels of expertise to contribute to software projects.

According to Business Insider (Barr, 2025), the integration of AI into development workflows offers several opportunities:

- Accelerated Development: AI tools can generate boilerplate code, suggest functions, and even identify bugs, significantly reducing development time.
- Knowledge Democratisation: By lowering the barrier to entry, these tools empower non-traditional developers to build functional software using natural language prompts

However, this rapid adoption also presents challenges:

- Overreliance on AI: Developers may become dependent on AI suggestions, potentially diminishing their problem-solving skills and understanding of underlying code structures.
- Code Quality Concerns: AI-generated code may lack the context-specific nuances that human developers provide, leading to maintainability issues.

As AI continues to reshape the development landscape, developers must balance the benefits of these tools with a critical understanding of their limitations.

Ethical Implications of AI-Generated Code in Software Development

The integration of AI into code generation raises several ethical concerns that developers and organisations must address:

- **Intellectual Property and Plagiarism**: AI models trained on vast code repositories may inadvertently reproduce copyrighted code without proper attribution, leading to potential legal issues. For instance, GitHub Copilot has been observed generating code snippets that closely resemble existing open-source projects, sometimes without preserving original license information. (Davis & Rajamanickam, 2022)
- **Transparency and Accountability:** When AI-generated code leads to errors or vulnerabilities, determining responsibility becomes complex. The lack of transparency in how AI models make decisions further complicates accountability.
- **Bias and Fairness:** AI tools may perpetuate existing biases present in their training data, leading to unfair or discriminatory outcomes in software applications. (BytePlus Editorial Team, 2025)

To navigate these ethical challenges, developers should:

- **Conduct Code Reviews:** Regularly review AI-generated code for potential IP infringements and biases.
- **Maintain Documentation:** Keep details records of AI tool usage and the origins of code snippets to ensure traceability.

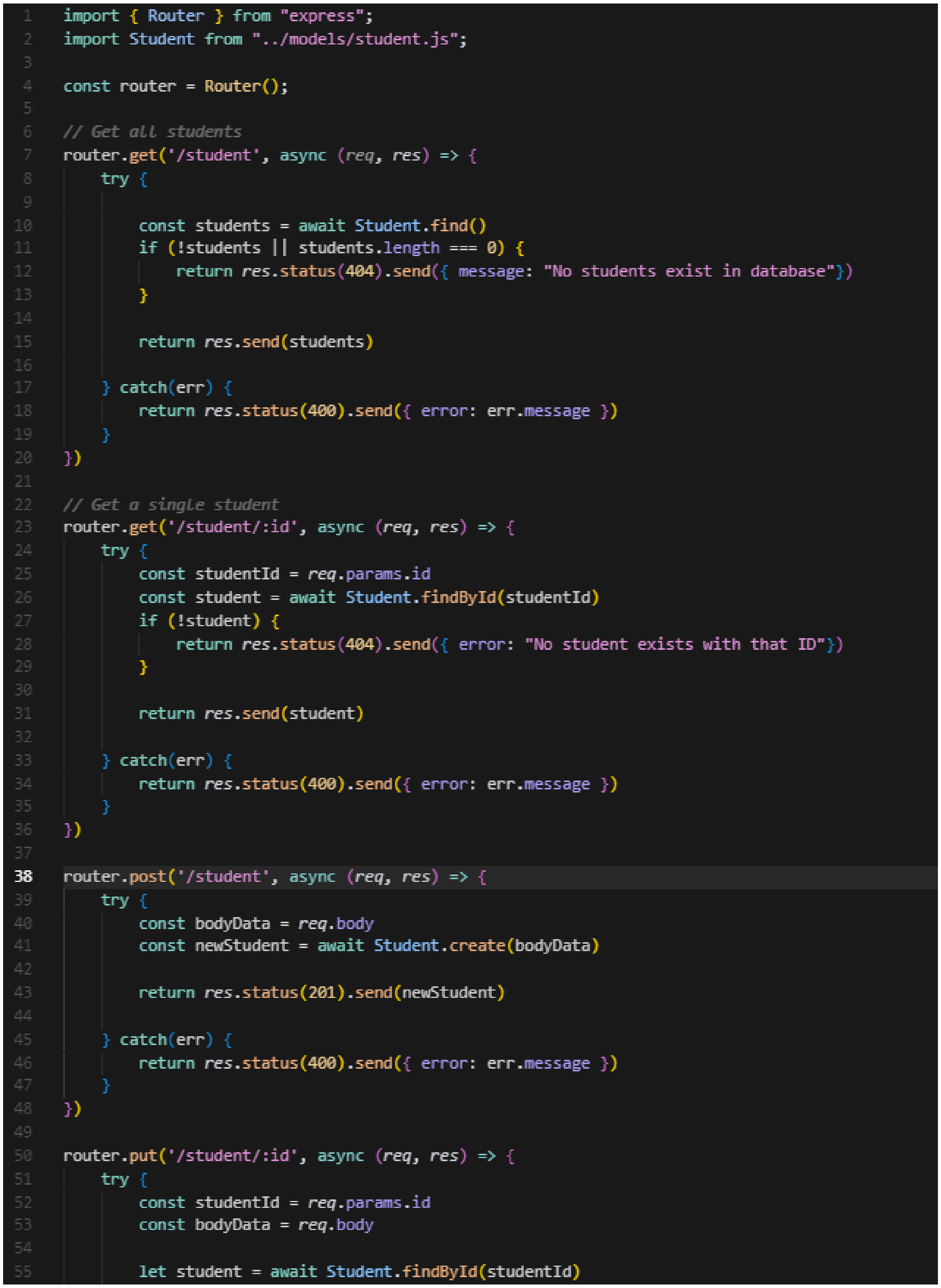
- **Implement Ethical Guidelines:** Establish clear policies on the use of AI in development to promote responsible practices. (Sanj.dev, 2025)

By proactively addressing these ethical considerations, the developer community can harness the benefits of AI while upholding standards of integrity and accountability.

My Skills vs AI

To explore the ethical and practical concerns of AI-generated code, I created a small programming project twice, once manually and once using Cursor. The chosen task was a simple CRUD application written in JavaScript (Node.js, Express). The backend lets users create, edit, and delete students, enrollments, teachers, courses, and grades.

Over a few days, using a spare few hours here and there, I was able to write my program; however, with one prompt, it took AI a mere 30 seconds. I then used AI to compare my human-written code with the AI-generated code, and AI made a harsh assessment of its work. These are the summarised results (full results can be found in the attached root file AI_vs_Human_Code_Comparison.md):

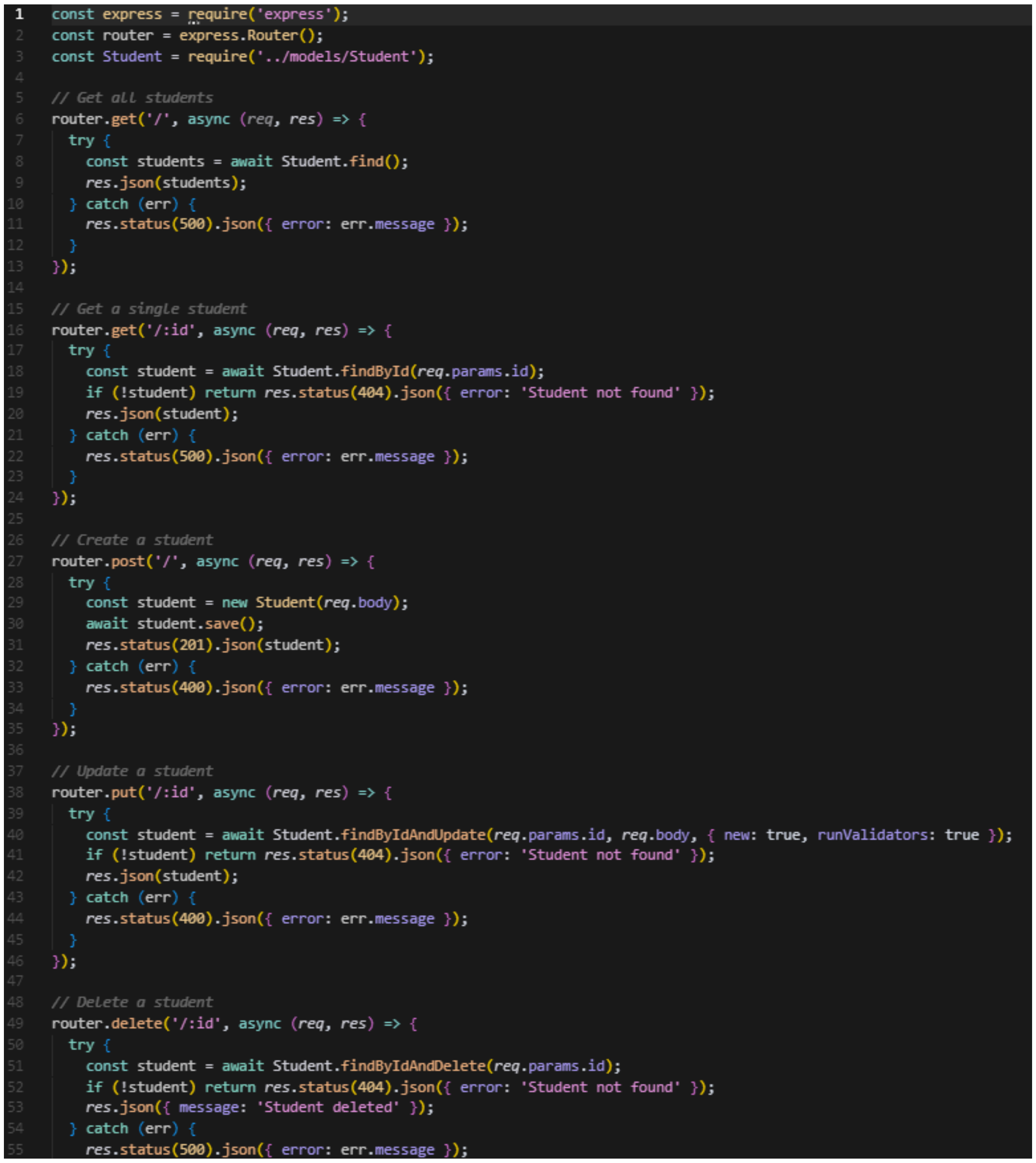


```
import { Router } from "express";
import Student from "../models/student.js";

const router = Router();

// Get all students
router.get('/student', async (req, res) => {
    try {

        const students = await Student.find()
        if (!students || students.length === 0) {
            return res.status(404).send({ message: "No students exist in database"})
        }

        return res.send(students)

    } catch(err) {
        return res.status(400).send({ error: err.message })
    }
})

// Get a single student
router.get('/student/:id', async (req, res) => {
    try {
        const studentId = req.params.id
        const student = await Student.findById(studentId)
        if (!student) {
            return res.status(404).send({ error: "No student exists with that ID"})
        }

        return res.send(student)

    } catch(err) {
        return res.status(400).send({ error: err.message })
    }
})

router.post('/student', async (req, res) => {
    try {
        const bodyData = req.body
        const newStudent = await Student.create(bodyData)

        return res.status(201).send(newStudent)

    } catch(err) {
        return res.status(400).send({ error: err.message })
    }
})

router.put('/student/:id', async (req, res) => {
    try {
        const studentId = req.params.id
        const bodyData = req.body

        let student = await Student.findById(studentId)
```

Fig 1. Human-Generated Student Routes Snippet

Human-Written API: High Distinction (92%)

- **Modern & Clean Code**: Uses ES6+ syntax, strong schema validation, and centralised error handling.
- **Secure by Design**: Includes Helmet.js, detailed input validation, and clear error messages.
- **Professional Docs**: A top-tier README with setup guides, API examples, and troubleshooting.
- **Developer-Friendly**: Modular structure, version control, and dev tools baked in.



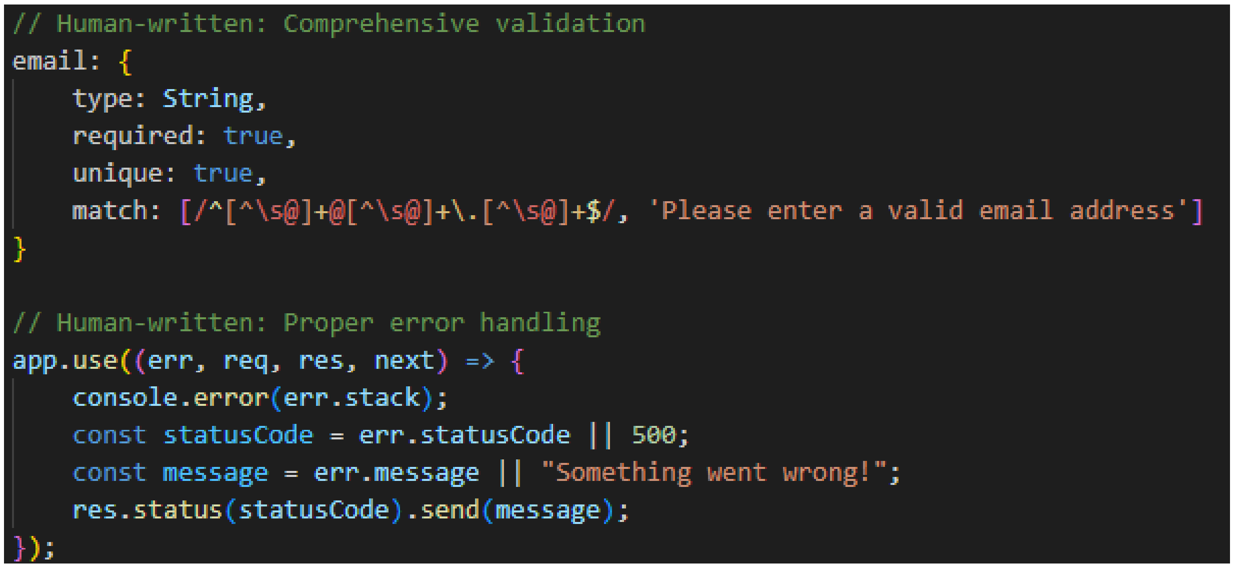
```javascript
const express = require('express');
const router = express.Router();
const Student = require('../models/Student');

// Get all students
router.get('/', async (req, res) => {
  try {
    const students = await Student.find();
    res.json(students);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Get a single student
router.get('/:id', async (req, res) => {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) return res.status(404).json({ error: 'Student not found' });
    res.json(student);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Create a student
router.post('/', async (req, res) => {
  try {
    const student = new Student(req.body);
    await student.save();
    res.status(201).json(student);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// Update a student
router.put('/:id', async (req, res) => {
  try {
    const student = await Student.findByIdAndUpdate(req.params.id, req.body, { new: true, runValidators: true });
    if (!student) return res.status(404).json({ error: 'Student not found' });
    res.json(student);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

// Delete a student
router.delete('/:id', async (req, res) => {
  try {
    const student = await Student.findByIdAndDelete(req.params.id);
    if (!student) return res.status(404).json({ error: 'Student not found' });
    res.json({ message: 'Student deleted' });
  } catch (err) {
    res.status(500).json({ error: err.message });
```

Fig 2. AI-Generated Student Routes Snippet

AI-Generated API: Pass Level (72%)

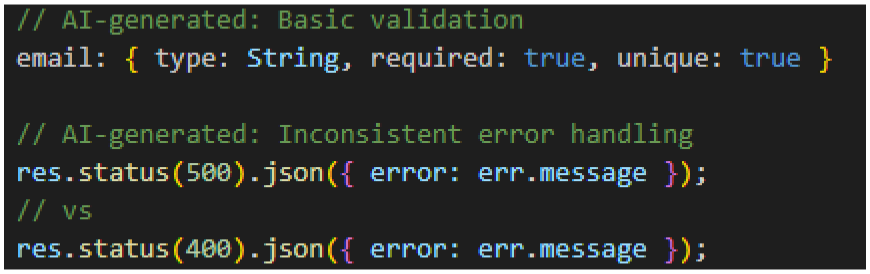
- **Works, but Barebones**: Functional CRUD routes with basic Express structure.
- **Security Gaps**: No protection middleware or input sanitization.
- **Inconsistent & Dated**: Uses CommonJS, lacks centralized error handling, and has minimal validation.
- **Docs Lack Depth**: Basic setup and endpoint listings, but missing real-world guidance.

Comparing the human-generated schema with the AI-generated one, we can surmise the following:



```
// Human-written: Comprehensive validation
email: {
    type: String,
    required: true,
    unique: true,
    match: [/^[^\s@]+@[^\s@]+\.[^\s@]+$/, 'Please enter a valid email address']
}

// Human-written: Proper error handling
app.use((err, req, res, next) => {
    console.error(err.stack);
    const statusCode = err.statusCode || 500;
    const message = err.message || "Something went wrong!";
    res.status(statusCode).send(message);
});
```

Fig 3. Human-written snippet



```
// AI-generated: Basic validation
email: { type: String, required: true, unique: true }

// AI-generated: Inconsistent error handling
res.status(500).json({ error: err.message });
// vs
res.status(400).json({ error: err.message });
```

Fig 4. Ai-generated snippet

Key Findings and Recommendations

1. Security Gap
   The AI-generated code lacks essential security measures, making it unsuitable for production use without significant modifications

2. Code Quality Disparity
   The human-written code demonstrates significantly better practices in validation, error handling, and modern JavaScript usage.

3. Maintainability Concerns
   AI-generated code would be difficult to maintain and extend in a team environment without human intervention

4. Documentation Excellence Gap
   The human-written project demonstrates exceptional documentation quality with a comprehensive README, user-friendly setup instructions, and practical examples, while the AI-generated project provides only basic functional documentation.

5. Development Experience
   The human-written project provides a much better developer experience with proper tooling, scripts, and comprehensive documentation that guides users through setup, usage, and troubleshooting.

Conclusion
This comparison demonstrates that while AI tools can generate functional code quickly, they often produce solutions that lack the depth, security, maintainability, and comprehensive documentation required for professional software development. The human-written project shows superior understanding of modern development practices, security considerations, code quality standards, and user experience design. (ChatGPT, 2025)

Justification for Tools & Languages

I chose JavaScript and Node.js because they are widely used for backend development and offer fast prototyping for small apps. Express.js was selected for its popularity and simplicity in setting up RESTful APIs. This tech also mirrors what AI tools like Copilot and ChatGPT are trained on, making the comparison more valid.

According to a 2024 Stack Overflow Developer Survey (Stack Overflow, 2024), JavaScript continues to be the most commonly used language worldwide, and tools like Copilot are often benchmarked using JavaScript-based examples. This made it the ideal language to highlight the differences in how humans and AI approach code structure and clarity.

Suggestions for Future Improvement

This project scratched the surface of comparing human-written vs. AI-generated code, but if I had more time, I would have loved to push things further.

First, I would add user authentication. Security is a critical part of any real-world application, and testing how both the AI and I handled things like password hashing, session management, or JWT tokens would be a great way to evaluate whether AI can write secure, robust logic or if it simply mimics insecure patterns from its training data.

Like any student project, there were a few avoidable limitations:

- Time constraints meant I had to keep the scope narrow - just a basic CRUD app.
- The task itself was relatively simple. While that made it easier to compare, it also meant the AI wasn't really pushed to handle more abstract or complex logic.
- I only used one AI program (Cursor). Trying another AI Programming Application could have provided more well-rounded insights into how different tools handle code generation.

Despite these limits, the exercise was still eye-opening, especially in terms of understanding where AI can genuinely help, and where it can fall short.

Some Advice for Other Developers Using AI Tools

If you're just starting with AI-assisted coding, here's what I'd say:

Understand the tools
Even if the AI gives you a working solution, don't blindly accept it. Read through every line. Try to break it. Try to fix it. If you can't explain what it's doing, you don't own that code.

Refactor and improve
Use AI-generated code as a rough draft. Clean it up. Make it your own. Add comments, rename variables, and split it into reusable modules. That's how you turn generated code into *good* code.

Give credit where it's due.
If a tool helped you, acknowledge it, whether that's in your documentation or project write-up. Transparency builds trust.

Don’t shortcut your learning
AI can be a helpful assistant, but it can’t replace the value of struggling through a problem, getting stuck, and finally figuring it out. That process builds real developer skills, the kind no model can give you.

In short, AI is a tool, not a replacement for thinking. If used thoughtfully, it can boost your productivity and even help you learn. But if you rely on it too heavily, you risk becoming a passive copy-paster rather than an active problem-solver. And in this industry, *understanding* always trumps *speed.*

References

Barr, A. (2025) *AI coding tools upend the 'buy versus build' software equation and threaten the SaaS business model*. Business Insider. Available at: https://www.businessinsider.com/ai-coding-tools-buy-versus-build-software-saas-netlify-bolt-2025-6africa.businessinsider.com+6businessinsider.com+6businessinsider.com+6 (Accessed: 2 July 2025).

BytePlus Editorial Team (2025) *Ethical concerns about AI code generation: Navigating the moral landscape of technological innovation*. Available at: https://www.byteplus.com/en/topic/381416?title=ethical-concerns-about-ai-code-generation-navigating-the-moral-landscape-of-technological-innovation (Accessed: 2 July 2025).

Davis, T. and Rajamanickam, S. (2022) *Ethical concerns of code generation through artificial intelligence*. *SIAM News*, 55(10). Available at: https://www.siam.org/publications/siam-news/articles/ethical-concerns-of-code-generation-through-artificial-intelligence/siam.org (Accessed: 2 July 2025).

GitClear (2024) *Coding on Copilot: 2023 Data Suggests Downward Pressure on Code Quality (Including 2024 Projections)*. Available at: https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality (Accessed: 4 July 2025).

Sanj.dev (2025) *Ethical Considerations in AI Code Generation*. Available at: https://sanj.dev/post/ethical-ai-code-generationsanj.dev (Accessed: 4 July 2025).

Stack Overflow (2024) *Stack Overflow Developer Survey 2024 – Technology*. Available at: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language-prof (Accessed: 4 July 2025).