Sonatype Finds AI Coding Assistants Hallucinate 27.75% of Package Upgrades—10,000+ Non-Existent Versions Recommended

Your AI coding assistant just recommended 10,000 software packages that don’t exist. And it was confident about half of them.

The Numbers That Should Keep You Up Tonight

Sonatype’s 2026 State of the Software Supply Chain Report, released January 28, delivers the most comprehensive audit of AI-assisted dependency management to date. The headline figure: leading LLMs—including GPT-5—hallucinated 27.75% of dependency upgrade recommendations across 36,870 real-world enterprise upgrades analyzed.

That’s not a rounding error. That’s more than one in four recommendations pointing developers toward package versions that simply don’t exist in any repository.

The raw numbers are staggering. Over 10,000 non-existent package versions were recommended by AI coding assistants during the study period. These weren't edge cases or obscure libraries. The analysis covered the major ecosystems: Maven Central, PyPI, npm, and NuGet—repositories that collectively saw 9.8 trillion downloads in 2025, representing 67% year-over-year growth.

The hallucinations weren't limited to phantom version numbers. The report documents AI recommendations that pointed to protestware, hijacked packages, and packages containing known malicious components. The AI didn't just make things up—it occasionally made things up in ways that could actively compromise your infrastructure.

The Confidence Paradox

Here's where the data gets genuinely alarming for anyone building AI-assisted developer tooling: GPT-5 achieved 98% accuracy on high-confidence recommendations. That sounds excellent until you read the next line. Only 3.68% of GPT-5's recommendations qualified as high-confidence.

Read that again. The AI was right 98% of the time when it was confident—but it was confident less than 4% of the time.

For medium-confidence recommendations, the hallucination rate climbed to 19.20%. For low-confidence recommendations, the model was wrong nearly half the time, with a 47.38% hallucination rate. In practical terms, every time the AI hedged with anything less than full certainty, you were essentially flipping a coin on whether the package it suggested actually existed.

> The problem isn't that AI coding assistants are wrong—it's that they're wrong while sounding authoritative. A 47% error rate with human-readable confidence signals is worse than a 50% error rate with obvious uncertainty markers.

This creates a perverse incentive structure. Developers learn to trust high-confidence AI outputs, but the AI generates high-confidence outputs so rarely that most interactions occur in the danger zone. The confidence calibration is technically accurate but practically useless.

Why Standard Training Can’t Solve This

Understanding why LLMs hallucinate package versions requires understanding how they process dependency information in the first place.

Large language models learn about software packages the same way they learn about everything else: by ingesting text. They’ve seen millions of Stack Overflow answers, GitHub issues, documentation pages, and blog posts mentioning package versions. From this corpus, they build probabilistic models of what package names and version numbers look like.

The problem is threefold.

First, package ecosystems change constantly. A model trained on data from six months ago has outdated information about current stable versions. A model trained yesterday has outdated information about packages released today. The SD Times coverage of the report notes that the average enterprise project updates dependencies monthly, meaning any static training data becomes stale almost immediately.

Second, version numbers follow patterns that are easy to extrapolate incorrectly. If a model has seen versions 2.3.1, 2.3.2, and 2.3.3 of a library, it’s trivially easy to hallucinate 2.3.4—which may not exist. The pattern-matching that makes LLMs useful for code completion becomes a liability when the patterns suggest nonexistent artifacts.

Third, package naming conventions create collision risks. The npm ecosystem alone contains over 2 million packages. Malicious actors routinely publish packages with names similar to popular libraries (typosquatting) or claim abandoned package names (namespace hijacking). When an LLM suggests “lodash-utils” instead of “lodash,” it might be hallucinating, or it might be directing you to a malicious package that actually exists.

The 1.233 million malicious packages detected by Sonatype—representing 75% growth since 2019—aren’t just a security statistic. They’re evidence that the attack surface for AI-assisted development is expanding faster than defenses can adapt.

The Grounded AI Benchmark

The most technically significant finding in the report isn't the hallucination rate—it's the control group.

Sonatype tested their own grounded AI approach against the same 36,870 upgrade scenarios. This system queries real-time repository data before generating recommendations, essentially giving the AI access to ground truth about what packages and versions actually exist.

The result: zero hallucinations on the identical dataset.

This isn't a minor improvement or a marketing claim that requires statistical squinting. It's a categorical difference. The grounded approach eliminated an entire class of errors by architecturally preventing the AI from recommending things that don't exist.

The implementation isn't complex in principle. Before the AI generates a recommendation, it queries the relevant package registry API to verify that the suggested package and version exist. The response is constrained to the actual possibility space rather than the probabilistic possibility space the model learned during training.

The latency cost is measurable but minor—a few hundred milliseconds per query. The accuracy improvement is absolute.

What Most Coverage Gets Wrong

The initial reaction to these findings will predictably split into two camps: "AI is unreliable, stop using it for anything critical" and "this is just growing pains, fine-tuning will fix it." Both responses miss the structural insight.

The 27.75% hallucination rate isn't evidence that AI coding assistants are bad at their jobs. It's evidence that we're deploying them with the wrong architecture.

Standard LLMs are generative systems. They produce outputs based on learned patterns without real-time access to external truth. Asking them to recommend package versions is like asking someone to recommend restaurants in a city they visited two years ago—they'll give you answers that were probably correct once,

but the landscape has changed.

The grounded AI approach isn't a different model. It's the same underlying capability with access to live data. The model still generates the recommendation; it just does so within constraints that prevent impossible outputs.

> The fix for AI hallucination in dependency management isn't better training—it's better architecture. You can't train away the problem of recommending packages that didn't exist when the training data was collected.

This distinction matters because it changes where engineering effort should go. Teams investing heavily in prompt engineering or fine-tuning to reduce hallucinations in package recommendations are solving the wrong problem. The solution isn't making the AI better at guessing—it's removing the need to guess.

The Underreported Risk: Malicious Package Injection

The security implications of AI hallucinations extend beyond broken builds.

When an AI recommends a package that doesn't exist, two outcomes are possible. In the benign case, the developer runs their install command, gets an error, and goes looking for the correct package. Annoying, but contained.

In the malicious case, an attacker has already claimed that package name. The developer runs their install command, and it succeeds—because the malicious package exists. The AI hallucinated a name that happened to match an attacker's trap.

This isn't theoretical. The report documents cases where AI recommendations pointed to known malicious packages. The 75% growth in open-source malware since 2019 creates an ever-expanding minefield of package names that look legitimate but contain harmful payloads.

The attack surface mathematics are unfavorable. If AI coding assistants recommend 10,000+ nonexistent packages, attackers only need to claim a fraction of those

names to catch developers following AI advice. At scale, this becomes a numbers game that favors the attacker.

Dependency confusion attacks already exploit the gap between public and private package registries. AI-assisted development adds a new vector: hallucination confusion, where the AI creates demand for packages that don't exist, and attackers race to supply them.

Practical Implications for Engineering Teams

If you're responsible for developer productivity or security at a company using AI coding assistants, here's what the data suggests you should do.

Immediate Actions

Audit your current AI-assisted workflow. Determine whether your AI coding tools have real-time registry access or operate purely on trained knowledge. Most IDE-integrated assistants (Copilot, Cursor, Codeium) operate without registry verification by default. This doesn't mean stop using them—it means don't trust their package recommendations without verification.

Add verification gates. If your CI/CD pipeline doesn't already validate that all dependencies resolve to real packages before proceeding, add that check. This seems obvious, but many organizations skip this step because they assume developers wouldn't add dependencies that don't exist. AI assistants change that assumption.

Monitor for suspicious installs. Track when developers install new packages, especially packages that weren't previously in your ecosystem. Flag packages with very low download counts, recent creation dates, or names similar to popular libraries. These are exactly the packages AI hallucinations tend to suggest.

Architectural Decisions

Evaluate grounded AI alternatives. Sonatype's approach achieved zero hallucinations on the benchmark dataset. They're not the only vendor exploring this architecture. Any AI system that claims to assist with dependency management should be evaluated on whether it has real-time registry access. If the answer is no, weight their recommendations accordingly.

Consider private registry constraints. If your organization uses private package registries (Artifactory, Nexus, private npm), configure your AI tools to only recommend packages available in those registries when possible. This limits the possibility space to packages you've already vetted.

Build feedback loops. When AI recommendations fail (packages don't exist, wrong versions, security vulnerabilities), feed that information back into your tooling decision-making. Quantify the cost of AI hallucinations in your specific environment—developer time lost, security incidents caused, builds broken.

Team Education

Reset expectations about AI confidence. The GPT-5 confidence data is counterintuitive: high accuracy when confident, but confident less than 4% of the time. Train developers to treat AI package recommendations as suggestions requiring verification, not authoritative answers. The confidence signal from the AI isn't calibrated in a way that supports human trust.

Normalize manual verification. Looking up a package on npm before installing it should be standard practice when the suggestion came from an AI. This adds 30 seconds to the process and eliminates an entire category of errors.

The Vendor Landscape Shift

This report accelerates a divergence in the AI coding assistant market that was already emerging.

On one side: general-purpose LLMs (GPT, Claude, Gemini) deployed through thin IDE wrappers. These tools are powerful for code generation, explanation, and refactoring, but they lack the infrastructure to ground their outputs in real-time external data. Their package recommendations are best-effort extrapolations from training data.

On the other side: specialized developer tools with integrated data infrastructure. Sonatype's grounded AI approach is one example. GitHub Copilot's eventual integration with GitHub's package graph could be another. These tools trade generality for accuracy in specific domains.

The market will likely segment. General-purpose assistants will remain valuable for

their breadth, while specialized tools capture workflows where accuracy is non-negotiable. Dependency management falls clearly into the latter category—a 27.75% error rate is unacceptable for production systems.

> Expect every major developer tooling vendor to announce “grounded AI” or “verified AI” features within the next twelve months. The question is whether they build the data infrastructure to back up the marketing.

The winners will be organizations that own authoritative data sources. Sonatype has decades of repository indexing infrastructure. GitHub has comprehensive package graph data. npm, PyPI, and Maven Central have canonical registries. Companies that can combine LLM capabilities with verified data access will outcompete those bolting AI onto thin data foundations.

Where This Goes in Twelve Months

Three predictions based on the trajectory this report reveals.

Grounded AI becomes table stakes. Within a year, any AI coding assistant that lacks real-time package registry access will be relegated to code generation only. The liability of recommending nonexistent packages is too high, and the fix is too straightforward, for this to remain unsolved. Developers will learn to distrust tools that haven’t made this architectural investment.

Package registries monetize verification. npm, PyPI, and Maven Central have been mostly passive data stores. This report positions them as critical infrastructure for AI accuracy. Expect premium API tiers with faster response times and richer metadata, marketed specifically to AI tooling vendors. The data about what packages exist becomes as valuable as the packages themselves.

Security auditing integrates with AI recommendations. The next generation of AI coding assistants won’t just verify that packages exist—they’ll verify that packages are safe. Real-time vulnerability scanning, maintainer reputation scoring, and supply chain risk assessment will become standard features. The 1.233 million malicious packages documented in this report make this integration inevitable.

The broader implication extends beyond dependency management. Every domain

where AI generates outputs that reference external entities—API endpoints, database schemas, file paths, user accounts—faces the same grounding problem. The solutions being developed for package management will generalize to these other domains.

The Real Lesson

The Sonatype report isn't fundamentally about AI failure. It's about architectural mismatch.

LLMs are extraordinarily powerful tools for pattern recognition, generation, and transformation. They're not databases. They don't know what exists in the world right now—they know what existed in their training data and what plausibly follows from learned patterns.

Deploying them as real-time information retrieval systems without external grounding is a category error. The 27.75% hallucination rate is the predictable result of asking a generative system to perform a verification function.

The fix isn't to make AI smarter. It's to give AI access to ground truth. That access requires data infrastructure, API integration, and architectural decisions that most current AI coding tools haven't made.

The organizations that understand this distinction will deploy AI assistants that actually assist. The organizations that don't will continue to debug builds broken by packages that never existed.

AI coding assistants are exactly as reliable as the data infrastructure behind them—and right now, most of that infrastructure doesn't exist.