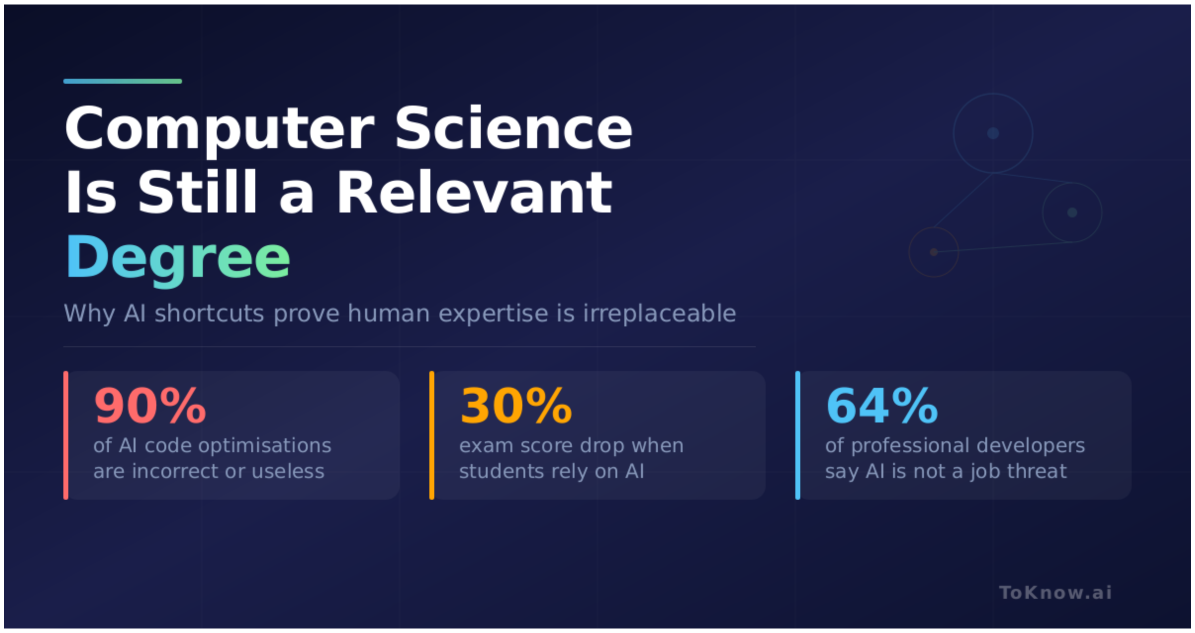
Computer Science Is Still a Relevant Degree

Kabui, Charles

2026-03-02

Table of Contents



Computer Science Is Still a Relevant Degree

Why AI shortcuts prove human expertise is irreplaceable

90% of AI code optimisations are incorrect or useless

30% exam score drop when students rely on AI

64% of professional developers say AI is not a job threat

ToKnow.ai

Introduction

Every few months, a new headline declares that AI has made the computer science degree obsolete. “Just prompt an LLM,” the argument goes. “Anyone can code now.”

The numbers say otherwise. The 2025 Stack Overflow Developer Survey found that **74% of professional developers hold at least a bachelor’s degree**, and **64% do not see AI as a threat to their job.** More telling: **46% of developers actively distrust the accuracy of AI tools**, while only 3% say they highly trust AI output. Two thirds (66%) report frustration with AI solutions that are “almost right, but not quite.” These trends have held: the 2024 survey reported similar degree rates (66%) and even higher confidence (70% seeing no threat).

Why? Because a growing body of research shows that LLMs cut corners when generating code. They default to the simplest, most common answer from their training data instead of reasoning toward the correct, efficient, or secure solution. They fail on exactly the kinds of problems a computer science education teaches you to solve.

---

1. How LLMs Cut Corners in Code

An LLM does not understand code the way a programmer does. It predicts the most probable next word based on patterns from billions of lines of existing code. When you ask it to write a sorting function, it is not reasoning about algorithmic complexity. It is recalling what sorting functions usually look like.

Sanyal et al. (2025) formalised this in *"Policy Optimization Prefers The Path of Least Resistance"* (arXiv:2510.21853). They found that when LLMs are trained with reinforcement learning, **they consistently abandon reasoning in favour of the fastest route to a reward**, even when reasoning is given 4x the reward weight. If a shortcut exists, the model learns to take it.

This plays out in practice. Codeflash (2025) tested leading LLMs on over 100,000 real-world function optimisations and found that **90% of suggestions were either incorrect or useless**: 62% introduced bugs, and most of the rest offered negligible improvement. Optimising code requires understanding trade-offs, runtime behaviour, and language-specific details that LLMs simply do not have.

Catir et al. (2025) confirmed this pattern in an academic setting (arXiv:2504.14964). LLMs solved introductory programming assignments easily but **struggled with second- and third-year CS problems** requiring algorithm design and multi-step reasoning. The models could identify the general approach but rarely produced correct, complete solutions for harder problems.

The effect shows up in classrooms too. Ding (2025) studied 117 university students and found that those using AI assistance scored near-perfectly on isolated homework problems but saw a **30% performance drop on proctored exams** (arXiv:2512.10758). The LLM handles templated problems well, but falls short when genuine understanding is required.

---

2. Why This Happens

These failures are not random. They come from well-documented properties of how neural networks learn.

Pattern matching over understanding. Geirhos et al. (2020) showed in *Nature Machine Intelligence* (arXiv:2004.07780) that neural networks latch onto surface-level patterns rather than learning the underlying concept. In code, this means an LLM learns that `sort_list` functions usually contain `sorted()`, but does not consider whether the problem needs a stable sort, a custom comparator, or an approach suited for nearly-sorted data.

Extreme preference for simple features. Shah et al. (2020) demonstrated at NeurIPS (arXiv:2006.07710) that neural networks can **rely exclusively on the simplest feature** in the data and ignore every other signal, even more accurate ones. For code generation, this means LLMs learn formatting, naming conventions, and common library calls thoroughly, while learning algorithmic correctness and edge case handling poorly.

One-directional learning. Berglund et al. (2023) showed that LLMs trained on “A is B” cannot answer “what is B?” (arXiv:2309.12288). In coding terms, if the model has seen many examples of using `requests.get()` to fetch URLs, it handles that direction well. But ask it to work backwards from observed behaviour and it struggles, because it never learned the reverse relationship.

---

3. Right Answers for Wrong Reasons

One of the most concerning findings is that LLMs can produce correct code without actually understanding it.

Orvalho and Kwiatkowska (2025) applied cosmetic changes to programs, such as renaming variables, swapping if-else branches, and converting for-loops to while-loops, that do not change what the program does (arXiv:2505.10443). **LLMs produced correct predictions based on flawed reasoning in 10% to 50% of cases**, and frequently changed their answers in response to these superficial changes.

Haroon et al. (2025) found the same problem with bug detection (arXiv:2504.04372). Across 750,013 tasks, **cosmetic code changes caused LLMs to fail on bugs they had previously found in 78% of cases.** The model's accuracy also depended on where the relevant code appeared in the input: earlier was better, regardless of logic.

Jain et al. (2024) showed that LLMs score significantly better on coding problems published before their training cutoff (arXiv:2403.07974). Much of what looks like reasoning is actually recall. The model memorised solutions rather than deriving them.

---

4. The Security Problem

The corner-cutting behaviour has direct consequences for security. Perry et al. (2023) conducted the first large-scale study on this question (arXiv:2211.03622) and found that developers with AI assistance **wrote less secure code** than those without, and were **more confident** that their code was secure. The LLM skips input validation, error handling, and security checks because they appear less frequently in training data. The developer trusts the output because it looks professional. Both are taking the easy path.

The 2025 Stack Overflow survey reinforces this: AI integration ranks second to last (9th of 10) among the factors that make developers choose a technology, while security and privacy concerns rank first among reasons to reject one.

5. What Works

The research also points to strategies that push LLMs past their default behaviour.

Self-debugging. Chen et al. (2023) showed that asking LLMs to explain their code and check it against test results improved accuracy by up to 12% (arXiv:2304.05128). Forcing the model to evaluate its own output catches many of the shortcuts.

Step-by-step verification. Lightman et al. (2023) at OpenAI demonstrated that rewarding correct intermediate steps, not just the final answer, raised problem-solving accuracy to 78% on a challenging maths dataset (arXiv:2305.20050). This makes shortcuts unrewarding because the model cannot get credit for a right answer reached the wrong way.

Better prompting. Specifying constraints ("write an in-place, stable sort optimised for nearly-sorted input with error handling"), asking the model to think step by step, and requesting it to identify what inputs would break its own code all push it beyond its default templates.

Verification over generation. The most reliable approach combines LLM-generated code as a starting point with automated testing, benchmarking, and human review. LLMs generate plausible code but cannot test it. That gap is where CS knowledge matters most.

6. Conclusion

The evidence is consistent across multiple independent research groups: **LLMs cut corners when writing code.** This is not a temporary limitation that bigger models will fix. It is a property of how these systems learn.

This is why a CS degree still matters. It builds the skills that AI cannot replicate:

- **Algorithmic reasoning.** LLMs fail on advanced CS problems that require understanding data structures, complexity, and multi-step logic.
- **Security awareness.** AI-assisted developers write less secure code and are more confident about it. Understanding threat models and defensive programming requires training that LLMs lack.
- **Verification.** LLMs generate plausible code but cannot test it. The ability to verify correctness, benchmark performance, and reason about edge cases is a core CS skill.
- **Understanding why, not just what.** LLMs produce correct answers based on flawed reasoning up to 50% of the time. A CS education teaches you why solutions work, so you can evaluate AI output rather than just accept it.

The developers who succeed in an AI-augmented world will be those who understand computer science deeply enough to catch the shortcuts, fix the edge cases, and build systems that LLMs cannot design on their own.

AI coding tools are powerful assistants. But an assistant that keeps cutting corners needs someone who knows the right way. That is what a CS education provides.

References

1. Sanyal, D., et al. (2025). *Policy Optimization Prefers The Path of Least Resistance.* arXiv:2510.21853.
2. Misra, S. (2025). *LLMs Struggle to Write Performant Code.* Codeflash.
3. Catir, E., et al. (2025). *Evaluating Code Generation of LLMs in Advanced Computer Science Problems.* arXiv:2504.14964.
4. Ding, K. (2025). *Designing AI-Resilient Assessments Using Interconnected Problems.* arXiv:2512.10758.
5. Geirhos, R., et al. (2020). *Shortcut Learning in Deep Neural Networks.* Nature Machine Intelligence.
6. Shah, H., et al. (2020). *The Pitfalls of Simplicity Bias in Neural Networks.* NeurIPS 2020.
7. Orvalho, P. & Kwiatkowska, M. (2025). *Are LLMs Robust in Understanding Code Against Semantics-Preserving Mutations?* arXiv:2505.10443.
8. Haroon, S., et al. (2025). *Assessing the Impact of Code Changes on the Fault Localizability of LLMs.* arXiv:2504.04372.
9. Jain, N., et al. (2024). *LiveCodeBench: Holistic and Contamination Free Evaluation of LLMs for Code.* arXiv:2403.07974.
10. Perry, N., et al. (2023). *Do Users Write More Insecure Code with AI Assistants?* ACM CCS 2023.
11. Berglund, L., et al. (2023). *The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A".* arXiv:2309.12288.
12. Chen, X., et al. (2023). *Teaching Large Language Models to Self-Debug.* arXiv:2304.05128.
13. Lightman, H., et al. (2023). *Let's Verify Step by Step.* arXiv:2305.20050.
