大規模言語モデル駆動型コーディング支援ツールにおける既知のバグおよび脆弱性再生成に関する実証的解析（2025–2026年最新動向）

序論

大規模言語モデル（LLM）を基盤とするAIコーディングアシスタントは、ソフトウェア開発のライフサイクル（SDLC）において不可欠なインフラストラクチャとして定着し、単なる静的なコード補完ツールから、文脈を解釈して自律的にタスクを遂行するエージェントへとパラダイムシフトを遂げた。市場を牽引するGitHub Copilotは、世界中の開発環境で標準的なツールとして採用されているが、その普及と並行して「AIツールは現状においても既知のバグや非推奨コード、およびセキュリティ脆弱性をそのまま再生成する傾向があるのか」という極めて重要な懸念が提起されている。

本分析が最新の実証データおよび学術研究（2025年～2026年）を網羅的に精査した結果、その問いに対する回答は明確に肯定される。すなわち、GitHub Copilotをはじめとする最先端のAIコーディングツールは、現在においても既知のバグやセキュリティの欠陥を再生成する強力な傾向を有している。この現象は、モデルの不具合というよりはむしろ、確率論に基づくテキスト生成アルゴリズムの根本的な仕様、学習データの汚染、そしてアーキテクチャ上の文脈理解の限界に起因する構造的な問題である。AIモデルは「正当性（Correctness）」や「安全性（Security）」ではなく、「統計的な蓋然性（Probability）」を最適化するように設計されているため、公開リポジトリに数十年にわたって蓄積された脆弱なパターンをそのまま出力するリスクを本質的に抱えている[1]。

本レポートでは、AI生成コードにおけるバグ再生成の根本メカニズム、脆弱性の定量的評価、GitHub Copilotが実装しているセキュリティ防御機構の実態とその限界、2025年から2026年にかけて開発者コミュニティから報告されているコード品質低下の要因、そして人間の認知バイアスが引き起こす技術的負債の増大について、多角的な視点から包括的な分析を提供する。

AI生成コードの品質とバグ再生成の根本メカニズム

AIコーディング支援ツールがバグを再生成するメカニズムを理解するためには、その基礎となる学習データの性質と、ツールのアーキテクチャパイプラインを紐解く必要がある。

大規模言語モデルは、GitHubなどの公開リポジトリに存在する数十億行のコードを学習データとして取り込んでいる。問題の核心は、これらのデータセットが必ずしもベストプラクティスに基づいているわけではなく、過去の開発者によって記述されたバグ、非効率なアルゴリズム、および既知のセキュリティ脆弱性を大量に含んでいる点にある[3]。AIモデルは、これらのコードパターンの文脈的な共起確率を学習しているに過ぎない。人間の開発者であれば、「このエンドポイントが順序外で呼び出された場合、システムはどのようにフェイルセーフを行うべきか」「認証されたユーザーであっても、この特定のオブジェクトにアクセスする権限を付与してよいのか」といった脅威モデリングに基づく推論

を行うが、AIはこのような批判的思考を持たない[1]。結果として、AIは構文的には完全に正しく、一般的なフレームワークのパターンに合致しているように見えながらも、エッジケースや悪用シナリオを一切考慮していない論理的バグを生成する[1]。

また、AI生成コードの品質に関する実証的研究は、生成されるコードの言語的熟練度が人間の開発者の期待を下回っていることを示している。2026年の「Mining Software Repositories」国際会議において発表された研究では、AIDevデータセットを活用し、3つの異なるAIエージェントによって生成された5,027のPythonファイルを含む591のプルリクエストを分析した[4]。この研究において、静的解析ツールpycefrを用いてPythonの構文を欧州言語共通参照枠（CEFR）に準じたA1（基礎）からC2（熟練）の6段階の熟練度レベルにマッピングした結果、AIエージェントが生成するコードの90%以上がA1およびA2の基礎レベルの構文に分類され、C2（熟練レベル）に分類されるコードは1%未満であったことが明らかになっている[4]。これは、AIが生成するコード自体は基礎的で理解しやすいものの、特定の高度なタスクにおいては、そのコードをレビューし保守する人間の開発者側に高度な熟練度が要求されるという非対称な関係を生み出している[4]。

さらに、バグ再生成の顕著な例として、非推奨（Deprecated）となったコードや存在しないAPIの「ハルシネーション（幻覚）」が挙げられる。Ryz Labsの分析データによれば、GitHub Copilotは全体の約15%の確率で、存在しないnpmパッケージや既に非推奨となったライブラリ、関数を提案する傾向がある[5]。AIの学習データのカットオフ（最新情報の欠落）と、古いコードベースに頻出するパターンを最新のプロジェクトに強引に適用しようとする推論の性質が組み合わさることで、古いAPIに内在する既知のセキュリティバグが最新のシステムに持ち込まれるという深刻なリスクが発生している[5]。

定量的評価：脆弱性とバグの発生頻度

2023年時点では、ChatGPTなどの汎用モデルのセキュリティ評価が研究の中心であったが、2025年から2026年にかけて、GitHub Copilotなどのコード生成特化型LLMに対する大規模かつ系統的な実証研究が多数発表され、AI生成コードがもたらすリスクが理論上の懸念ではなく、定量的に証明された実害であることが確認された。

以下の表は、近年の主要な学術研究および産業界の調査レポートが明らかにした、AI生成コードにおける脆弱性とバグの発生頻度に関する定量データを要約したものである。

調査機関 / 研究者 (発表年)	調査の前提条件および対象データ	主要な定量的発見および結論	出典
Pearce et al. (2025)	MITRE「Top 25 CWE」に基づく89のシナリオで、GitHub Copilotを用いて1,689のプログラムを生成。	セキュリティに敏感なコンテキストにおいて、生成されたコードの約**40%**に致命的な脆弱性が含まれていることを確認。	[8]

Veracode GenAI Report (2025)	Java, JavaScript, Python, C#の4言語において100以上のLLMの出力を人間のコードと比較。	全言語において、**AI**生成コードは人間の書いたコードと比較して平均**2.74**倍の脆弱性を内包していることを実証。	[14]
Fortune 50 Enterprise Research (2026)	実際のエンタープライズ環境におけるAIツールの導入効果とリスク要因を測定。	AIの導入により開発速度が4倍に向上する一方、人間と比較して**10**倍のセキュリティリスクが生じ、権限昇格パスが322%増加した。	[15]
Elsisi et al. (2026)	リポジトリの履歴からAI支援ツールのコミットを長期間にわたり追跡・分析。	全てのAIツールのコミットの15%以上が新たなバグを導入。さらに、**AI**が導入したバグの**24.2%**は最新のリビジョンでも修正されずに残留。	[13]
Empirical Study of 3.8K Bugs (2026)	Claude-Code, Codex, Gemini CLIのオープンソースリポジトリで報告された3,800件以上のバグを手動分析。	全バグの67%が機能性に関連。根本原因の37.3%がAPI、統合、または構成エラーに起因し、ツール呼び出しフェーズで37.6%のエラーが発生。	[16]

これらの定量的データは、AIコーディング支援ツールの導入がもたらすトレードオフを浮き彫りにしている。AIは構文のボイラープレートを迅速に生成し、開発速度を劇的に向上させる一方で、「セマンティック（意味論的）な脆弱性」を指数関数的に増加させる。たとえば、Fortune 50企業の内部調査では、2026年1月から3月にかけて、AIによって生成された脆弱なコードのコミット数が6倍に急増したことが報告されている [15]。

具体的な脆弱性の生成例として、GitHub CopilotにPython（Flask）を用いたファイルアップロード機能の実装を指示したケースが挙げられる。Copilotは、セキュリティ制御が一切施されていないベアボーンズのコードを生成し、任意のファイルアップロード（リモートコード実行に直結する脆弱性）を許容する実装を提案した [17]。さらに、HTTPレスポンスヘッダーにおいてHSTS（HTTP Strict Transport

Security）を設定するコードを生成させた際、セキュリティ上極めて重要なpreloadディレクティブを欠落させるなど、一見すると正常に機能しているように見えるが、セキュリティのベストプラクティスからは完全に逸脱したコードが頻繁に出力されている[17]。

セキュリティ防御機構（Vulnerability Prevention System）の実態とその限界

GitHubは、Copilotが脆弱なコードを生成する傾向があることを認識しており、その対策としてアーキテクチャ内部に複数のセキュリティレイヤーを追加している。Copilotのパイプラインは現在、ワークスペースや開いているタブからコンテキストを収集するフェーズ、責任あるAIやコンテンツ除外ポリシーを適用するプレモデルフィルター、適切なモデルへルーティングするプロキシサービス、AIモデルによる推論、そして重複コードの検出や安全確認を行うポストモデルフィルターという多段階の構造を採用している[18]。

このパイプラインのポストモデルフィルターに該当するのが、「AIベースの脆弱性防止システム（Vulnerability Prevention System）」である。このシステムは、生成されたコードが開発者のエディタに返される直前に、LLMを用いてコードを分析し、ハードコードされたクレデンシャル（CWE-798）、SQLインジェクション（CWE-89）、パストラバーサルといった一般的で危険なコーディングパターンをリアルタイムでブロックすることを目指している[19]。

しかし、実証研究および第三者機関の監査は、この防御機構の実効性に対して極めて懐疑的な見解を示している。Credo AIの評価によれば、この脆弱性フィルターは想定されるすべてのセキュリティ脆弱性を特定してブロックできる可能性は低いと結論づけられている[19]。さらに、2025年にIEEEの学会で発表された「Artificially Insecure: Examining GitHub Copilot's AI-based Vulnerability Prevention System」と題する研究では、研究者らがこのAIベースの脆弱性防止システムを意図的にバイパスし、脆弱なコードをCopilotに生成させることに成功したことが詳細に報告されている[25]。

フィルターベースの防御機構が抱える根本的な限界は、文脈依存の複雑なロジックバグを構文のパターンマッチングだけでは検出できない点にある。例えば、特定の呼び出し順序でのみ発生する競合状態（Race Condition）や、複雑なビジネスロジックにおける権限昇格（Privilege Escalation）の欠陥などは、コードの断片だけを見ても脆弱性として識別することが不可能である[30]。

事態をさらに複雑にしているのが、AI自身に生成物をレビューさせる「Copilot Code Review」機能の導入である。2025年9月に発表された実証研究では、多様なオープンソースプロジェクトから抽出されたラベル付け済みの脆弱なコードサンプルを用いて、Copilot Code Reviewの性能が評価された[31]。その結果、AIレビューアはSQLインジェクション、クロスサイトスクリプティング（XSS）、安全でないデシリアライゼーションといった重大な脆弱性を頻繁に見逃すことが判明した[31]。その代わりに、AIレビューアはコードのスタイル規約やタイポ（スペルミス）といった、セキュリティ上の重要度が極めて低い表面的な問題ばかりを指摘する傾向が確認された[31]。この現象は、AIがセマンティクス（意味論）よりもシンタックス（構文・形式）の解析に過度に偏重していることを証明しており、開発者がAIのレビュー結果を信頼した場合、致命的なバグが本番環境へとスルーされる危険性が劇的に高まることを示唆している。

新たな脅威ベクトル：Copilotインフラストラクチャ自身を標的とした攻撃

AIが生成するコードのバグに加えて、2025年から2026年にかけては、GitHub Copilotというツール自体が直接的な攻撃ベクトルとなる事象が表面化した。これは、AIコーディングアシスタントが単なるエディタのプラグインから、CI/CDパイプラインやGitHubインフラに深く統合された自律的エージェントへと進化したことに起因する。以下の表は、この期間に報告された主要なCopilot関連のCVEおよび脆弱性を示している。

脆弱性識別子 / 攻撃名称	脆弱性の種類とメカニズム	CVSS / 深刻度	出典
CVE-2025-64660	不適切なアクセス制御 **(Improper Access Control)** GitHub CopilotおよびVS Codeにおいて、認証された攻撃者がネットワーク経由でセキュリティ機能をバイパスすることを可能にする脆弱性。	5.7 (Moderate) EPSS: 0.134%	[32]
CVE-2025-59286	不適切なアクセス制御 **(類似)** Copilotインフラにおけるアクセス制御の欠陥。詳細なソースコードは非公開だが、GHSA-hh8r-hx5c-8r8rとして追跡されている。	Moderate EPSS: 0.096%	[34]
CVE-2025-53773 (Rules File Backdoor)	プロンプトインジェクション **(Prompt Injection)**	High / Critical	[22]

	設定ファイル内に隠しUnicode文字を埋め込み、Copilotのルールを書き換えることでリモートコード実行（RCE）を引き起こすサプライチェーン攻撃。		
CVE-2025-32711 (EchoLeak)	ゼロクリック・プロンプトインジェクション Copilotに対して、ユーザーの介入なしに外部からのメールや文書を通じてプロンプトを注入し、データを流出させる攻撃手法。	High / Critical	[36]
CamoLeak	データ流出 **(Data Exfiltration)** プルリクエストの説明文に悪意のある指示を隠し、Copilotにコードベース内の機密データを読み取らせ、GitHubのインフラ経由で外部へ送信させる攻撃。	高度な脅威	[37]

これらの事例は、AIの自律性が高まるにつれて、AI自身が「バグを書く」だけでなく、AI自身が「ハッキングの経路（プロキシ）」として悪用されるリスクが顕在化していることを明確に示している。特にCamoLeakやEchoLeakのようなプロンプトインジェクション攻撃は、LLMが「システムからの正当な指示」と「ユーザーまたは外部からの信頼できない入力」を厳密に区別できないというアーキテクチャ上の根本的な欠陥（OWASP LLM01:2025）を突いたものであり、AIアシスタントがエンタープライズの境界線を越えて情報を処理する際の致命的な脆弱性となっている [35]。

2025〜2026年におけるコード品質低下の根本原因（アーキテクチャの観点から）

2026年初頭、GitHub CommunityやStack Overflowなどの開発者フォーラムにおいて、「Copilotの

提案品質が顕著に低下している」「以前は文脈を理解していたのに、現在は全く的を外したコードを提案する」といった不満が爆発した[5]。この感覚は定量的にも裏付けられており、2025年のStack Overflow Developer Surveyでは、AIコーディングツールに対する肯定的な感情が、2023〜2024年の70%超からわずか60%へと急落している[5]。

この品質低下はユーザーの錯覚ではなく、以下のアーキテクチャ上の構造的変化とビジネスモデルの転換によってもたらされた現象である。

「モデル・カルーセル」現象とチューニングの乖離

品質低下の最大の要因は、背後で稼働する基盤モデルが頻繁に、かつユーザーへの明確な通知なく変更される「モデル・カルーセル（Model Carousel）」現象にある[5]。GitHub Copilotは初期のOpenAI Codexから始まり、GPT-4の様々なバリアントを経て、2025年後半からはGPT-5シリーズのモデル、さらにはClaude 3.7 SonnetやGemini 2.0といった複数プロバイダーのモデルを動的にルーティングする構造へと移行を繰り返してきた[5]。

一般に、より巨大で新しいモデル（例：GPT-5）は言語理解において優れていると想定されがちだが、ソフトウェア・エンジニアリングの文脈では必ずしもそうではない。古いモデル（Codexなど）に最適化されていたプロンプトエンジニアリングの手法やコンテキスト選択のアルゴリズムが、新しいモデルの特性に合致せず、結果として性能の後退（リグレッション）を引き起こしている[5]。このモデルの入れ替えに伴うチューニングの遅れが、バグの再生成率を高め、ユーザー体験の悪化に直結している。

コンテキストウィンドウの制限と「マルチファイル・ブラインドネス」

現代のエンタープライズ・ソフトウェア開発において、バグの多くは単一ファイル内の構文エラーではなく、複数ファイルにまたがるアーキテクチャ上の不整合によって生じる。Copilotのインライン補完における標準的なコンテキストウィンドウは約8,000トークンに制限されている[5]。

この8,000トークンという制限は、巨大なモノレポ（Monorepo）環境において致命的である。プロジェクト固有の規約やインターフェース定義をAIが参照しきれず、結果として汎用的だがプロジェクトには適合しない（あるいは論理バグを引き起こす）コードが生成される[5]。Ryz Labsの分析データによれば、10,000行を超えるプロジェクトにおいて、Copilotが正確な提案を行える確率はわずか50%にまで低下する[5]。10個以上のファイル変更を伴うタスクにおいては、アーキテクチャの相互関係を理解するのに必要なトークン数が不足するため、「マルチファイル・ブラインドネス」と呼ばれる状態に陥り、論理エラーの発生率が劇的に跳ね上がる[5]。

提案受け入れ率の低下とインフラストラクチャの不安定性

コンテキストの欠如とハルシネーションの増加により、2026年時点でのCopilotの提案受け入れ率（Acceptance Rate）は35〜40%にまで低下しており、これは競合であるCursor（42〜45%）を下回る水準となっている[5]。シニアエンジニアの75%が「手動でコードを書くよりも、Copilotが生成した誤ったコード（バグ）の修正により多くの時間を費やしている」と報告しており、生産性向上という本来の目的が相殺されつつある[5]。

さらに、IDE（Visual Studioなど）内でのCopilotは、長時間のセッションや複雑なスレッドでトークン制限に達した際、脈絡を失ったり（"scatty"な挙動）、アプリケーション全体を数分間フリーズさせたり、

最悪の場合はクラッシュさせるといった動作の不安定さも2026年初頭に広く報告されており、ツールとしての信頼性にも課題を残している[41]。2026年の報告では、Webベースのエージェントの起動に90秒以上かかるスラギッシュネス（遅延）の問題も指摘されている[5]。

また、2026年3月には「PR Ads Controversy（プルリクエスト広告論争）」と呼ばれるインシデントが発生した。Copilotが150万件以上のプルリクエストの説明文の中にプロモーション用の「ヒント（広告）」を意図せず注入（インジェクト）してしまい、Microsoftがこれを「プログラミングロジックの問題」として機能を無効化する事態となった。この事件は、AIの出力制御の難しさを示すとともに、開発者のプラットフォームに対する信頼を著しく損なう結果となった[5]。

競合AIコーディングエージェントとの比較分析

GitHub Copilotの品質問題が議論される中、AIコーディングアシスタントの市場は多様化し、特定の機能においてCopilotを凌駕する競合ツールが台頭している。以下の表は、2026年時点での主要なAIコーディングツールの特性と、バグ削減およびコンテキスト理解のアプローチを比較したものである。

ツール名称	コンテキスト理解とバグ削減のアプローチ	主な強みと弱み	出典
GitHub Copilot	エディタ統合型、Agent Modeへの移行中。脆弱性防止システムを内蔵するが、8,000トークンの制限により大規模リポジトリで文脈を見失いやすい。	【強み】圧倒的な企業導入率、エコシステム統合。 【弱み】マルチファイル編集でのバグ発生率、モデル変更による品質のばらつき。	[5]
Cursor	ディープインデクシング技術による高度な文脈把握。Plan modeによる複雑なタスクの設計と視覚的な差分表示。	【強み】提案受け入れ率が高い（42-45%）。マルチファイル編集に優れる。 【弱み】独自のフォークエディタを使用する必要がある。	[5]
Claude Code	Claude 3.7 Sonnet	【強み】巨大なコード	[5]

	等の100万（1M）トークンの巨大なコンテキストウィンドウを活用。ターミナルベースで自律的に動作。	ベース全体を推論可能。コンテキスト欠落によるバグが少ない。 【弱み】ターミナル操作に特化しておりインライン補完の体験が異なる。	
Cline	完全自律型のオープンソースエージェント。コマンドの実行、ファイルの編集、エラーの読み取りを自律的に行う。	【強み】APIキー持ち込みで安価。反復タスクの自動化に強力。 【弱み】監視（ベビーシッティング）が必要。動作が不安定になるリスク。	[7]
Amazon Q Developer	AWS環境に特化。EC2, Lambda, CloudFormationなどのインフラストラクチャコード（IaC）のセキュリティスキャンに強み。	【強み】AWSのAPIに最適化された正確なコード生成。IP補償。 【弱み】AWS以外の環境における汎用性に欠ける。	[45]
Tabnine	プライバシーを重視し、ユーザーのコードを学習データとして使用しないことを保証。ローカルでの実行オプションを提供。	【強み】高いプライバシー水準。エンタープライズのコンプライアンス要件に合致。 【弱み】推論能力において最新のクラウドベースLLMに劣る場合がある。	[21]

この比較から明らかなように、Copilotのエージェントモードは「有能だがクラス最高ではない（competent but not best-in-class）」という評価が定着しつつある[42]。特に、Claude Codeの100万トークンのコンテキストウィンドウや、Cursorのディープインデクシングと比較すると、Copilotは複雑な複数ファイルのリファクタリングにおいて、コンテキストの不一致に起因するバグを生成しやすい

アーキテクチャ上の弱点を抱えている[5]。

2026年におけるライセンスモデルの変化とデータプライバシー

バグの発生頻度や品質低下の背景には、GitHubが採用しているビジネスモデルとリソース割り当ての変更も影響している。

2026年に再編された価格階層（Free, Pro, Pro+, Business, Enterprise）において、個人向けのProプラン（月額10ドル）には、「最新モデルを利用できるプレミアムリクエスト」が月間300回に制限されるというキャップが導入された[5]。このプレミアムリクエストはAgent Modeの利用や複雑なマルチステップタスクにおいて急速に消費されるため、数日の激しいコーディングセッションで上限に達する開発者が続出している[42]。上限に達すると自動的に性能の低いベースモデルにフォールバックされるため、「月の半ばで突然コードの品質が落ち、バグだらけになる」という現象の引き金となっている[5]。また、2026年3月の発表により、学生向けの無償プランからもGPT-5.4やClaude Opus/Sonnetといったプレミアムモデルへの任意選択権が制限されることとなった[48]。プラットフォームの保守性向上のため、JetBrains IDEの古いバージョン（2024.2および2024.3）のサポートも終了している[49]。

さらに、データプライバシーとモデル改善の観点から、2026年4月24日に発効した「インタラクションデータの使用ポリシー（Interaction Data Usage Policy）」の更新は極めて重要な意味を持つ[50]。この更新により、Copilot Free、Pro、およびPro+のユーザーから収集されたインタラクションデータ（入力プロンプト、出力結果、コードスニペット、カーソル位置周辺のコンテキスト、受諾・修正の履歴）は、ユーザーが明示的にオプトアウトしない限り、AIモデルのトレーニングと改善に使用されるようになった[50]。

GitHubはこの実世界のデータを活用することで、静的なデータセットのみで学習した初期モデルの欠点を補い、バグの事前検知能力の向上や、より安全なコードパターン（Secure Code Pattern）の提供を目指していると説明している[50]。ただし、Copilot BusinessおよびEnterpriseユーザーのデータ、ならびにエンタープライズが所有するリポジトリのデータは、このトレーニングプログラムの対象外として保護されている[50]。

開発者の心理的バイアス（Automation Bias）と技術的負債の増大

技術的な制約以上に、ソフトウェア開発のエコシステムにとって深刻な問題は、人間とAIの相互作用において生じる心理的要因である。AIが極めて高速に、かつ自信に満ちたトーンでコードを生成するスピード感は、開発者の認知プロセスに深刻な死角を生み出す。

過剰信頼（Over-reliance）とレビューの形骸化

学術的な行動分析（Perry et al., 2023; Sabouri et al., 2025; Ahuchogu et al., 2025）は、開発者がAI生成コードの品質に対して「過剰信頼（Excessive Trust）」を寄せており、適切な検証を行わずにコードを盲目的に受け入れている実態を浮き彫りにしている[8]。

AIツールの普及は、ソフトウェアエンジニアの主要な役割を「コードをゼロから記述する執筆者（Author）」から「AIが生成したコードを検証するレビューア（Reviewer）」へと根本的に転換させた[4]。しかし、AIが提示するコードは「構文的に美しく、もっともらしく見える」ため、人間は心理的な警戒を解き、批判的思考を停止させやすい。セキュリティの欠陥は、明らかな文法エラーの中にあるのではなく、一見正しく見えるロジックの暗黙の仮定（Assumptions）やエッジケースの中にある[1]。

結果として生じるのは、検証されていないバグやセキュリティの脆弱性がプロダクション（本番環境）のコードベースへと統合され、後に多大な修正コストと時間を要求する「技術的負債（Technical Debt）」として蓄積されていく現象である[8]。実証研究を行ったHe et al. (2025) は、AIコーディングアシスタントの導入によって一時的な開発速度の向上（Transient velocity boost）が見られるものの、長期的にはコードの複雑性が持続的に増大することを証明している[8]。これは、AIが生成した不要なボイラープレートや冗長なロジックを、人間が適切にリファクタリングせずに放置していることを意味する。

自律型エージェント（**Agent Mode**）によるバグの広範な伝播

かつてインライン補完が主流であった時代には、AIが生成するバグはカーソル付近の数行に局所化されており、影響範囲（Blast Radius）を開発者が視覚的にコントロールしやすかった。しかし、2025年に導入され、2026年に普及したCopilot Coding Agent（クラウド・エージェント）は、バックグラウンドで非同期的に稼働し、コードベースを自律的に探索し、マルチファイルにわたる変更を一括で行い、テストを実行してコミットを完了させた状態でプルリクエストを作成する[22]。

この自律的なループの中でAIが「不適切な前提」を学習した場合、そのバグや脆弱性は1つの関数にとどまらず、複数のマイクロサービスやエンドポイントにわたって一貫して静かに伝播する[1]。たとえば、AIが誤った認証フローや安全でないロギングパターンの実装を決定した場合、そのプロジェクトのすべての新規エンドポイントに同じ欠陥が複製され、「均一に見えるが実は広範に露出した攻撃対象領域（Uniform Exposure）」を作り出してしまう[1]。Agent Modeは構文エラーやビルドエラーなどの「明確な失敗」には自己修正（Course-correction）機能を通じて適切に対処できるが[53]、テストケース自体が欠落している場合や、テストが論理的な脆弱性をカバーしていない場合、AIは致命的なバグを含んだまま「テスト通過（成功）」と判断して処理を完了させてしまう。

結論と戦略的提言

一連の実証データとアーキテクチャの動向分析から、ユーザーの問いに対する回答は極めて明確である。**GitHub Copilot**は、**2026**年現在においても、既知のバグや非推奨コード、セキュリティ脆弱性をそのまま再生成する強い傾向を保持している。

この傾向は、AIが「統計的に最も出現頻度の高いコード（過去の脆弱なパターンを含む）」を生成しようとするLLMの基本原理に起因しており、Copilotに内蔵されたAIベースの脆弱性防止システム（リアルタイムフィルター）の能力を凌駕している。さらに、モデルの頻繁な切り替え（Model Carousel）によるチューニングの乖離や、大規模プロジェクトにおける8,000トークンというコンテキストウィンドウの枯渇が、論理的バグやハルシネーションの発生率を押し上げている。

AIツールの進化（Agent Modeによるマルチファイル自律編集）は、開発スピードを劇的に向上させる一方で、バグや脆弱性が広範なアーキテクチャに一瞬にして伝播するリスクをもたらした。開発者が「AIを過剰に信頼し、十分な検証を行わずにコードを受け入れる（Automation Bias）」という心理的要

因が相まることで、今日のソフトウェア・サプライチェーンには深刻な技術的負債が急速に蓄積されている。

AI生成コードに伴うバグ再生成の脅威に対抗するためには、AIへの依存を前提とした新たなエンジニアリング・パラダイムと防御層の構築が不可欠である。

1. **AI駆動型SAST**（静的アプリケーションセキュリティテスト）の必須化 Copilotが自律的に生成する膨大なコードを監視するために、従来のシグネチャベースのSASTだけでなく、コンテキストや実行フローを意味論的に解釈できるAIベースのセキュリティスキャナーをCI/CDパイプラインに深く統合する必要がある[1]。AIが高速でコードを生成するならば、その検証も同等の速度と精度で自動化されなければならない。
2. プロンプトの細分化（**Prompt Partitioning**）とコンテキストの統制 Copilotの限られたコンテキストウィンドウを最大限に活かし、ハルシネーションの発生を防ぐためには、複雑なタスクを小さく分割し、不必要なファイルを閉じてAIの「視界（Blast Radius）」を人為的に制限することが有効である[47]。
3. **Copilot Code Review**への過信の排除 AI（Copilot）が書いたコードをAI（Copilot Code Review）にレビューさせるという「閉鎖的なエコーチェンバー」を避けるべきである。実証研究が示す通り、AIレビューはSQLインジェクションなどの重大な脆弱性を見逃すため[31]、セキュリティクリティカルなパスにおいては必ず人間の専門家または独立した堅牢なセキュリティツールによる監査を実施しなければならない。
4. プロンプトインジェクションに対する防御層（**Defense-in-Depth**）の構築 EchoLeakやRules File Backdoorの事例が示すように、プルリクエストのコメントや設定ファイルに仕込まれた悪意あるコードがCopilotを介して実行されるリスクに備える必要がある[22]。外部入力を処理する際の厳格なコンテンツセキュリティポリシー（CSP）の適用や、プロビナンス（出所）ベースのアクセス制御を実装することが強く推奨される[36]。

AIコーディングアシスタントは、もはや単なる「タイピングの補助」ではなく、システムの基盤を担う「無口な共同執筆者」である。その生成物を盲信することは、過去数十年間のオープンソース界隈に蓄積されたすべてのバグと脆弱性を、自身のプロジェクトへ無批判に迎え入れることに他ならないという認識を、組織全体で共有する必要がある。

引用文献

1. Vulnerabilities of Coding with GitHub Copilot: When AI Speed Creates Invisible Risk, 4月 18, 2026にアクセス、https://brightsec.com/blog/vulnerabilities-of-coding-with-github-copilot-when-ai-speed-creates-invisible-risk/
2. What is the Responsibility of Developers Using Generative AI? A, 4月 18, 2026にアクセス、https://learn.modernagecoders.com/blog/what-is-the-responsibility-of-developers-using-generative-AI
3. [2512.05239] A Survey of Bugs in AI-Generated Code - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/abs/2512.05239
4. When is Generated Code Difficult to Comprehend? Assessing AI Agent Python

Code Proficiency in the Wild - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/html/2604.00299v1
5. Is GitHub Copilot Getting Worse in 2026? What Changed & Why ..., 4月 18, 2026にアクセス、https://nxcode.io/resources/news/github-copilot-getting-worse-2026-developers-switching
6. Choosing the Best AI for Coding in 2026 - Blog - Silk Data, 4月 18, 2026にアクセス、https://silkdata.tech/blog/article/best-ai-for-coding
7. I Tested 12 AI Coding Tools. Here's the Only Ranking That Matters. - Medium, 4月 18, 2026にアクセス、https://medium.com/lets-code-future/i-tested-12-ai-coding-tools-heres-the-only-ranking-that-matters-27c7c00e0e7f
8. Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild, 4月 18, 2026にアクセス、https://arxiv.org/html/2603.28592v1
9. A Survey of Bugs in AI-Generated Code - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/html/2512.05239v1
10. TypePilot: Leveraging the Scala type system for secure LLM-generated code - ACL Anthology, 4月 18, 2026にアクセス、https://aclanthology.org/2025.ommm-1.11.pdf
11. A Survey on Large Language Models in Software Security: Opportunities and Threats, 4月 18, 2026にアクセス、https://www.mdpi.com/2073-431X/15/4/226
12. Malicious Attack Challenges and Mitigation ... - ScholarSpace, 4月 18, 2026にアクセス、https://scholarspace.manoa.hawaii.edu/bitstreams/74c9194b-4b3f-4997-a15a-9acb295f8540/download
13. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's ..., 4月 18, 2026にアクセス、https://www.researchgate.net/publication/388193053_Asleep_at_the_Keyboard_Assessing_the_Security_of_GitHub_Copilot's_Code_Contributions
14. AI-Generated Code Security Risks - Why Vulnerabilities Increase 2.74x and How to Prevent Them - SoftwareSeni, 4月 18, 2026にアクセス、https://www.softwareseni.com/ai-generated-code-security-risks-why-vulnerabilities-increase-2-74x-and-how-to-prevent-them/
15. The 23.7% Security Vulnerability Tax: Is AI-Generated Code Worth the Risk? - TianPan.co, 4月 18, 2026にアクセス、https://tianpan.co/forum/t/the-23-7-security-vulnerability-tax-is-ai-generated-code-worth-the-risk/3865
16. Engineering Pitfalls in AI Coding Tools: An Empirical Study of Bugs in Claude Code, Codex, and Gemini CLI - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/html/2603.20847
17. Insecure coding workshop: Analyzing GitHub Copilot suggestions - Invicti, 4月 18, 2026にアクセス、https://www.invicti.com/blog/web-security/analyzing-security-github-copilot-suggestions
18. GitHub Copilot Workflow Cheatsheet — 2026 Edition, 4月 18, 2026にアクセス、

https://sukurcf.github.io/resources/github-copilot-cheatsheet.html
19. Github Copilot - AI Vendor Risk Profile, 4月 18, 2026にアクセス、https://www.credo.ai/ai-vendor-directory/github-copilot
20. Week 2: Features & Data handling – Copilot Free learning & cert prep #151641 - GitHub, 4月 18, 2026にアクセス、https://github.com/orgs/community/discussions/151641
21. 16 Best AI Coding Assistants to Boost Your Engineering Productivity in 2025 | DigitalOcean, 4月 18, 2026にアクセス、https://www.digitalocean.com/resources/articles/best-ai-coding-assistant
22. A Developer's Guide to Writing Secure Code with GitHub Copilot - StackHawk, 4月 18, 2026にアクセス、https://www.stackhawk.com/blog/github-copilot-secure-coding-guide/
23. What is GitHub Copilot? - Pangea.app, 4月 18, 2026にアクセス、https://pangea.app/glossary/github-copilot
24. 1月 1, 1970にアクセス、https://github.blog/2023/02/14/github-copilot-now-has-a-vulnerability-prevention-system-to-block-insecure-code-patterns/
25. XOXO: STEALTHY CROSS-ORIGIN CONTEXT POI ... - OpenReview, 4月 18, 2026にアクセス、https://openreview.net/pdf/6179550cf18cac2acd7b49b7e499ae47ac39270e.pdf
26. Artificially Insecure: Examining GitHub Copilot's AI-based Vulnerability Prevention System, 4月 18, 2026にアクセス、https://ieeexplore.ieee.org/document/11133629/
27. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions, 4月 18, 2026にアクセス、https://www.computer.org/csdl/proceedings-article/sp/2022/131600a980/1FlQxERjKCs
28. Connected Papers | Find and explore academic papers, 4月 18, 2026にアクセス、https://www.connectedpapers.com/main/5de990a34c7ea1da2b3a98545422573ba6f44873
29. AI-assisted Collaboration: Exploring Developer Experience with GitHub Copilot and Windsurf - IEEE Computer Society, 4月 18, 2026にアクセス、https://www.computer.org/csdl/magazine/so/5555/01/11429684/2eNCjd5qusw
30. AI Code Security: Why Defenders Can't Afford to Fall Behind, 4月 18, 2026にアクセス、https://pluto.security/blog/ai-code-security-defenders/
31. GitHub's Copilot Code Review: Can AI Spot Security Flaws Before You Commit? - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/html/2509.13650v1
32. Improper access control in GitHub Copilot and Visual... · CVE-2025-64660, 4月 18, 2026にアクセス、https://github.com/advisories/GHSA-j8xq-6qq7-vfv7
33. CVE-2025-64660 Detail - NVD, 4月 18, 2026にアクセス、https://nvd.nist.gov/vuln/detail/CVE-2025-64660
34. Copilot Spoofing Vulnerability · CVE-2025-59286 · GitHub Advisory Database, 4月 18, 2026にアクセス、https://github.com/advisories/GHSA-hh8r-hx5c-8r8r
35. Prompt Injection Attacks in Large Language Models and AI Agent Systems: A Comprehensive Review of Vulnerabilities, Attack Vectors, and Defense Mechanisms - Preprints.org, 4月 18, 2026にアクセス、

https://www.preprints.org/manuscript/202511.0088?utm_source=chatgpt.com
36. EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System - arXiv, 4月 18, 2026にアクセス、https://arxiv.org/html/2509.10540v1
37. CamoLeak: How GitHub Copilot Became An Exfiltration Channel | BlackFog, 4月 18, 2026にアクセス、https://www.blackfog.com/camoleak-how-github-copilot-became-an-exfiltration-channel/
38. Prompt Injection Attacks in Large Language Models: Vulnerabilities, Exploitation Techniques, and Defense Strategies | by Khmaïess Jannadi | Medium, 4月 18, 2026にアクセス、https://medium.com/@jannadikhemais/prompt-injection-attacks-in-large-language-models-vulnerabilities-exploitation-techniques-and-e00fe683f6d7
39. Is GitHub Copilot Getting Worse in 2026? What Changed & Why Devs Are Switching, 4月 18, 2026にアクセス、https://www.nxcode.io/resources/news/github-copilot-getting-worse-2026-developers-switching
40. Why Teams Use GitHub Copilot Wrong and Fix It in 2026 - ThoughtMinds, 4月 18, 2026にアクセス、https://thoughtminds.ai/blog/why-teams-use-github-copilot-wrong-and-fix-it
41. Grrr Co-Pilot in Visual Studio 2026 is so impressive and frustrating ..., 4月 18, 2026にアクセス、https://github.com/orgs/community/discussions/187525
42. GitHub Copilot 2026: Complete Guide to Pricing, Agent Mode & Coding Agent | NxCode, 4月 18, 2026にアクセス、https://www.nxcode.io/resources/news/github-copilot-complete-guide-2026-features-pricing-agents
43. Cursor vs. GitHub Copilot: Which AI Coding Assistant Is Better? | DataCamp, 4月 18, 2026にアクセス、https://www.datacamp.com/es/blog/cursor-vs-github-copilot
44. Using AI Agent Cline to Remove Deprecated Code from SAP Fiori Applications, 4月 18, 2026にアクセス、https://community.sap.com/t5/technology-blog-posts-by-sap/using-ai-agent-cline-to-remove-deprecated-code-from-sap-fiori-applications/ba-p/14333374
45. CodeWhisperer (Amazon Q) vs. Copilot: Best AI Coding Assistant for Enterprise?, 4月 18, 2026にアクセス、https://www.augmentcode.com/tools/codewhisperer-vs-copilot-best-ai-coding-assistant-for-enterprise
46. GitHub Copilot · Your AI pair programmer, 4月 18, 2026にアクセス、https://github.com/features/copilot
47. What I've Learned About GitHub Copilot Agent Mode - DEV Community, 4月 18, 2026にアクセス、https://dev.to/anchildress1/what-ive-learned-about-github-copilot-agent-mode-4co2
48. Important Updates to GitHub Copilot for Students · community · Discussion #189268, 4月 18, 2026にアクセス、https://github.com/orgs/community/discussions/189268

49. Important updates: GitHub Copilot support ending for JetBrains 2024.2 and 2024.3, 4月 18, 2026にアクセス、 https://devblogs.microsoft.com/java/important-updates-ghsupport-ending-for-jetbrains-2024-2-and-2024-3/
50. Updates to GitHub Copilot interaction data usage policy - The ..., 4月 18, 2026にアクセス、 https://github.blog/news-insights/company-news/updates-to-github-copilot-interaction-data-usage-policy/
51. Usage, Effects and Requirements for AI Coding Assistants in the Enterprise: An Empirical Study - arXiv, 4月 18, 2026にアクセス、 https://arxiv.org/html/2601.20112v1
52. (PDF) Usage, Effects and Requirements for AI Coding Assistants in the Enterprise: An Empirical Study - ResearchGate, 4月 18, 2026にアクセス、 https://www.researchgate.net/publication/400178243_Usage_Effects_and_Requirements_for_AI_Coding_Assistants_in_the_Enterprise_An_Empirical_Study
53. Agent mode 101: All about GitHub Copilot's powerful mode - The ..., 4月 18, 2026にアクセス、 https://github.blog/ai-and-ml/github-copilot/agent-mode-101-all-about-github-copilots-powerful-mode/