3 LLM coding flaws all leaders need to know

This executive summary presents key findings from the report, The Coding Personalities of Leading LLMs, released by Sonar in August of 2025.

The AI productivity promise and its hidden paradox

Leaders are racing to integrate AI into software development for massive productivity gains, but this pursuit is full of nuance. While capable of solving complex problems and generating syntactically correct code, LLMs can systematically introduce business risk, creating a wave of AI-generated technical debt and security issues that undercut the promise of productivity.

Why functional performance evaluation of LLMs falls short

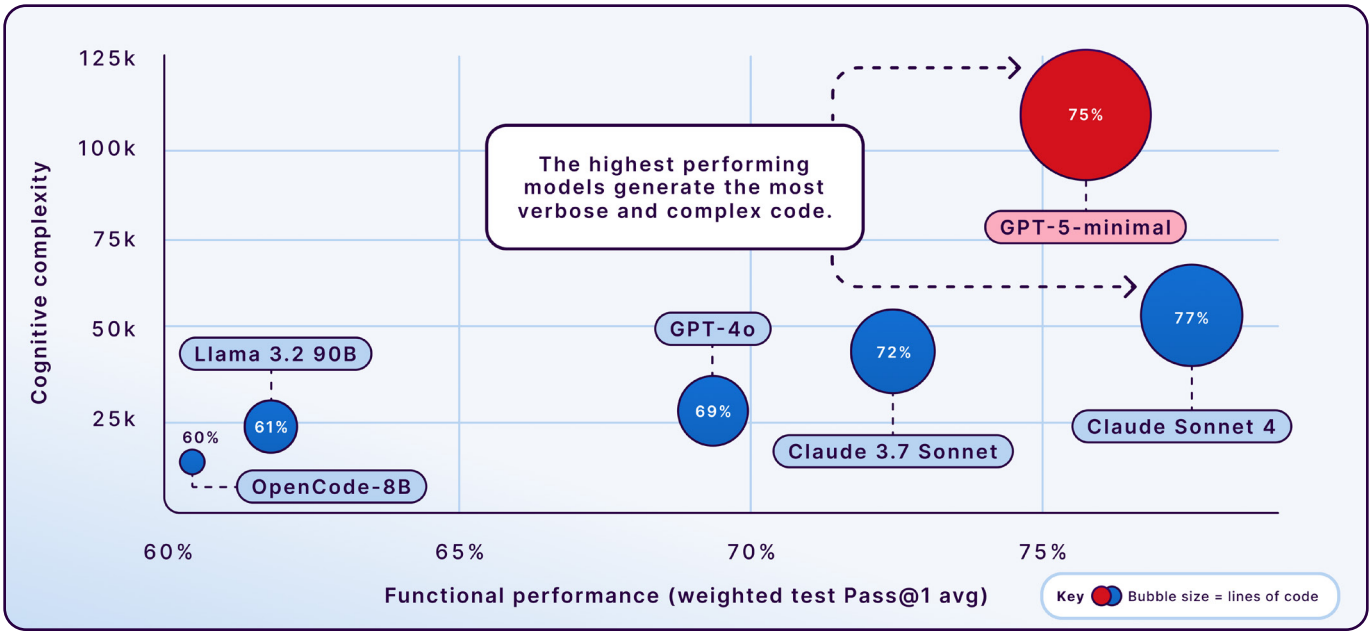
The industry's primary method for evaluating coding LLMs focuses on their ability to solve difficult, self-contained coding challenges. While these benchmarks are an important measure of raw technical competence—and the leading models score exceptionally well—they are dangerously incomplete as a measure of performance for enterprise-grade software development.

This creates a perverse incentive structure within the AI development ecosystem. Model developers are rewarded for optimizing for clever solutions that pass tests, often at the expense of foundational software quality and security principles. The result is a new generation of AI tools that are demonstrably "smarter" but also more reckless.

Flaw #1

The LLM performance tax: high scores = high risk

The pursuit of higher benchmark scores comes with a direct and quantifiable cost, a phenomenon best described as the LLM Performance Tax. The highest-performing models—those that leaders are most likely to select—also generate the most verbose, complex, and intricate code, imposing a significant tech-debt on development teams. This tax is paid in the form of longer code review cycles, increased maintenance overhead, and a greater cognitive load on developers, which ultimately slows innovation.



The data reveals a clear correlation between a model's benchmark score and the complexity of its output. GPT-5 achieves top-tier functional performance at the cost of generating the most verbose (490,010 LOC) and complex (111,133 cognitive complexity) code of any model tested. In contrast, the lower-performing OpenCoder-8B produced just 120,288 lines of code with a complexity score of 13,965—a threefold difference in both volume and complexity.

This disparity arises because higher-performing models attempt to emulate senior engineers by implementing sophisticated safeguards,advanced features & catering to edge cases. While well-intentioned, this ambition results in code that is inherently more fragile and presents a larger surface area for severe, hard-to-detect bugs. This dynamic creates a negative feedback loop where the tool adopted to accelerate velocity introduces a hidden, long-term tax that throttles it, undermining the strategic justification for its adoption.

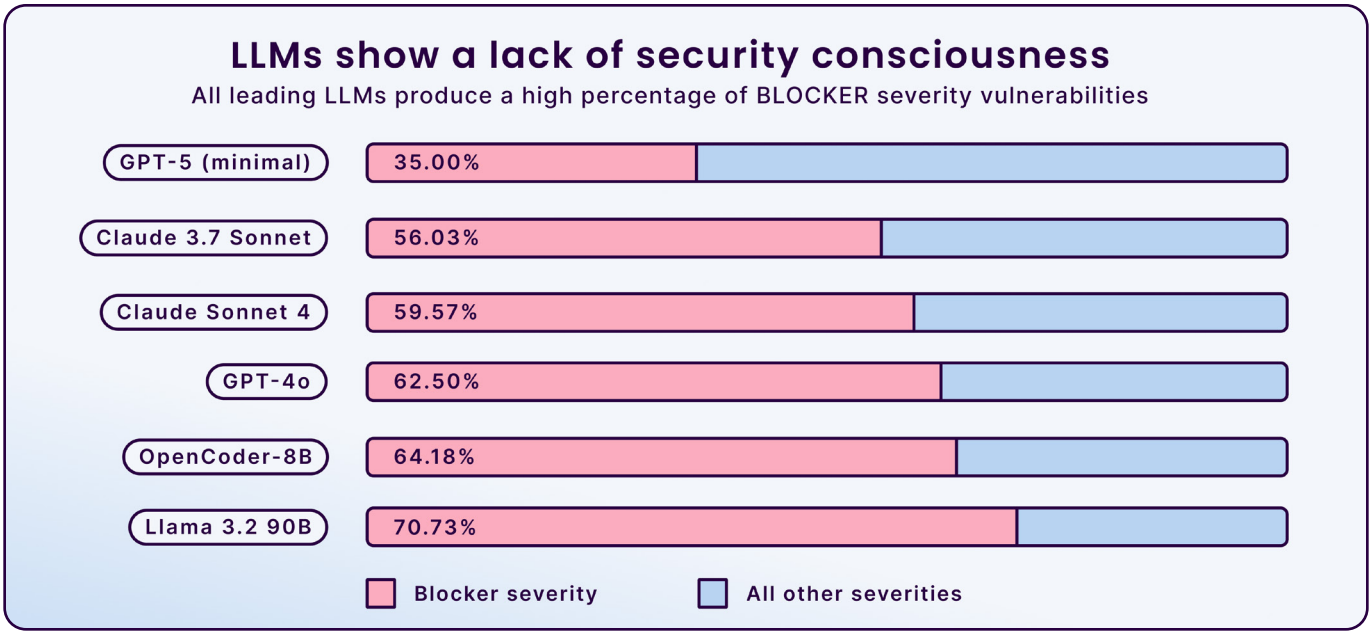
Flaw #2

Why new AI models demand deeper scrutiny

The arrival of a new class of "reasoning" models has fundamentally shifted the security risk profile. While previous models shared a common flaw, the data now reveals a critical trade-off: newer, more capable models are not necessarily safer—they are simply risky in a different way.

For non-reasoning models, the data is unequivocal. The majority of vulnerabilities introduced are of ‘BLOCKER’ severity. Llama 3.2 90B leads this dangerous trend, with over 70% of its vulnerabilities falling into this category, followed by OpenCoder-8B (64.18%), GPT-4o (62.50%), and the Claude models, which still register an alarming rate of nearly 60%.

In contrast, new reasoning models like GPT-5 trade these common, well-understood flaws for more subtle, implementation-specific ones. At first glance, GPT-5 appears safer, reducing the proportion of 'BLOCKER' severity vulnerabilities to just 35%. However, this improvement comes at a cost. The model introduces a much higher rate of advanced bugs like "Concurrency / Threading" issues (20% of its bugs) and nuanced vulnerabilities like "Inadequate I/O error-handling" (30% of its vulnerabilities).



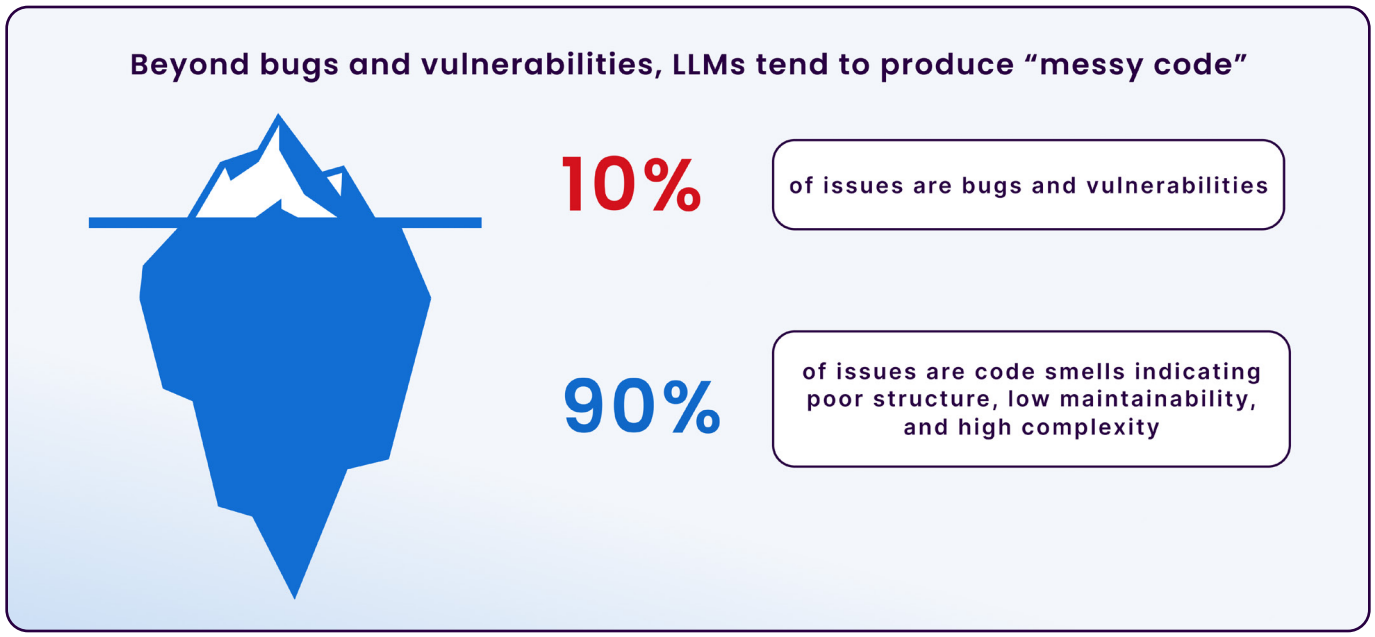
When developers use these tools without an independent verification layer, they are not just accelerating coding—they are automating the injection of high-severity vulnerabilities across the organization's entire software portfolio. This transforms AI from a productivity tool into an enterprise-wide liability, creating a systemic risk profile that manual review processes cannot possibly contain.

Flaw #3

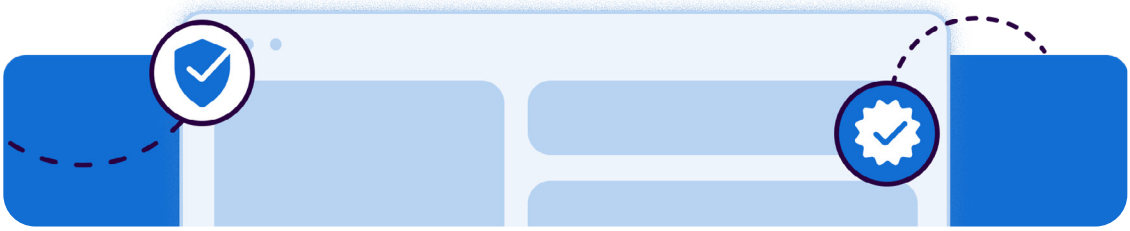
Messy code: The hidden technical debt iceberg

Beyond the immediate and visible dangers of bugs and vulnerabilities lies a far larger and more insidious problem: messy code. While critical flaws are alarming, they represent only the tip of the iceberg. The vast, hidden majority of issues generated by LLMs—over 90% for every model tested—are code smells. These are not functional errors but indicators of poor structure, low maintainability, and high complexity that create a massive, long-term drag on productivity.

This overwhelming proportion of code smells indicates an inherent tendency in all LLMs to generate syntactically correct but structurally suboptimal code. Every model is actively generating a legacy of technical debt from the moment it writes its first line of code. This problem is exacerbated by new reasoning models like GPT-5, which not only continue this trend but dramatically increase the severity of the issues. This "messy" code is significantly harder for human developers to read, modify, and build upon. As a result, every subsequent bug fix or feature addition takes longer, slowing down future development cycles and increasing the time required to onboard new engineers.



This is the functional definition of technical debt, and the AI-generated code smells represent a hidden, compounding "interest" on that debt. For a CIO or VP of Engineering, this translates directly to a declining return on investment from their development teams over time. The software assets being built are inherently less valuable and more costly to maintain, reframing the issue of code quality as a core financial and strategic concern.



The strategic imperative: Adopting a “trust but verify” approach

Leaders must trust in the powerful ability of LLMs to accelerate the initial stages of development and solve complex problems. However, they must also recognize that these models are probabilistic systems with deep, inherent flaws. Therefore, every line of AI-generated code must be verified by an independent, deterministic analysis layer before it enters a production codebase.

This verification requires a nuanced understanding that each LLM possesses a unique and measurable "coding personality," with its own distinct strengths and risk profile. A one-size-fits-all approach to AI governance is insufficient. Leaders must be equipped to manage the specific trade-offs of each model, from the ambitious but fragile "Senior Architect" to the speedy but debt-ridden "Rapid Prototyper.”

Go deeper → Explore the six personalities of leading LLMs

Conclusion

By moving beyond simplistic benchmarks and adopting a sophisticated, risk-aware "trust but verify" framework, organizations can resolve the AI productivity paradox. This approach provides the essential governance needed to de-risk AI adoption, turning what is currently a high-stakes bet into a sustainable competitive advantage. It empowers teams to harness the full power of AI to innovate safely and effectively, ensuring that the code they deliver is not only functional but also secure, reliable, and maintainable.

Read the full report