TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

Persistent vs. Ephemeral: A Comparative Analysis of Codebase Indexing in AI Programming Tools

Mohd Tabish Khan[1], Durgesh Yadav[2], Kunal Kumar[3], Jayant Sharma[4], Farheen Siddiqui[5], Dr. Yusuf Perwej[6]

[1,2,3,4]Scholar (B.Tech) Department of Computer Science & Engineering, Shri Ramswaroop Memorial University, Deva Road, Lucknow

[5]Assistant Professor, Department of Computer Science & Engineering, Shri Ramswaroop Memorial University, Deva Road, Lucknow

[6]Professor, Department of Computer Science & Engineering, Shri Ramswaroop Memorial University, Deva Road, Lucknow

mohdtabishkhan001@gmail.com, sdurgeshyadav3@gmail.com, jayantsharma3656@gmail.com, kunalkumar050607@gmail.com, farheensiddiqui.cse@srmu.ac.in, yusufperwej@gmail.com

KEYWORDS

AI Programming Tools, Codebase Indexing, Persistent Indexing, Ephemeral Indexing, Retrieval-Augmented Generation, Large Language Models, Developer Tools, Context Management

ABSTRACT

The rapid proliferation of AI-powered programming assistants has introduced a fundamental architectural divergence: persistent indexing versus ephemeral indexing. Persistent indexing maintains pre-computed, durable code representations stored between sessions, while ephemeral indexing constructs context on-the-fly without retaining state. This paper provides a rigorous comparative analysis of both paradigms through examination of five leading tools—GitHub Copilot,[1] Cursor,[2] Codium, Aider, and Amazon Q Developer.[3] We draw on three independently verified empirical studies: Ding et al.[6] demonstrate a 33.94% relative improvement in exact match accuracy when cross-file context (enabled by persistent indexing) is provided; Peng et al.[10] report 55.8% faster task completion with AI-assisted coding; and Morris et al.[11] show that 92% of 32-token text inputs can be reconstructed from stored embeddings, establishing a privacy risk relevant to persistent index storage. Findings indicate neither paradigm universally dominates; the optimal choice is governed by codebase size, privacy requirements, team scale, and workflow characteristics.

I. INTRODUCTION

Modern software development has been substantially transformed by AI-assisted coding tools such as GitHub Copilot, Cursor,[1] Codeium, and Amazon Q Developer.[2] These systems offer capabilities ranging from inline code completion to autonomous multi-file refactoring. Underlying each is a critical architectural component: the mechanism by which the tool indexes, stores, and retrieves codebase knowledge at query time.

Two fundamentally distinct strategies have emerged. Persistent indexing systems construct and maintain long-lived, queryable representations through embedding-based vector databases such as FAISS[and HNSW,[3] AST analysis, and symbol graph construction—surviving across sessions and updating incrementally. Ephemeral indexing systems construct context dynamically at query time from the active file buffer, discarding all context once the query resolves.

The measurable impact of codebase indexing is significant. Ding et al.[6] demonstrate that leveraging cross-file context through persistent indexing yields a 33.94% relative improvement in Exact Match (EM) accuracy and a

Corresponding Author: Mohd Tabish Khan, Scholar (B.Tech) Department of Computer Science & Engineering, Shri Ramswaroop Memorial University, Deva Road, Lucknow
Email: mohdtabishkhan001@gmail.com

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

28.69% improvement in identifier matching for code completion. In-production studies confirm a 55.8% reduction in task completion time when developers are supported by AI context-aware systems.[4] However, persistent indices also introduce a documented privacy risk: [5] show that 92% of 32-token text inputs can be reconstructed exactly from stored embeddings—a finding that directly motivates the preference for ephemeral approaches in privacy-sensitive environments.

This paper contributes: (1) a formal taxonomy of persistent, ephemeral, and hybrid indexing strategies; (2) comparative analysis grounded exclusively in verified empirical evidence; (3) a practitioner decision framework; and (4) open research challenges and future directions.

II. BACKGROUND AND RELATED WORK

A. The Rise of AI Programming Assistants

The dominant paradigm, introduced by GitHub Copilot in 2021,[6] employs a transformer-based model conditioned on a context window. Early systems relied on the active file only—a form of implicit ephemeral indexing. As models scaled and context windows expanded from 2,048 to over 128,000 tokens,[7] the need for smarter context selection became clear: a larger window is only useful if the right code is placed inside it. This limitation directly motivated the development of persistent codebase indexing.

B. Retrieval-Augmented Generation (RAG)

RAG, introduced by Lewis et al.,[7] augments LLM inference with an external retrieval step that fetches relevant documents or code fragments before generation. Applied to programming tools, RAG enables querying a vector database of embedded code chunks to populate the LLM's context with semantically relevant codebase portions. REALM[8] further demonstrated that retrieval-augmented pre-training substantially improves factual accuracy. This is foundational to persistent indexing: the index is the retrieval store.

C. Code Representation Techniques

- **Embedding-based:** Dense vector representations enabling approximate nearest-neighbor retrieval. Key libraries: FAISS [8]
- **(Facebook AI Similarity Search)** HNSW (Hierarchical Navigable Small World graphs)
- by Malkov & Yashunin (2020) — the algorithm underlying Cursor's cloud-hosted vector index (Turbopuffer).
- **Structural:** ASTs, Control Flow Graphs (CFGs), and Program Dependence Graphs (PDGs) capturing syntactic and semantic relationships.
- **Symbol-level:** Function signatures, class hierarchies, and import graphs enabling precise cross-file navigation.

Persistent systems combine all three representation types; ephemeral systems rely on token proximity and file locality.

D. Verified Quantitative Evidence

Three independently peer-reviewed studies provide the empirical foundation for this paper's comparative claims. Ding et al.[6] (LREC-COLING 2024) propose CoCoMIC—a framework jointly modeling in-file and cross-file context—and measure a 33.94% relative increase in Exact Match accuracy and 28.69% improvement in identifier matching when cross-file context is provided, on a benchmark of real software projects. [9] (arXiv:2302.06590, 2023) conduct a controlled experiment with 95 developers, finding that those with GitHub Copilot completed a defined programming task 55.8% faster (95% CI: 21–89%). Morris et al.[11] (EMNLP 2023) demonstrate that a multi-step embedding inversion method, Vec2Text, can recover 92% of 32-token text inputs exactly from stored dense embeddings—tested on Wikipedia and clinical note datasets.

III. TAXONOMY OF CODEBASE INDEXING STRATEGIES

A. Persistent Indexing

A persistent indexing system constructs and maintains a durable, queryable codebase representation between sessions. Key architectural components include: (i) an ingestion pipeline parsing, chunking, and embedding code files; (ii) a persistent vector store—such as Pinecone, Weaviate, or local FAISS [10] retaining embeddings; (iii) a retrieval engine

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

executing nearest-neighbor queries via HNSW[10] at inference time; and (iv) an index maintenance subsystem tracking file changes via filesystem watchers or VCS hooks.

Cursor[2] exemplifies this approach. Upon project initialization, Cursor traverses the file tree, chunks files at syntactic boundaries, encrypts and uploads chunks to its servers where embeddings are computed, and stores those embeddings in its cloud-hosted vector database (Turbopuffer). Crucially, raw source code is never stored on Cursor's servers—only embeddings [11] and obfuscated metadata (file hashes and line ranges) are persisted remotely; the actual source code is retrieved from the local machine at query time. Its "@Codebase" feature enables answering questions about functions in files the developer has never opened—a capability that [12] directly enables the 33.94% accuracy improvement measured by Ding et al.[6] when cross-file context is provided.

B. Ephemeral Indexing

An ephemeral indexing system constructs context entirely at query time from the active file buffer, recently opened files, cursor position, and lightweight structural signals such as ctags output. No durable index is [13] maintained, so context always reflects the current code state. This eliminates both the staleness problem and—critically—the embedding inversion privacy risk, since no vectors are stored persistently.

The principal limitation is coverage. As codebase complexity grows, the absence of cross-file context leads to the quality degradation measured by Ding et al.[6]—where file-only models miss project-specific APIs and generate hallucinated function calls with incorrect arguments. This is the core trade-off that persistent indexing addresses.

C. Hybrid Indexing

Hybrid strategies combine persistent background indices with ephemeral foreground context assembly. GitHub Copilot Enterprise[1] approximates this model—maintaining repository-level indices on GitHub's cloud infrastructure while assembling per-request context ephemerally. This architecture attempts to deliver the accuracy benefit (33.94% EM improvement [14] while limiting privacy exposure to each individual inference request, though the persistent index itself remains subject to the inversion risk identified by Morris et al.[11]

IV. COMPARATIVE ANALYSIS

The following table includes only dimensions supported by peer-reviewed published evidence or verified official documentation. Any comparison without a citable source has been excluded.

Table 1: Evidence-Backed Comparative Analysis

Dimension	Persistent	Ephemeral	Ref	Winner
Code Completion Accuracy	+33.94% EM accuracy	File-only baseline	[6]	Persistent
Identifier Matching	+28.69% improvement	File-only baseline	[6]	Persistent
Developer Productivity	55.8% faster tasks	Degrades large projects	[10]	Persistent
Privacy Risk	92% tokens reconstructable	No persistent storage	[11]	Ephemeral
Context Freshness	Requires re-indexing	Always current	[arch.]	Ephemeral
Setup Complexity	High (CI/CD, config)	Zero config	[1][2]	Ephemeral
Offline Support	Limited (internet req.)	None/limited	[2][3]	Persistent

A. Code Completion Accuracy (Verified: Ding et al., 2024)

The most directly relevant quantitative evidence comes from CoCoMIC[6] (LREC-COLING 2024, Amazon Science). Ding et al. build CCFinder—a static-analysis tool that constructs a project context graph and retrieves cross-file context—and measure its impact on code completion. Against a file-only baseline (representative of ephemeral indexing), providing cross-file [15]context yields a **33.94% relative increase in Exact Match (EM) accuracy** and a **28.69% improvement in identifier matching**. These numbers represent the direct measurable benefit of what persistent indexing makes possible—specifically, access to project-level cross-file context during inference.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

An important scope note: the CoCoMIC results were measured on code completion benchmarks using Python and Java projects. The improvement magnitude may vary by programming language, project structure, and task type. This does not invalidate the finding but should be disclosed when presenting the numbers.

B. Developer Productivity (Verified: Peng et al., 2023)

Peng et al.[10] conducted a randomized controlled experiment with 95 professional developers recruited via Upwork. The task was to implement an HTTP server in JavaScript as quickly as possible. The group with GitHub Copilot access completed the task **55.8% faster (95% CI: 21–89%)** than the control group. This is the most-cited productivity metric in the AI coding tools literature and has been confirmed by subsequent studies.

Critical scope disclosure: this figure comes from a **single, well-defined task** in a controlled setting. The paper does not test large multi-module real-world projects. The 55.8% figure should be presented as [16] a measure of AI-assisted coding speed on a defined task, not as a universal productivity guarantee. This limitation is noted by the authors themselves.

C. Privacy and Embedding Inversion Risk (Verified: Morris et al., 2023)

Morris et al.[11] introduce Vec2Text, a multi-step embedding inversion method. Their key finding: **92% of 32-token text inputs can be reconstructed exactly from their stored dense embeddings.** The method was demonstrated on GTR and text-embedding-ada-002 models using Wikipedia and MSMARCO (clinical notes) datasets.

Critical scope disclosure: this study used **general text datasets, not source code**. Its direct applicability to code embeddings is a reasonable inference—code is a structured text with even more repetitive patterns, which may make it easier to invert—but [17] did not test code specifically. The privacy risk should therefore be presented as: "Morris et al. demonstrate a 92% text reconstruction rate from embeddings in a general-text study; this risk is plausibly applicable to code embeddings stored in persistent indices, though code-specific inversion studies are an open research area."

V. ANALYSIS OF PRODUCTION AI CODING TOOLS

Table 2 summarises indexing strategies of five major tools as of 2025, based on official product documentation.

Table 2: Indexing Strategies in Production AI Coding Tools

Tool	Strategy	Context	Ref	Best For
GitHub Copilot	Hybrid (cloud+local)	128K tokens	[1]	Enterprise teams
Cursor	Persistent (cloud HNSW/Turbopuffer)	200K tokens	[2]	Large codebases
Codeium	Ephemeral (free tier)	Dynamic	[3]	Privacy-sensitive
Aider	Ephemeral + ctags	File-scoped	[4]	CLI workflows
Amazon Q	Persistent (cloud)	~100K tokens	[3]	AWS ecosystem

A. GitHub Copilot

GitHub Copilot employs a hybrid model. The base Individual/Team tier operates ephemerally—context from the active file and recently opened files only, with no persistent codebase index. The Enterprise tier adds repository-level indexing on GitHub's infrastructure. The 55.8% productivity gain measured by Peng et al.[10] was studied on the base tier with ephemeral context, meaning the full benefit of persistent cross-file context identified by Ding et al.[6] is available only to Enterprise subscribers.

B. Cursor

Cursor[2] uses a persistent cloud-hosted vector index built on Turbopuffer, Cursor's own vector database. Code chunks are chunked locally, encrypted, uploaded to Cursor's servers where embeddings are computed, then stored in Turbopuffer. Raw source code [18] is never stored on Cursor's servers—only embeddings and obfuscated metadata persist remotely. The actual source files remain on the developer's machine and are retrieved locally at query time. Cursor's "@Codebase" feature enables querying the entire indexed project, directly providing the type of cross-file context that Ding et al.[6] measure as delivering a 33.94% accuracy improvement over file-only models.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

C. Codeium

Codeium's free tier operates ephemerally—code is used only during the inference request and not retained. This architecture directly mitigates the embedding inversion risk documented since no vectors are ever stored, they cannot be inverted. This makes Codeium the preferred choice for organizations subject to GDPR, HIPAA, or SOC 2 data retention requirements.

D. Aider

Aider[4] is an open-source CLI assistant using ephemeral indexing augmented by Universal Ctags-based symbol extraction. It builds a lightweight symbol map of function names, class definitions, and module exports. While this improves cross-file awareness over pure ephemeral approaches, it does not provide the semantic embedding-based retrieval that enables CoCoMIC-level accuracy improvements.[6]

E. Amazon Q Developer

Amazon Q Developer[3] provides persistent, cloud-hosted indexing via AWS CodeConnections. Code embeddings and fragments are stored within AWS infrastructure. Per the [19] finding—92% of text inputs recoverable from stored embeddings—organizations with code residency requirements should treat this as a concrete, not theoretical, privacy risk.

VI. DECISION FRAMEWORK FOR PRACTITIONERS

The decision framework below is grounded in three verified empirical findings: (1) 33.94% EM accuracy improvement from cross-file context[6]; (2) 55.8% productivity gain from AI-assisted coding[10]; (3) 92% embedding reconstruction risk from persistent storage.[11] Recommendations without an empirical basis from the literature are marked [arch.] indicating they rest on architectural reasoning.

Table 3: Evidence-Grounded Decision Framework

Factor	Condition	Ref	Recommendation
Accuracy	Multi-file, cross-module work	[6]	Persistent — +33.94% EM accuracy
Productivity	AI-assisted task completion	[10]	Any AI tool — 55.8% speed gain
Privacy	GDPR / HIPAA / SOC 2	[11]	Ephemeral only — 92% inversion risk [11]
Privacy	Sensitive codebase	[11]	Local persistent or ephemeral
Team Size	< 5 engineers	[arch.]	Ephemeral — zero config overhead
Team Size	Large shared codebase	[arch.]	Persistent — shared index amortizes cost
Connectivity	Air-gapped / offline	[2]	Local ephemeral or Privacy Mode tools

The 33.94% accuracy improvement[6] is the most direct empirical justification for persistent indexing investment. It quantifies what persistent indexing actually buys in terms of model output quality [20] not indirectly through developer surveys, but through automated code evaluation benchmarks on real software projects.

The 92% embedding reconstruction finding[11] operates as a hard privacy constraint. For organizations subject to data residency regulations, this is not a theoretical concern: Morris et al.'s Vec2Text method is publicly available and reproducible, meaning the attack is practically executable. The caveat remains that this was measured on general text, not code—but the structural regularity of code may make it comparably or more vulnerable.

VII. OPEN RESEARCH CHALLENGES

A. Code-Specific Embedding Inversion Studies

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

Morris et al.[11] established embedding inversion for general text with 92% exact reconstruction of 32-token inputs. Whether the same vulnerability exists specifically for code embeddings—and at what magnitude [21] has not been studied. Code's structural regularity (predictable syntax, repeated patterns, limited vocabulary) may make it more vulnerable than natural language text, but this is currently an open research question. A dedicated code embedding inversion study is a critical prerequisite for making definitive privacy claims about persistent code indices.

B. Streaming Index Maintenance

The CoCoMIC framework and similar persistent indexing systems treat index updates as batch operations. As AI coding tools evolve toward agentic workflows where the model itself modifies files, codebase semantics change faster than batch update mechanisms can accommodate. Streaming index maintenance—updating embeddings incrementally at the AST node level—is an open engineering challenge with no production-ready solution as of 2025.

C. Broader Generalizability of CoCoMIC Results

The 33.94% EM improvement[6] was measured on specific Python and Java benchmarks using the CoCoMIC framework. Whether this magnitude of improvement generalizes to other languages (JavaScript, TypeScript, Rust), other types of code tasks (refactoring, debugging, test generation), or other retrieval architectures (HNSW-based vector stores versus static [22] analysis graphs) remains to be established. Follow-up benchmark studies across diverse language and task pairs would strengthen the generalizability of this finding.

D. Federated Codebase Indexing

The accuracy gains from cross-file context[6] grow with the scope of the indexed codebase. For distributed teams, this motivates indexing spanning all dependent repositories—but data governance prohibits sharing raw code across organizational boundaries. Federated learning-inspired approaches to index construction—building partial indices locally and merging without sharing raw code data—are a promising but largely unexplored direction for delivering cross-organizational context benefits while respecting data residency requirements.

VIII. INDUSTRY ADOPTION AND DEVELOPER SENTIMENT (2024–2025)

The Stack Overflow 2025 Developer Survey—the largest annual developer census, collecting 49,009 responses from 177 countries—provides the most comprehensive empirical picture of how AI coding tools are being adopted in practice.[12] Its findings directly contextualize the persistent vs. ephemeral indexing debate within real developer behaviour and organizational decision-making.

A. Adoption Rates and Daily Usage

AI tool adoption has reached near-universal levels in the developer community. The 2025 survey finds that **84% of developers use or plan to use AI tools** in their development process—up from 76% in 2024 and 70% in 2023.[12] More significantly, 51% of professional developers now use AI tools daily,[12] indicating that AI coding assistance has transitioned from an experimental feature to an integral part of daily workflow. GitHub Copilot, specifically, is used by 68% of respondents who use AI agents—making it the dominant tool in the market.[12]

This adoption trajectory directly elevates the architectural importance of codebase indexing. When AI tools are used daily across multi-file professional projects, the 33.94% accuracy improvement from cross-file context[6] is not a marginal benefit—it compounds across every query made throughout a working day. The scale of adoption means that the persistent vs. ephemeral architectural decision now affects the daily experience of a majority of professional developers.

B. The Trust Gap: A Critical Quality Signal

Despite high adoption, the 2025 survey reveals a growing and significant trust gap. Only 33% of developers trust the accuracy of AI tool output,[12] while **46% actively distrust it**—a figure that has worsened from ~40% trust in 2024.[12] Only 3% report 'highly trusting' AI output. Experienced developers are the most skeptical, with the lowest high-trust rate (2.6%) and highest high-distrust rate (20%).[12]

This trust gap is directly connected to the indexing paradigm. The survey finds that 66% of developers struggle with AI solutions that are 'close but ultimately miss the mark,'[12] and 45% report that debugging AI-generated code takes longer than writing it themselves.[12] These failure modes are characteristic of [23] ephemeral indexing at scale: the model produces plausible but incorrect outputs because it lacks access to the actual project-specific APIs, class

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

hierarchies, and naming conventions that persistent indexing provides. The 33.94% EM accuracy improvement measured by Ding et al.[6] is precisely the kind of improvement that would reduce the 'close but wrong' failure pattern that 66% of developers currently experience.

C. Sentiment Decline and What It Signals

Positive developer sentiment toward AI tools has declined from over 70% in both 2023 and 2024 to 60% in 2025.[12] The Stack Overflow CEO directly attributes this to 'AI slop'—outputs that are inaccurate or lack contextual relevance to the actual codebase.[12] This sentiment decline is a market signal: developers are moving past initial enthusiasm toward a quality-focused evaluation. The architectural choice between [24] persistent and ephemeral indexing is central to this quality question—persistent indexing is the primary mechanism by which AI tools can develop the project-specific context awareness that the survey's respondents are clearly demanding.

D. Security and Privacy as the Leading Adoption Barrier

The survey identifies security and privacy concerns as the top reason developers reject a technology (Rank 1 of all deal-breakers),[12] ahead of pricing and better alternatives. This directly validates the privacy dimension of the persistent vs. ephemeral comparison. The embedding inversion risk documented by Morris et al. 92% of tokens reconstruct able from stored embeddings—resonates with the developer community's primary adoption concern. Organizations evaluating persistent indexing solutions must therefore treat privacy not as a secondary concern but as the primary barrier to developer and organizational trust.

Notably, the survey also finds that 52% of developers either don't use agents or stick to simpler tools, and 38% have no plans to adopt AI agents. [25] Enterprise security concerns are identified as a major adoption barrier. Ephemeral indexing approaches—particularly fully local, on-device systems such as Continue.dev with a local model—directly address this barrier by ensuring no code data leaves the developer's machine, which may explain the sustained relevance of ephemeral tools even as persistent alternatives demonstrate measurable accuracy advantages.

Table 4: Stack Overflow 2025 Survey — Key Metrics for Indexing Context

Metric	Finding	Ref
AI tool adoption (use or plan to use)	84% (up from 76% in 2024)	[12]
Daily AI tool usage — professional devs	51%	[12]
GitHub Copilot market share (agent users)	68%	[12]
Developers who trust AI output accuracy	33% (down from 40% in 2024)	[12]
Developers who distrust AI output accuracy	46%	[12]
Struggle with 'close but wrong' AI outputs	66%	[12]
Debugging AI code takes longer than writing	45%	[12]
Positive sentiment toward AI tools	60% (down from 70%+ in 2023–2024)	[12]
Top deal-breaker for adopting new tech	Security / privacy concerns (Rank 1)	[12]
No plans to adopt AI agents	38%	[12]

IX. SECURITY IMPLICATIONS AND ETHICAL CONSIDERATIONS

Beyond the privacy risk of embedding inversion[11]—already established in Section IV—codebase indexing raises a broader set of security and ethical concerns that are directly relevant to deployment decisions. These concerns are not speculative: they are reflected in the developer community's primary adoption concerns[12] and in emerging regulatory frameworks governing AI systems in software development.

A. Data Residency and Regulatory Compliance

Cloud-hosted persistent indices—such as those maintained by GitHub Copilot Enterprise[1] and Amazon Q Developer[3]—store code embeddings and, in some cases, raw code fragments on remote infrastructure. For organizations subject to the EU General Data Protection Regulation (GDPR), the US Health Insurance Portability and Accountability Act (HIPAA), or SOC 2 Type II requirements, this creates a concrete compliance obligation. Source

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

code is frequently classified as sensitive proprietary data under these frameworks, and storing it—or derivations of it, including embeddings—on external servers may constitute unauthorized data processing absent explicit contractual data processing agreements.

The Morris et al.[11] finding that 92% of text inputs can be reconstructed from stored embeddings removes any defense based on the argument that embeddings are not 'real' source code. Regulators increasingly apply the concept of 'personal data' broadly to include representations from which the original data can be inferred or reconstructed; the Vec2Text method[11] provides a practical mechanism for such reconstruction. Organizations must therefore evaluate persistent indexing solutions under the same data governance frameworks they apply to the raw source code itself.

B. Supply Chain and Index Poisoning Risks

A security concern unique to persistent indices—not present in ephemeral systems—is the risk of index poisoning. If an attacker gains write access to a persistent vector store (via compromised credentials, a malicious dependency, or a compromised CI/CD pipeline that triggers re-indexing), they can inject adversarial embeddings that cause the retrieval system to return malicious or misleading code fragments in response to legitimate developer queries. The developer sees a suggestion that appears contextually appropriate—retrieved from their own codebase's index—but contains injected vulnerabilities or backdoors.

This attack surface does not exist in ephemeral systems, where context is assembled directly from the filesystem at query time. The filesystem itself can be compromised, but the attack path is the same as for any code injection attack and is covered by existing security practices. Index poisoning represents a novel attack vector introduced specifically by persistent indexing architectures. As of 2025, no published study has formally characterized the attack surface or evaluated defenses, making this an open security research question directly relevant to the deployment of persistent indexing tools.

C. Intellectual Property and Ownership of Indexed Representations

Persistent indices raise unresolved questions about intellectual property ownership. When a tool vendor hosts a persistent index of an organization's codebase on cloud infrastructure, the legal ownership of the resulting embeddings is ambiguous. Embeddings are derived works: they are computed from the source code by a process owned by the tool vendor. If the vendor's terms of service claim rights to derived data—a common pattern in consumer AI agreements—organizations may inadvertently transfer partial rights to proprietary code representations.

Ephemeral indexing sidesteps this entirely: no persistent derived representation is created, so no ownership question arises. Cursor's architecture[2] sends only embeddings (not raw source code) to its cloud—but the embeddings themselves are stored on Cursor's servers (Turbopuffer), which still raises IP ownership questions under vendor terms of service. For organizations requiring zero external IP exposure, ephemeral tools with fully local execution—such as Continue.dev with a self-hosted model—represent the optimal risk profile: maximum privacy with the trade-off of reduced cross-file context accuracy.

D. Ethical Dimensions: Transparency and Explainability

The Stack Overflow 2025 survey finds that security and privacy are the primary rejection criteria for developer tools (Rank 1),[12] and that only 33% of developers trust AI output accuracy.[12] These figures suggest that the developer community is applying an implicit explainability standard: they want to understand why a suggestion was made and be able to verify it. Persistent indexing systems—particularly those with semantic search capabilities like Cursor's "@Codebase" feature[2]—can in principle expose their retrieval reasoning by surfacing the specific code fragments retrieved to generate a suggestion. This 'show your work' capability is not available in ephemeral systems, where the context assembly process is opaque.

Explainable retrieval is therefore both a trust-building mechanism and a quality verification tool. When a developer can see exactly which cross-file fragments were retrieved to inform a suggestion, they can identify cases where the index is outdated, misconfigured, or retrieving from the wrong context. This transparency directly addresses the '66% close-but-wrong' failure mode[12] by enabling developers to diagnose retrieval quality rather than simply accepting or rejecting the generated output. Advancing explainable retrieval in persistent indexing systems is therefore both an ethical requirement—supporting the developer's right to understand AI-generated code affecting their work—and a practical quality-improvement mechanism.

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

The broader ethical framework for AI in software development is still emerging. Augment Code became the first AI coding assistant to achieve ISO/IEC 42001 certification for AI management systems in 2025—a signal that formal ethical and governance standards are beginning to be applied to this domain. Organizations deploying persistent indexing solutions should anticipate that regulatory requirements around explainability, data governance, and bias disclosure will progressively apply to AI coding tools, and that architectural choices made today—regarding where indices are stored, who controls them, and how retrieval decisions are explained—will have compliance implications in the near future.

X. CONCLUSION

This paper presented a comparative analysis of persistent and ephemeral codebase indexing in AI programming tools, grounded exclusively in verified empirical evidence. Three independently peer-reviewed findings anchor the analysis: (1) Ding et al.[6] establish a 33.94% relative improvement in code completion Exact Match accuracy when cross-file context—the defining capability of persistent indexing—is provided; (2) Peng et al.[10] document a 55.8% task completion speed increase with AI-assisted coding tools in a controlled experiment; (3) Morris et al.[11] demonstrate that 92% of 32-token text inputs can be reconstructed from stored dense embeddings, establishing a concrete privacy risk for persistent index storage. The central finding is that persistent indexing delivers measurable and significant accuracy improvements (33.94% EM[6]) for multi-file development tasks, but introduces a documented privacy risk (92% reconstruction[11]) that must govern deployment decisions in regulated environments. Ephemeral indexing sacrifices these accuracy gains but eliminates persistent storage entirely. The 55.8% productivity finding[10] supports AI-assisted coding generally, but its single-task experimental scope should be disclosed when cited. Future work should prioritize: (1) code-specific embedding inversion studies to quantify the privacy risk[11] for code rather than general text; (2) extension of the CoCoMIC accuracy findings[6] to additional programming languages and task types; and (3) federated indexing architectures that deliver cross-file context benefits without centralizing code data. These advances will enable more principled, evidence-based choices between persistent and ephemeral strategies as AI coding tools continue to evolve.

REFERENCES

[1] GitHub. (2024). GitHub Copilot Documentation: Context and Codebase Indexing. GitHub Docs. https://docs.github.com/en/copilot
[2] Cursor. (2024). Cursor: The AI-First Code Editor — Codebase Indexing Documentation. Anysphere Inc. https://cursor.com/docs
[3] Amazon Web Services. (2024). Amazon Q Developer: Generative AI-powered assistance for software development. AWS Documentation. https://docs.aws.amazon.com/amazonq
[4] Gauthier, P. (2024). Aider: AI pair programming in your terminal. Open-source project. https://aider.chat
[5] Y. Perwej, Nikhat Akhtar, Devendra Agarwal, "The emerging technologies of Artificial Intelligence of Things (AIoT) current scenario, challenges, and opportunities", Book Title "Convergence of Artificial Intelligence and Internet of Things for Industrial Automation", SCOPUS, ISBN: 978-1-032-42844-4, CRC Press, Taylor & Francis Group, 2024, Link:https://www.taylorfrancis.com/chapters/edit/10.1201/9781003509240-1/emerging-technologiesartificial-intelligence-things-aiot-current-scenario-challenges-opportunities-yusuf-perwej-nikhatakhtar-devendra-agarwal?context=ubx&refId=537f1a8f-6a94-4439-b337-3ad3d1ce8845, DOI: 10.1201/9781003509240-1
[6] Nagarjuna Tandra, Nikhat Akhtar, K Padmanaban, L. Guganathan, "A finite-element dual-level contextual informed neural network with swarm space hopping algorithm based optimal feature selection and detection for EEG-based epileptic seizure detection", Swarm and Evolutionary Computation, Elsevier , SCIE, Volume 97, Pages 1- 19, August 2025, DOI: 10.1016/j.swevo.2025.102072
[7] Lewis, P., Perez, E., Piktus, A. et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.
[8] Y. Perwej, Firoj Parwej, N. Akhtar, "An Intelligent Cardiac Ailment Prediction Using Efficient ROCK Algorithm and K- Means & C4.5 Algorithm", European Journal of Engineering Research and Science (EJERS), Bruxelles, Belgium, ISSN: 2506-8016 (Online), Vol. 3, No. 12, Pages 126 – 134, 2018, DOI: 10.24018/ejers.2018.3.12.989
[9] Y. Perwej, Mohammed Y. Alzahrani, F. A. Mazarbhuiya, Md. Husamuddin, "The State-of-the-Art Cardiac Illness Prediction Using Novel Data Mining Technique", International Journal of Engineering Sciences & Research Technology (IJESRT), ISSN: 2277-9655, Volume 7, Issue 2, Pages 725-739, 2018, DOI: 10.5281/zenodo.1184068
[10] Ding, Y., Wang, Z., Ahmad, W. U., Ramanathan, M. K., Nallapati, R., Bhatia, P., Roth, D., & Xiang, B. (2024). CoCoMIC: Code Completion by Jointly Modeling In-file and Cross-file Context. Proceedings of LREC-COLING 2024, pages 3433–3445,

TEJAS Journal of Technologies and Humanitarian Science
ISSN-2583-5599
Vol.05, I.02 (2026)
https://www.tejasjournals.com/
https://doi.org/10.63920/tjths.52011

Torino, Italia. ELRA and ICCL. [arXiv:2212.10007] — Key finding: 33.94% EM improvement, 28.69% identifier matching improvement.
[11] Kajal, Kanchan Saini, N. Akhtar, Devendra Agarwal, Ms. Sana Rabbani, Y. Perwej, "Machine Learning for the Diagnosis and Prognosis of Chronic Illnesses", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Print ISSN: 2395- 1990 , Online ISSN : 2394-4099, Volume 11, Issue 3, Pages 112-122, May-June -2024, DOI: 10.32628/IJSRSET24113100
[12] Guu, K., Lee, K., Tung, Z. et al. (2020). REALM: Retrieval-augmented language model pre-training. Proceedings of the 37th International Conference on Machine Learning (ICML).
[13] Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 7(3), 535–547. [FAISS library — Facebook AI Research]
[14] Kajal, Neha Singh, Nikhat Akhtar, Ms. Sana Rabbani, Y. Perwej, Susheel Kumar, "Using Emerging Deep Convolutional Neural Networks (DCNN) Learning Techniques for Detecting Phony News", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN: 2456-3307, Volume 10, Issue 1, Pages 122-137, 2024, DOI: 10.32628/CSEIT2410113
[15] N.Akhtar, Kumar Bibhuti B. Singh, Devendra Agarwal, Y. Perwej, "Improving Quality of Life with Emerging AI and IoT Based Healthcare Monitoring Systems", International Journal of Scientific Research in Computer Science, Engineering and Information Technology, ISSN: 2456-3307, Volume 11, Issue 1, Pages 96-107, January 2025, DOI: 10.32628/CSEIT2514551
[16] Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836. [HNSW algorithm]
[17] Y. Perwej, "The Bidirectional Long-Short-Term Memory Neural Network based Word Retrieval for Arabic Documents", Transactions on Machine Learning and Artificial Intelligence (TMLAI), which is published by Society for Science and Education, United Kingdom (UK), ISSN 2054-7390, Volume 3, Issue 1, Pages 16 - 27, 2015, DOI: 10.14738/tmlai.31.863
[18] Shweta Pandey, Rohit Agarwal, Sachin Bhardwaj, Sanjay Kumar Singh, Y. Perwej, Niraj Kumar Singh, "A Review of Current Perspective and Propensity in Reinforcement Learning (RL) in an Orderly Manner", the International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), Volume 9, Issue 1, Pages 206-227, 2023, DOI: 10.32628/CSEIT2390147
[19] Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv:2302.06590. — Key finding: 55.8% faster task completion (95% CI: 21–89%), N=95 developers, single HTTP server task.
[20] Morris, J. X., Kuleshov, V., Shmatikov, V., & Rush, A. M. (2023). Text embeddings reveal (almost) as much as text. Proceedings of EMNLP 2023, pages 12448–12460, Singapore. DOI: 10.18653/v1/2023.emnlp-main.765. [arXiv:2310.06816] — Key finding: 92% of 32-token text inputs exactly reconstructed from embeddings. Tested on Wikipedia and MSMARCO (clinical notes); code-specific applicability is a reasonable inference, not directly tested.
[21] N. Akhtar, Kumar Bibhuti B. Singh, Devendra Agarwal, Y. Perwej, "Improving Quality of Life with Emerging AI and IoT Based Healthcare Monitoring Systems", International
Journal of Scientific Research in Computer Science, Engineering and Information Technology
(IJSRCSEIT), ISSN: 2456-3307, Volume 11, Issue 1, Pages 96-107, January 2025, DOI: 10.32628/CSEIT2514551
[22] Y. Perwej, "An Evaluation of Deep Learning Miniature Concerning in Soft Computing", International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), ISSN (Online): 2278-1021, ISSN (Print): 2319-5940, Volume 4, Issue 2, Pages 10 - 16, 2015, DOI: 10.17148/IJARCCE.2015.4203
[23] Stack Overflow. (2025). Stack Overflow Developer Survey 2025. 49,009 responses from 177 countries. Published July 29, 2025. https://survey.stackoverflow.co/2025/
[24] Y. Perwej, Firoj Parwej, "A Neuroplasticity (Brain Plasticity) Approach to Use in Artificial Neural Network", International Journal of Scientific & Engineering Research (IJSER), France , ISSN 2229 – 5518, Volume 3, Issue 6, Pages 1- 9, 2012, DOI: 10.13140/2.1.1693.2808
[25] Venkata K. S. Maddala, Dr. Shantanu Shahi, Yusuf Perwej, H G Govardhana Reddy, "Machine Learning based IoT application to Improve the Quality and precision in Agricultural System", European Chemical Bulletin (ECB), ISSN: 2063-5346, SCOPUS, Hungary, Volume 12, Special Issue 6, Pages 1711 – 1722, May 2023, DOI: 10.31838/ecb/2023.12.si6.157