Reading is the New Writing

Why the AI Era Demands Absolute Mastery of the Basics

CONF42 LLMS · 2026

The State of Play in 2026

LLMs have turned raw code syntax into a commodity. What used to take hours now takes seconds — but the engineering bar has not lowered. It has risen.

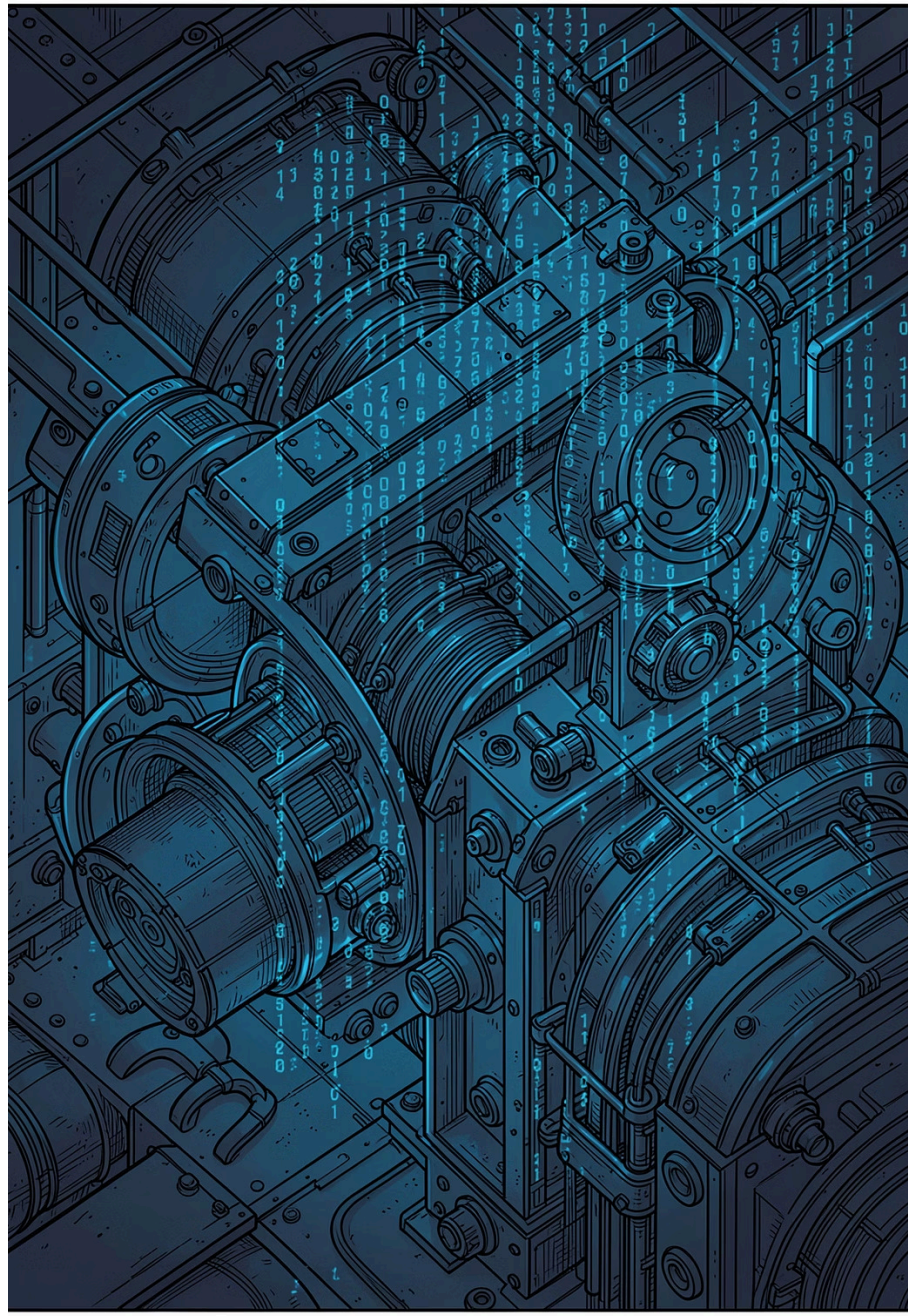
Before AI

Hours spent writing boilerplate, scaffolding, and repetitive patterns. Output speed was a proxy for skill.

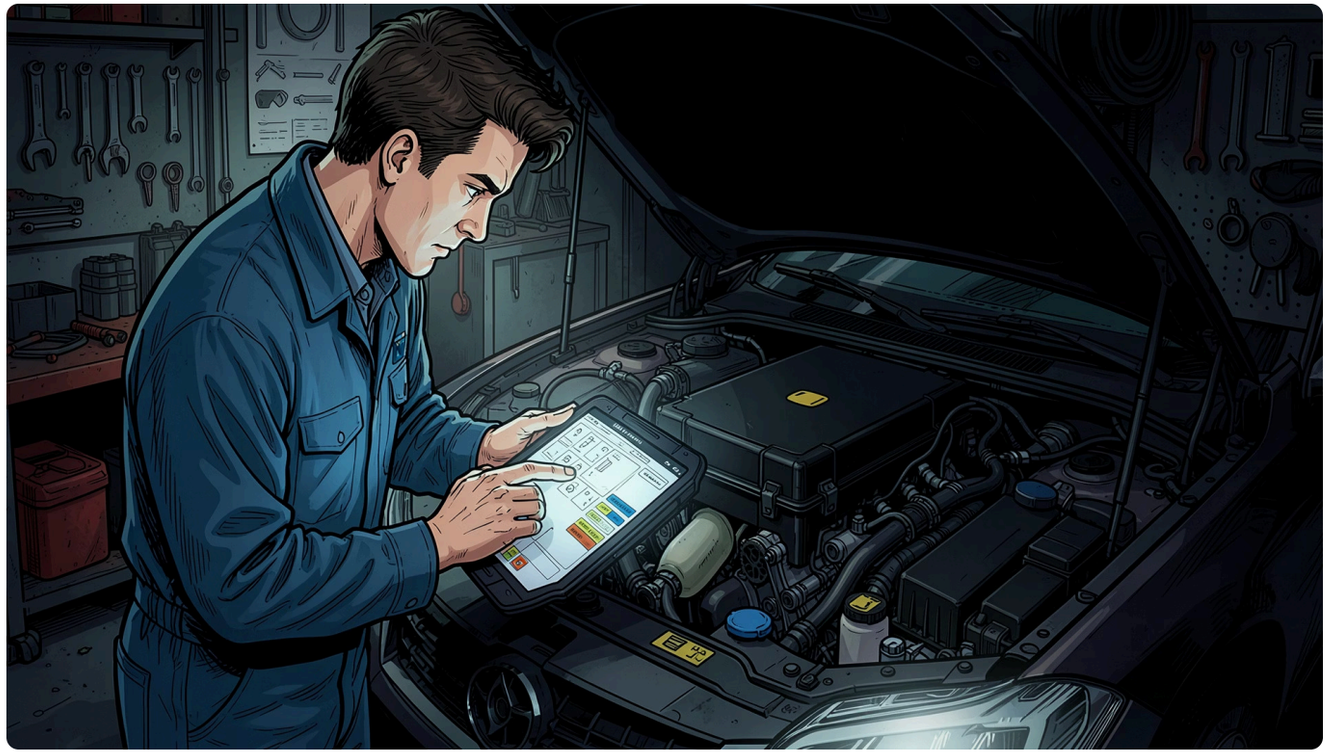
Now

Seconds spent writing a prompt. The LLM generates the code. The real work begins when it stops.

Raw coding output is no longer the metric of a software engineer.



The Threat: The Black Box & The Delusion of Productivity



We mistake **"compiled and working"** for **"good engineering."** Shipping code you didn't write — and don't fully understand — is a liability, not a win.

Cargo Culting AI Output

Patterns that look right but embed subtle anti-patterns, security gaps, or inefficient logic.

Blind Trust at Scale

Every accepted suggestion compounds technical debt you didn't choose and may not see until production.

Code is for Reading, Not Just Writing

We now spend **~90% of our time** auditing, debugging, and maintaining what the model generated. Reading code is the primary engineering act.

1 Skim & Accept

Looks correct at a glance.

Hidden assumptions, edge cases, and subtle bugs slip through.

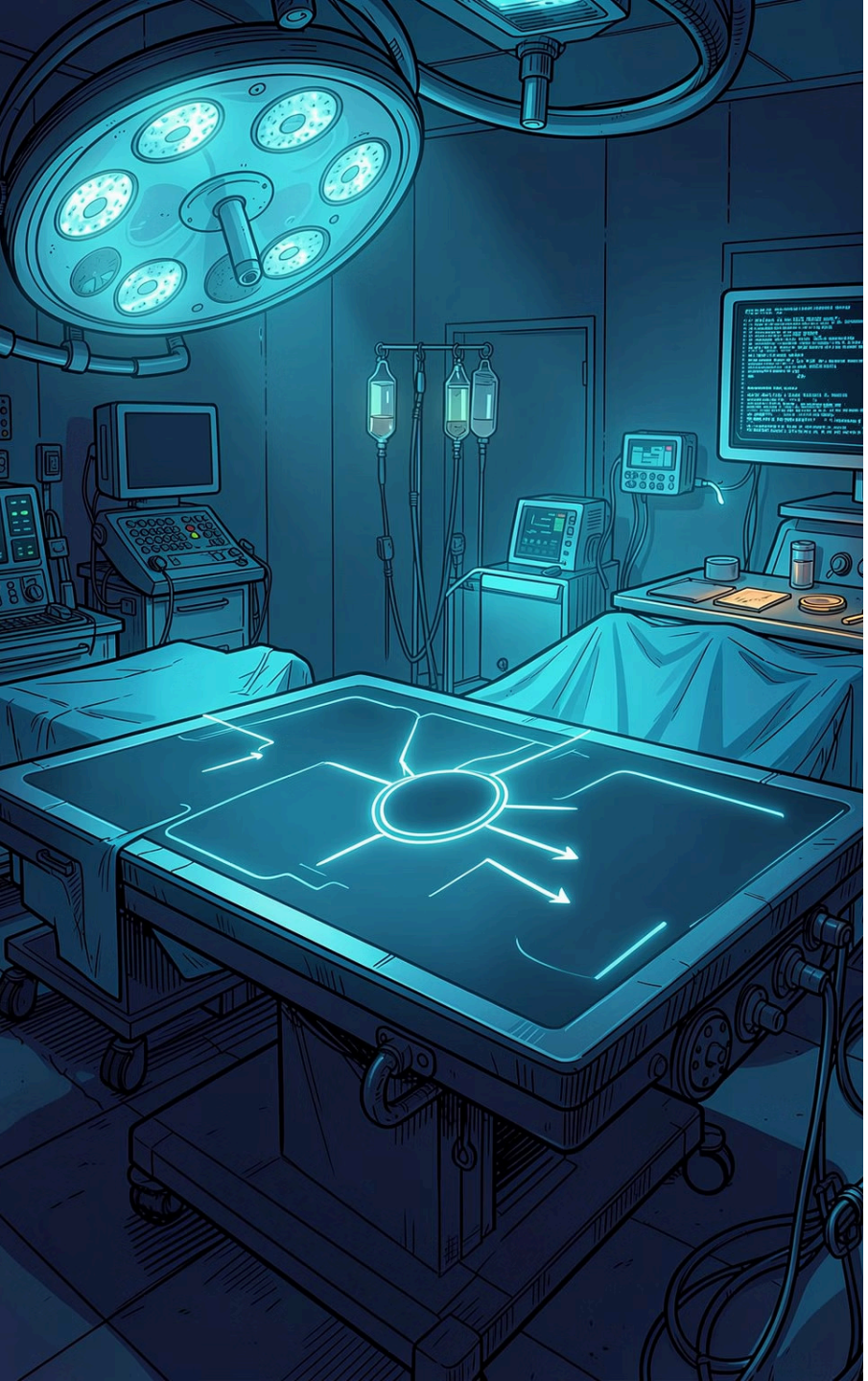
2 Read & Verify

Trace the logic line by line.

Test the behavior, question the output, and understand the tradeoffs.

> ⚠ **If you can't understand code deeply enough to catch what the LLM missed — security flaws, performance issues, or architectural misalignment — you cannot responsibly own the codebase.**

The LLM writes. You must understand, validate, and own every line that ships.



Core Argument 1: The "Autopsy" Problem

When the system fails under load, you are performing an autopsy on logic you didn't write. Relying blindly on probabilistic code generation completely bypasses the developer's internal mental model of the system. Because you didn't actively architect it, the intuition for debugging is entirely absent.

You Didn't Write It

The mental model for debugging is absent. You're reverse-engineering intent from output.

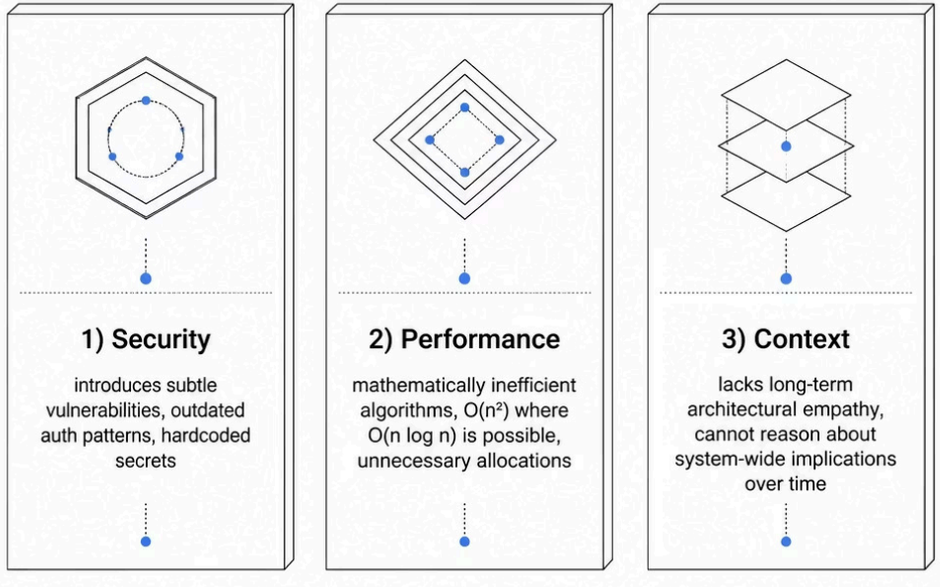
It Fails Unexpectedly

Edge cases, race conditions, and resource leaks surface only in production — under real load.

You Own the Outcome

The on-call pager doesn't care who wrote the code. The incident report has your name on it.

Core Argument 2: Human Fundamentals as the Security & Scale Guardrail



LLMs are pattern matchers, not engineers. They lack **contextual empathy** for your system's long-term trajectory.

- Subtle security vulnerabilities slip through undetected
- Outdated library versions and deprecated APIs
- Algorithmically inefficient logic that scales poorly
- No awareness of your team's architectural constraints

Foundational mastery is the filter that catches what the model misses.

Defining "The Basics" in an AI World

It's not about memorizing syntax. It's about the **Computer Science bedrock** that lets you evaluate any code, regardless of who — or what — wrote it.

Data Structures & Algorithmic Complexity

Understanding O(n) trade-offs, memory layout, and why they matter at scale.

Concurrency & Memory Management

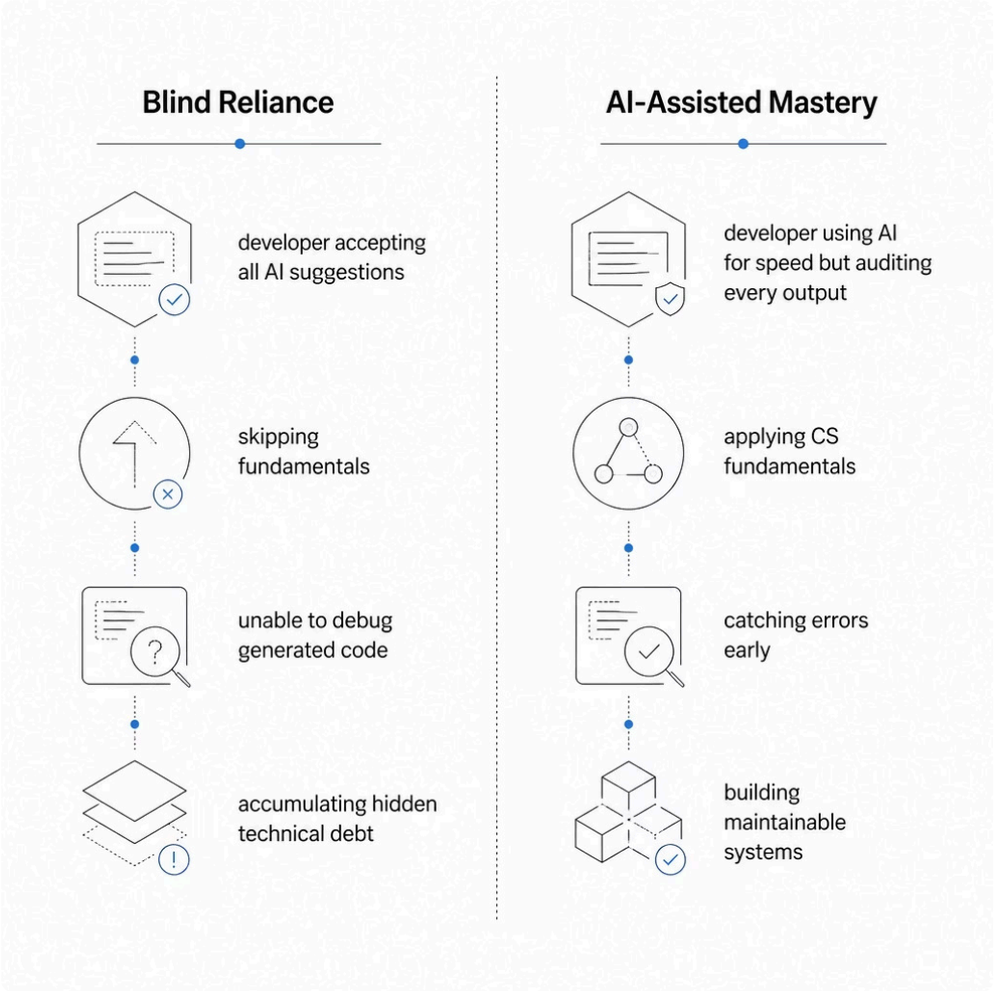
Race conditions, deadlocks, garbage collection — the invisible bugs LLMs introduce.

System Design & Secure Coding

Architectural reasoning, threat modeling, and the principles that outlast any framework.

The Dangers of Mental Muscle Atrophy

Using AI to *escape* learning the fundamentals doesn't make you faster — it makes you **dependent.** Problem-solving is a cognitive muscle. Stop exercising it, and it weakens.



The Real Risk

Every time you accept AI output without understanding it, you're trading short-term speed for long-term capability.

- Weakened debugging intuition
- Inability to reason about unfamiliar code
- Growing dependency on the tool

The Modern Engineer as the Managing Editor

The career path is shifting: from **code typist** to **system reviewer.** AI amplifies leverage — but only for those who know how to code *well.*

Audit

Read every line the model produces with a critical, informed eye.

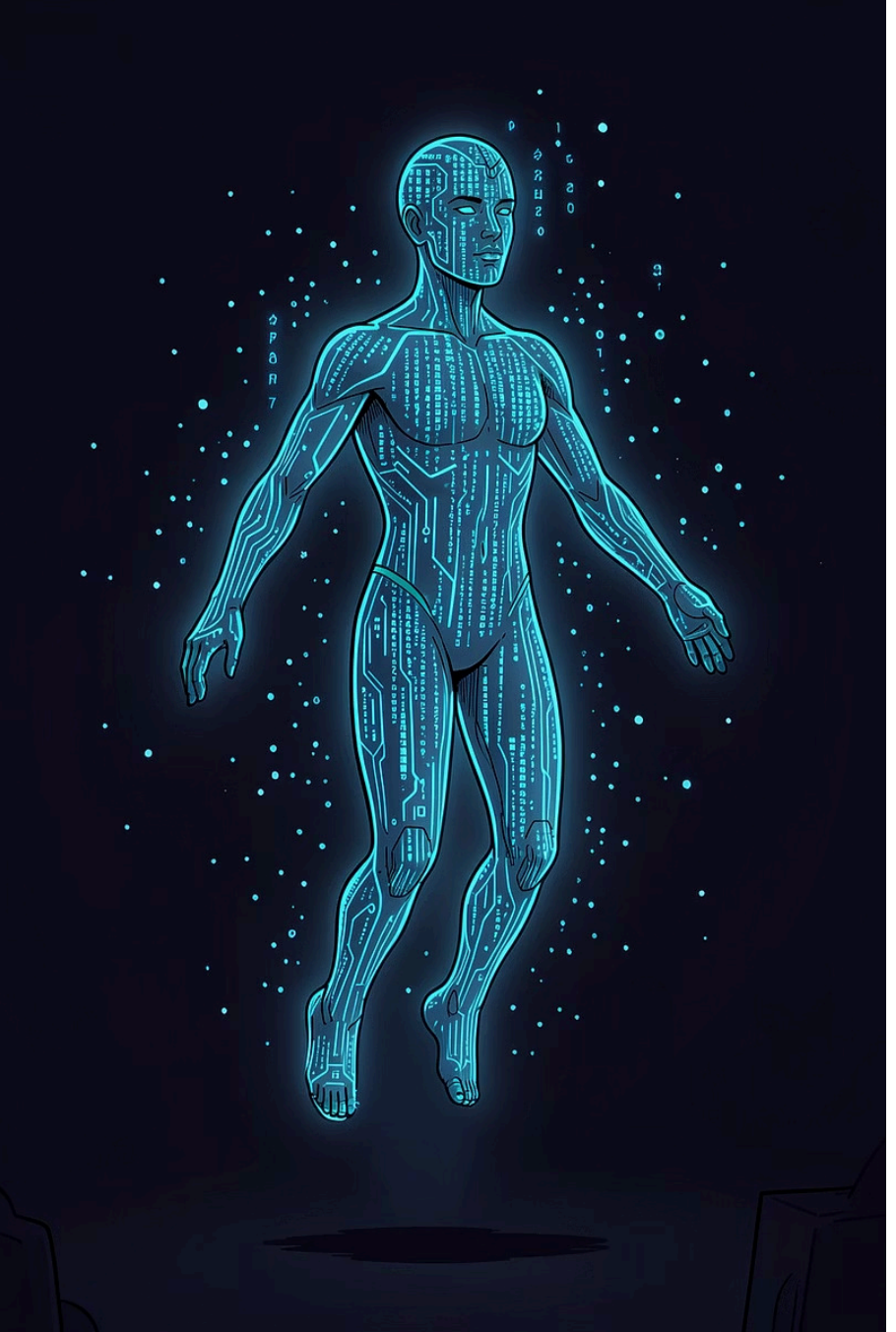
Secure

Apply security fundamentals the model doesn't inherently possess.

Scale

Design systems that hold up under real-world load — because you understand why.

AI does not replace the need to know the fundamentals — it makes knowing them your primary differentiator.



> "Don't just use AI to write faster. Use your mastery to audit, secure, and scale better."

Thank You

Conf42 LLMs · 2026

Connect

LinkedIn: Petras Bazdaras

Key Takeaway

Reading code is the highest-leverage skill in the AI era. Master it.