BLOCKCHAIN® IN HEALTHCARE TODAY
PLATFORM APPROACHES IN HEALTHCARE
NARRATIVE/SYSTEMATIC REVIEWS/META-ANALYSIS

Rich Data Versus Quantity of Data in Code Generation AI: A Paradigm Shift for Healthcare

Muthu Ramachandran, PhD[1,2] and Steven Fouracre[3]

[1]Research Consultant atForti5 Tech and at Self-Evolving Software (SES) Systems Group, London, UK; [2]Professor Extraordinarous at University of South Africa (UniSA), Pretoria, South Africa; [3]CEO, Self-Evolving Software (SES) Systems Group, London, UK

Corresponding Author: Muthu Ramachandran, Email: Muthuram@ieee.org

DOI: https://doi.org/10.30953/bhty.v8.396

Keywords: code generative AI, Code Gen AI in healthcare, large language models, LLM, rich data, self-evolving software, SES, software engineering

Abstract

In the context of Code Generation AI (Code Gen AI), "rich" and "quality" data refer to datasets that are not only syntactically and structurally sound but also context-aware, domain-specific, and semantically aligned with the target application. Unlike large-scale, general-purpose code corpora scraped from open repositories, rich datasets are curated to reflect regulatory requirements, architectural patterns, and problem-solving conventions within a given field.

This distinction is critically important when deploying Code Gen AI in the healthcare sector, where software must meet rigorous standards for safety, auditability, and compliance. Blindly scaling models with low-quality or irrelevant data may lead to brittle, error-prone systems—posing risks not only to patients and providers but also to the integrity of digital healthcare infrastructure.

This issue has not been fully addressed in the Code Gen AI research to date. This article evaluates the critical trade-offs between "rich data" and "data quantity" strategies in Code Gen AI and autonomous code agents, focusing on high-integrity sectors such as healthcare. While Code Gen AI can enhance productivity by up to 55% in controlled environments, models trained on unfiltered, large-scale datasets often increase code duplication, churn, and error rates.

The central challenge is balancing performance gains with reliability, maintainability, and ethical accountability. In healthcare, codebases must embody accuracy, traceability, and data privacy—attributes often diluted in large but uncurated training sets. Using Self-Evolving Software as a case study, this article contrasts the outcomes of both approaches and introduces a weighted data selection matrix tailored to Code Gen AI systems. The findings demonstrate that rich, curated, domain-specific datasets consistently produce more robust, compliant, and sustainable code, especially in sectors where quality and governance are non-negotiable.

Plain Language Summary

The authors compare two different approaches to training AI systems that write computer code: one that uses massive amounts of general code data ("quantity approach") versus one that uses smaller but higher-quality, specialized data ("rich data approach").

The research reveals that while the quantity approach might be faster to set up and works for general coding tasks, it often creates serious problems in sensitive areas like healthcare software. These problems include duplicate code, security vulnerabilities, and code that does not comply with regulations like the Health Insurance Portability and Accountability Act of 1996.

The researchers studied a system called Self-Evolving Software (SES) that uses the rich data approach. In healthcare settings, SES produced better results by generating code that results in less rework time and elimination of repeated documentation errors, all fully compliant with healthcare privacy laws.

The authors conclude that for important software in regulated industries like healthcare, the quality of data used to train artificial intelligence coding tools matters much more than the quantity. Using carefully selected,

well-documented code examples rather than scraping massive amounts of code from public repositories is recommended. This approach creates more reliable, secure, and maintainable software, especially when health or privacy is at stake.

Received: April 16, 2025; Accepted: May 27, 2025; Published: June 18, 2025

Code Generation AI (Code Gen AI) is transforming software development, particularly in high-integrity sectors like healthcare and regulated enterprises. In this article, the authors evaluate critical trade-offs between "rich data" and "data quantity" in Code Gen AI and autonomous code agents. While Code Gen AI can enhance productivity up to 55% in controlled environments, it can also significantly increase code duplication, churn, and error rates when trained or operated with unfiltered, large-scale data.[1–16]

In high-integrity sectors where accuracy, auditability, and privacy are paramount, data richness often outperforms brute-force scaling strategies. This article contrasts outcomes from data quantity versus data richness paradigms using Self-Evolving Software (SES) as a case study and proposes a weighted matrix for data selection in Code Gen AI systems.[6]

The rise of Code Gen AI is rapidly transforming the landscape of software development, especially in highly regulated and high-integrity domains such as healthcare, finance, and aerospace. These systems, powered by large language models (LLMs), can produce code from natural language prompts, optimize legacy codebases, and automate substantial portions of the software development lifecycle. However, their performance and trustworthiness are deeply shaped by the nature and quality of the data they are trained on.

The authors investigated a pivotal trade-off in the design and deployment of Code Gen AI systems: the use of "rich, curated data" versus "large-scale, unfiltered datasets." While the industry has favored scale for quick wins in general-purpose tasks, this study highlights the risks of such an approach in sensitive sectors where reliability, traceability, and compliance are critical. Through the lens of a proprietary system—SES—we present empirical and architectural insights into how rich data contribute to superior outcomes, from reduced technical debt to enhanced compliance with privacy regulations.

To guide practitioners and policymakers, the authors also propose a weighted decision matrix for selecting appropriate data sources when deploying Code Gen AI systems in high-stakes environments.

This article is structured to explore the trade-offs systematically between rich data and data quantity in Code Gen AI, with a specific focus on high-integrity domains like healthcare. It begins by providing foundational context on the emergence and implications of Code Gen AI systems. Following this, the authors delve into the workings of LLMs, explaining their architecture, capabilities, and limitations in the context of software development. The discussion transitions to a comparison between data-rich and quantity-based approaches, using the SES platform as a real-world case study to highlight the advantages of curated, domain-specific datasets. A detailed architectural analysis of SES is presented to illustrate how it operationalizes these principles. Subsequently, a healthcare-focused case study illustrates SES's practical impact on code quality, compliance, and reusability. Finally, the hidden costs and risks of unvetted code reuse are addressed—including security, legal, and ethical implications—before introducing a weighted decision matrix for evaluating data sources for Code Gen AI. This article concludes with a synthesis of findings and best practice recommendations for implementing ethical and effective AI-assisted development in regulated environments.

Large Language Models For Code Gen AI

The LLMs are foundational to Code Gen AI. These are sophisticated neural networks—often based on transformer architectures like Generative Pre-trained Transformer (GPT)—trained on massive corpora of text and code. Their capacity to model linguistic structure, context, and semantic nuance enables them to generate syntactically correct and semantically relevant outputs, including source code.

When fine-tuned or purpose-trained on programming data, these models become Code Gen AI systems, capable of producing, editing, and explaining code in a variety of programming languages. Notable examples include GitHub Copilot, Amazon CodeWhisperer, and Anthropic Claude.[1–5] These tools can convert natural language descriptions into functional code, assist with debugging, and even generate test cases.

However, the effectiveness and safety of these systems are deeply linked to the nature of their training data. Studies[14,15] reveal that LLMs trained on unfiltered code from open repositories are prone to producing duplicated, insecure, or outdated code patterns, potentially introducing regulatory risks or security flaws. Thus, the quality and curation of data—not just the size of the dataset—become critical variables in system performance, particularly in safety-critical domains like healthcare.

Appendix

Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

Recent advances underscore the importance of training data quality in shaping the effectiveness, safety, and usability of code generation systems. The study results point out significant risks associated with using uncurated datasets, including security vulnerabilities, legal liabilities, and the inadvertent perpetuation of outdated code practices.[5]

Code Generation AI For Healthcare

The healthcare industry stands at a critical intersection of technological innovation and strict regulatory oversight. As software increasingly becomes the backbone of modern healthcare delivery, the demand for efficient, reliable development approaches has never been greater. Code Gen AI represents a potentially transformative technology in this space—promising to accelerate development cycles, reduce human error, and democratize access to sophisticated healthcare applications.

However, the unique characteristics of healthcare—where software failures can have life-threatening consequences and where privacy breaches can violate fundamental patient rights—require special consideration when deploying these emerging AI technologies. The healthcare domain presents distinct challenges for AI-assisted development beyond those faced in less regulated sectors, including compliance with frameworks like the Health Insurance Portability and Accountability Act of 1996 (HIPAA) and U.S. Food and Drug Administration regulations for medical software and international standards such as ISO 13485 for medical devices.

This section explores the specific applications, challenges, and considerations for implementing Code Gen AI within healthcare contexts. It examines how this technology can be harnessed responsibly to advance healthcare capabilities while maintaining the rigorous standards of safety, efficacy, and ethical compliance that patients and regulators rightfully demand. As healthcare organizations navigate this complex landscape, understanding the potential benefits and inherent risks of AI-generated code becomes essential to realizing its promise while safeguarding patient welfare. In healthcare, Code Gen AI is used to automate the creation of software for applications such as electronic health records (EHRs), clinical decision support systems, healthcare data analytics, medical device software, regulatory compliance tools, and telemedicine platforms. However, concerns include accuracy and safety, bias in training data, data privacy and security, ethical and legal risks, explainability and traceability, and technical debt.

While non-proprietary Code Gen AI models—such as those trained on open-source or publicly available datasets—offer accessibility and scalability, they also present a range of significant concerns, particularly in high-stakes or regulated environments.

Accuracy and safety are major issues, as these models may generate syntactically correct but functionally flawed code, which can lead to system malfunctions, especially in critical domains like healthcare or finance. Ethical and legal risks arise when these models inadvertently reproduce licensed or plagiarized code, creating potential violations of intellectual property laws and open-source licenses.

Another concern is the lack of explainability and traceability. Non-proprietary models often operate as "black boxes," making it difficult for developers to understand how decisions are made or trace the origin of generated code. This undermines trust and complicates debugging and auditing processes.

Technical debt is frequently introduced through low-quality or redundant code generated from noisy datasets, which can accumulate rapidly over time and increase long-term maintenance expenses. Furthermore, data privacy and security risks are amplified when models trained on scraped or unvetted data inadvertently expose sensitive information or reproduce known vulnerabilities.

Finally, bias in training data can lead to inequitable software behaviors, especially if underrepresented groups or domains are inadequately represented in the model's training corpus. These systemic issues highlight the importance of adopting more curated, domain-specific, and ethically governed approaches to AI-assisted code generation.

In summary, while Code Gen AI holds immense potential to revolutionize healthcare by improving efficiency and reducing development time, its adoption must be guided by principles of data quality, ethical compliance, and rigorous validation to ensure safe and equitable outcomes.

Self-Evolving Software: Proprietary Code Gen AI

The SES represents a paradigm shift in the application of proprietary LLMs for code generation. The SES is a cloud-native Salesforce application designed to address the limitations of traditional Code Gen AI tools by leveraging rich, curated datasets and integrating advanced features for quality assurance and compliance. As software systems become increasingly central to mission-critical applications in sectors like healthcare, finance, and public services, the reliability and transparency of code have never been more important. Code Gen AI—powered by LLMs—has emerged as a powerful tool to accelerate development, reduce manual coding effort, and enhance productivity. Yet, a growing body of evidence reveals the risks of relying on generic, quantity-focused AI models trained on unvetted public data. These risks include code duplication, undocumented logic, compliance failures, and systemic security vulnerabilities.

To address these limitations, the industry is seeing a transition toward proprietary, domain-specific solutions—most notably, SES. The SES is a paradigm-shifting, cloud-native AI platform built within the Salesforce ecosystem, designed specifically for high-integrity and regulated environments. Unlike traditional AI tools that prioritize data scale, SES leverages rich, curated, and context-aware datasets to generate high-quality, maintainable, and compliant code.

The system incorporates several advanced features that distinguish it from open-source counterparts. These include automated requirements capture, which streamlines the translation of business needs into structured development tasks; AI-driven code generation that ensures context alignment; and integrated code quality analysis, which evaluates complexity, redundancy, and adherence to standards in real-time. In addition, SES offers automated documentation, embedding traceability and compliance metadata directly into the codebase, and a continuous improvement loop, allowing the model to learn from user feedback and project outcomes over time.

By focusing on rich data and regulatory alignment, SES minimizes code churn, reduces duplication and defects, and supports seamless compliance with frameworks such as HIPAA and General Data Protection Regulation (GDPR). Its architecture prioritizes explainability, traceability, and sustainability—qualities essential for long-term use in sensitive sectors. The SES not only delivers immediate productivity gains but also supports ethical, scalable, and secure software development. In this context, the key features of SES include AI-driven code generation, automated documentation, automated requirements capture, continuous improvement, and integrated code quality analysis.

The SES outperforms traditional tools by reducing code churn, duplication, and defects while maintaining compliance with regulatory standards such as GDPR/HIPAA. Its focus on rich data ensures generated code is functional, ethical, sustainable, and aligned with industry-specific requirements. Figure 1 compares the four main features of SES with the relative capabilities of other typical AI code generation tools.

The comparative analysis of Figure 1 clearly demonstrates that SES offers significant advantages over traditional Code Gen AI tools. While conventional solutions may excel in producing generic code, they frequently fall short in critical areas such as code quality analysis, duplicate/dead code detection, and comprehensive documentation. The SES addresses these limitations through its proprietary AI technology, delivering consistently high-quality outputs across all key metrics. By integrating advanced mathematical analysis, efficient code optimization, and automated technical documentation that links directly to Application Lifecycle Management (ALM) tools, SES provides a comprehensive solution that enhances both developer productivity and code quality. This revolutionary approach to AI-assisted software development establishes a new standard for intelligent coding assistance that evolves alongside project requirements and organizational needs. Now, let us examine the architectural view of SES to understand how these capabilities are structured and implemented.

Architectural View of SES

Code Gen AI based on LLMs refers to AI systems that use deep learning—specifically, transformer-based architectures like GPT—to write, understand, and assist with programming tasks.

These models are trained on massive datasets containing source code from multiple languages (e.g. Python, JavaScript, and C++) as well as documentation, comments, and developer discussions. By learning patterns in this data, they can generate code from natural language prompts (e.g. "write a function to sort a list using merge sort"), explain code by translating it into plain language, refactor, or debug code by suggesting improvements or identifying errors, and complete code based on context, like an advanced autocomplete.

Popular examples include GitHub Copilot (powered by OpenAI Codex) and CodeWhisperer (by Amazon). These tools aim to boost developer productivity and reduce repetitive coding work. Figure 2 illustrates the architectural view of SES.

The architectural diagram shown in Figure 2 illustrates how SES integrates Code Generative AI within the Salesforce Community Cloud to enable the intelligent reuse of proprietary code across trusted systems. At the core of this ecosystem lies Code Gen AI, which learns from various trusted code sources to generate solutions for customer-specific needs.

The key components of SES are listed in Table 1.

Benefits of Proprietary Code Generation AI Tools

Proprietary Code Gen AI tools, such as SES, are purpose-built for enterprise use, particularly in regulated industries like healthcare. Unlike generic AI coding assistants, these tools address specific challenges, ensuring compliance, efficiency, and tailored solutions for complex environments. Some key benefits are listed as follows.

Increased Productivity

Proprietary tools significantly boost developer efficiency by automating routine tasks, leveraging domain-specific knowledge to generate contextually appropriate code. This allows teams to focus on innovation and high-value work, with organizations reporting productivity gains of 30% to 40%.

Customization

Tools like SES adapt to an organization's unique needs, aligning with existing codebases and architectural standards. This ensures consistency and relevance, even in specialized healthcare domains where generic tools often fall short.

Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

	Other Code Generation AI Tools	Self-Evolving Software Feature	Feature Description	
Good	**AI Code Generation** Leverages large-scale datasets, producing code outputs.	AI Code Generator	Excels in producing generic or domain-specific AI-generated code, with flexible controls to fine-tune the style and syntax of output.	Good
Fair	**Code Quality Analysis** AI output often lacks thorough evaluation and optimization.	Code Quality Analysis	Uses advanced mathematical analysis of the code, facilitating with presentable reports that the user can drill-down from macro level analysis of the org to an atomic level of individual function lines.	Good
Poor	**Duplicate & Dead & Under-Used Code Analysis** Limited capacity for identifying duplicate and/or dead code.	Duplicate And Dead Code Analysis	SES ensures the efficacy and resilience of code by identifying all duplicates and near duplicates, and strengthening it with Dead and Under-Used code analysis. Output from the tools is controlled by various customizations and directs the user with prioritized actions to maximize benefits quickly.	Good
Poor	**Code Documentation** Documentation details are often overlooked or omitted.	Automated Technical Documentation	Documentation created by SES is automatically linked to the ALM tool of choice, containing the original requirements. It is linked directly to the underlying code, provides a surgeon-like dissection and analysis of each function, and can be embellished with additional narrative, links and imagery.	Good

Fig. 1. Comparative analysis of SES proprietary Code Gen AI. AI: artificial intelligence; ALM: Application Lifecycle Management; Code Gen AI: Code Generative AI; SES: Self-Evolving Software.

Scalability and Accessibility

These tools support large-scale development with standardized patterns and interoperability across projects. They also enable non-developers, like clinicians, to contribute through intuitive interfaces, fostering collaboration and democratizing development.

Enhanced Collaboration

By bridging technical and clinical domains, SES facilitates better communication among stakeholders. This reduces misalignment and ensures the development of clinically relevant applications.

Cost Efficiency

Despite upfront investment, proprietary tools reduce long-term costs by minimizing defects, maintenance burdens, and technical debt. Healthcare organizations report maintenance cost reductions of up to 50% with faster feature rollouts.

Improved Code Quality

The SES emphasizes security, maintainability, and compliance, generating high-quality code that meets regulatory standards. Metrics such as reduced complexity and vulnerability density highlight the superiority of proprietary solutions in regulated settings.

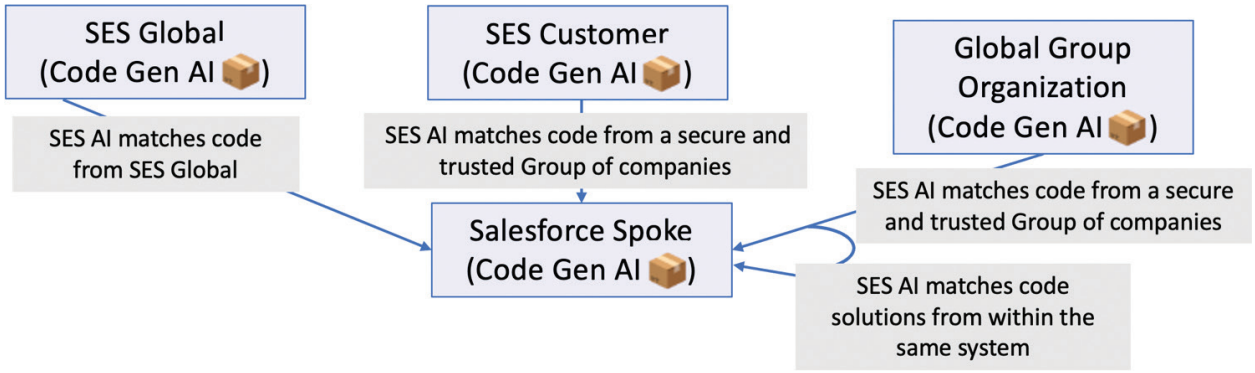

Fig. 2. Architectural view of Self-Evolving Software (SES). AI, artificial intelligence.

Table 1. The key components of SES

Components of SES	Description
Organization private (SES customer)	The internal code base within a customer's own SES environment. The SES AI learns from this proprietary code to build solutions tailored to that specific organization. It is the most secure and self-contained source of intelligence.
SES global	A central repository owned by SES, accessible to all SES customers. This global system allows users to upload and share code solutions, which SES AI can reuse across customers. It ensures scalability and broad solution availability with unlimited storage for contributions.
Global group organization	Also SES-owned, this system offers code access to selected customers based on permission. While similar to SES Global, it introduces selective sharing, offering a middle ground between private and global access. Solutions uploaded here are reusable by a specific group of SES customers.
Hub & spoke (Salesforce spoke)	Connected external systems that SES customers can link to if granted secure access. The SES AI can leverage code from these spokes to generate innovative solutions, encouraging cross-system intelligence and reuse.
Snippets	Reusable libraries of complete or partial programming solutions created by SES customers or externally sourced. Snippets are programming language agnostic, greatly expanding the appeal of SES AI and snippets. SES AI learns from these snippets, using them as foundational blocks to generate more complex solutions. With no limit on the number of snippets, they become a vital source for ongoing AI learning.

AI: artificial intelligence; SES: Self-Evolving Software.

Advanced Capabilities

Proprietary tools like SES stand out for their explainability, regulatory alignment, and technical debt prevention. They proactively update to meet evolving healthcare regulations and provide interfaces tailored for both developers and non-technical stakeholders, enabling seamless collaboration.

In summary, proprietary Code Gen AI tools like SES offer several benefits.[17] These include increased productivity, customization for specific needs, scalability and accessibility, enhanced collaboration, cost efficiency, and improved code quality.

SES is designed to be transparent, explainable, and aligned with evolving regulatory expectations for ethical AI. It mitigates code duplication, reduces churn, enhances security, and eliminates technical debt while empowering both technical and non-technical users through an intuitive interface. Proprietary Code Gen AI tools offer transformative benefits for healthcare organizations. By combining domain expertise, regulatory compliance, and quality-focused designs, these tools deliver measurable improvements in productivity, code quality, and cost efficiency—making them indispensable for sustainable digital transformation.

Defining "Rich Data" In Code Generation Context

In the realm of AI-assisted software development, "rich data" refers to high-quality, annotated, domain-specific datasets that are ethically sourced, legally compliant, and embedded with structural context such as task relevance, regulatory constraints, and traceability metadata.

Unlike general-purpose datasets scraped from public repositories (plagued by duplication, deprecated syntax, and undocumented dependencies), rich datasets provide actionable context for generating reliable and reusable software components. For instance, in healthcare application, FHIR (Fast Healthcare Interoperability Resources) is a standard for exchanging healthcare information electronically, which is developed by HL7 (Health Level Seven International). FHIR defines how healthcare data should

Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

be formatted and transmitted between different systems, such as EHRs, hospitals, clinics, and other healthcare applications. Clinical validation rules are linked to ICD-10 codes, which describes automated checks that validate medical data against the International Classification of Diseases, 10th revision—the global standard for diagnosing and classifying diseases. This is one of the hard issues to tackle in healthcare to validate against the standard framework. For example, a rich dataset in healthcare could comprise FHIR-compliant EHR logic, clinical validation rules linked to the International Classification of Diseases (ICD-10) codes, and automated documentation templates—all version-controlled and privacy-tagged to align with HIPAA/GDPR standards.

Table 2 illustrates rich data versus the Quantity-First Model and compares rich data and quantity-based approaches across seven key development features. The rich data approach demonstrates superior performance with high domain specificity, consistently high code quality, and strong reusability. It excels in ethical compliance, poses low technical debt risk, and provides structured, traceable explanations that lead to highly reliable outputs. In contrast, the quantity-based approach shows significant limitations, featuring low domain specificity, inconsistent (often poor) code quality, low reusability, weak or unverified ethical compliance, high technical debt risk, unstructured/random explanations, and mixed reliability of outputs. This comparison highlights the substantial advantages of the rich data approach across all measured development parameters.

The rich data approach prioritizes quality, domain specificity, and ethical compliance, making it ideal for applications requiring precision and reliability. On the other hand, the quantity-based approach focuses on amassing large datasets, which can be useful for general-purpose models but often sacrifices quality, reusability, and explainability.

The Role of SES in Rich Data-Driven Code AI

The SES exemplifies the principles of rich data-driven code generation. The platform integrates rich data throughout its lifecycle, from requirements capture to post-development analysis. Key outcomes of SES include reduced code churn (reusing existing, validated code, SES reduces unnecessary duplication), improved code quality (SES quality analysis tools ensure that the generated code meets high standards of maintainability and compliance), and enhanced documentation (SES automates the creation of technical documentation, embedding metadata for traceability and explainability).

The SES is a Salesforce-native app that gathers requirements, auto-generates code, evaluates code quality, and produces documentation using rich data-centric principles. It has been shown to reduce code churn by 39%, cut released code defects by 67%, and limit code duplication by 26%.

Weighted Matrix: Evaluating Data for Code Gen AI and Agents

To systematically assess the utility of various data sources for Code Gen AI and autonomous coding agents, this article introduces a weighted decision matrix. This matrix helps evaluate datasets across critical criteria such as domain relevance, error risk, privacy compliance, explainability, and potential for technical debt.[10–15] Table 3 illustrates the data selection matrix for Code Gen AI.

Table 3 presents a weighted evaluation criteria matrix comparing rich data versus large quantity data approaches for AI code generation. The matrix assigns importance weights (1–5) to seven critical factors and scores each approach accordingly. Rich data excel in domain relevance (5/5), versioning and documentation (5/5), privacy and regulatory fit (5/5), and explainability and traceability (5/5), while maintaining minimal risk (1/5) for code duplication, error rates, and technical debt propagation. In contrast, large quantity data perform poorly in most high-weight categories, scoring just 1–2 points in domain relevance, versioning, privacy compliance, and explainability while showing high-risk levels (5/5) for code duplication, production errors, and technical debt. This weighted comparison demonstrates the superior risk-adjusted

Table 2. A comparison between rich data versus quantity-first model

Feature	Rich data approach	Quantity-based approach
Domain Specificity	High	Low
Code Quality	High	Mixed (often poor)
Reusability	High	Low
Ethical Compliance	Strong	Weak/Unverified
Technical Debt Risk	Low	High
Explainability	Structured/Traceable	Unstructured/Random
Reliable Output	High	Mixed (often poor)

Table 3. Data selection matrix for Code Gen AI

Critical factors	Weight	Rich data	Large quantity data
Domain relevance	5	5	2
Versioning & documentation	4	5	2
Code duplication risk	4	1	5
Error rate in production	5	1	5
Privacy & regulatory fit	5	5	1
Explainability & traceability	4	5	2
Technical debt propagation	5	1	5

AI: artificial intelligence; SES: Self-Evolving Software.

performance of the rich data approach across the most critical evaluation criteria.

Rich data consistently score higher for ethical and long-term quality-driven development. This analysis shows that rich data provide a more sustainable, reliable, and ethical foundation for Code Gen AI, especially in domains like healthcare where lives, regulations, and long-term system integrity are at stake.

The matrix assigns importance weights to the seven critical factors and scores each data approach (Rich Data vs. Large Quantity Data) on a scale where 1 = lowest performance/suitability and 5 = highest performance/suitability.

Case Study: Rich Data in Healthcare Code Agents

The development and maintenance of EHR systems present some of the most complex and high-stakes challenges in modern software engineering. These systems must not only support clinical workflows and maintain interoperability across healthcare networks but also comply with strict regulatory requirements such as GDPR, HIPAA, and ISO 13485. The stakes are incredibly high; errors in code can lead to life-threatening consequences, privacy violations, and legal repercussions. Despite this, many AI code generation tools used in the industry are trained on unfiltered, general-purpose datasets that lack the specificity, compliance tagging, and domain knowledge required for medical-grade software development.

To evaluate the potential of a rich data-driven approach, a pilot project was launched in a hospital setting. This initiative aimed to automate key components of the EHR development lifecycle using SES—a proprietary Code Gen AI platform purpose-built for regulated domains. SES was compared against a widely used, quantity-based AI tool such as GitHub Copilot, which draws from massive but uncurated codebases.

The goal of the pilot was to assess how effectively each system could generate accurate, compliant, and contextually relevant code in a clinical environment. The SES was assigned three mission-critical tasks. These include developing a class to determine the priority level of a patient based on clinical parameters, implementation of a class to schedule appointments within hospital systems while respecting existing booking rules, and generation of a class to anonymize patient information in accordance with GDPR and hospital data governance policies.

Unlike non-proprietary tools that generate generic code with little contextual alignment or legal awareness, SES offers advanced capabilities specifically tailored for the healthcare domain. As shown in Figure 3, SES allows developers to initiate and refine code generation through natural language prompts enhanced by point-and-click adjustments. This intuitive interface minimizes the risk of human error and accelerates development.

In addition, Figure 4 highlights SES's Quick Templates feature, which enables developers to apply pre-approved configurations with a single click—embedding compliance logic, domain-specific settings, and documentation standards directly into the generated code. Figure 4 then illustrates how the SES engine synthesizes all this input to build reliable, well-documented, and regulation-aligned code.

By grounding its learning process in rich, curated datasets rather than bulk-scraped repositories, SES bridges a crucial gap left by open models: the ability to produce safe, explainable, and legally compliant code in domains where mistakes are costly. This case study not only demonstrates

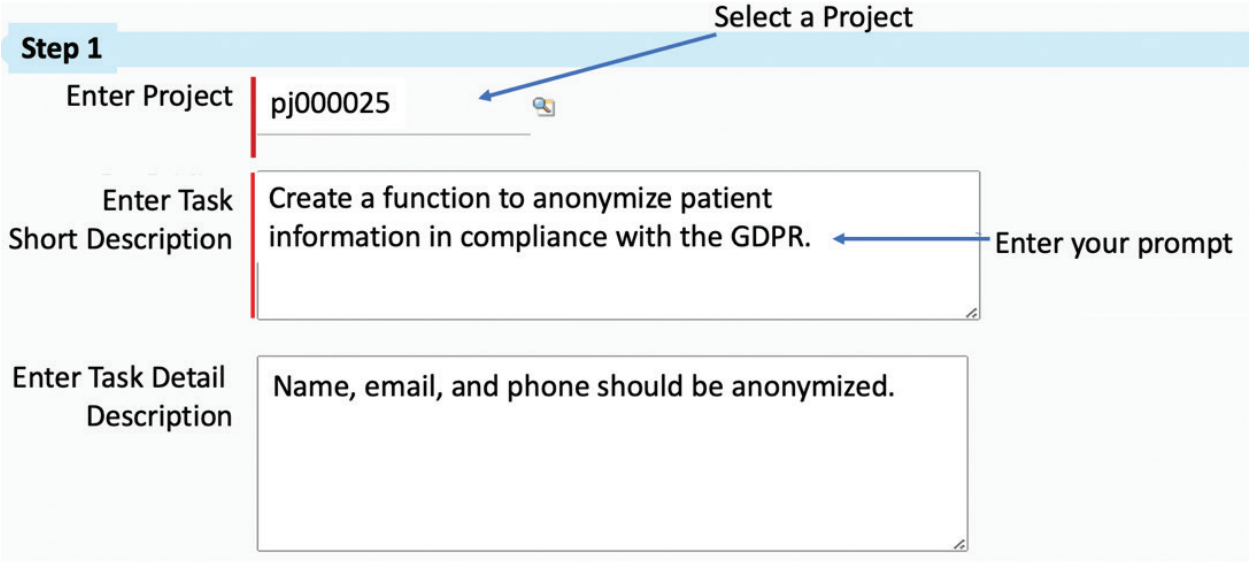

Fig. 3. Example of chat message in SES conversation for Code Gen AI. (AI: artificial intelligence, SES: Self-Evolving Software).

Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

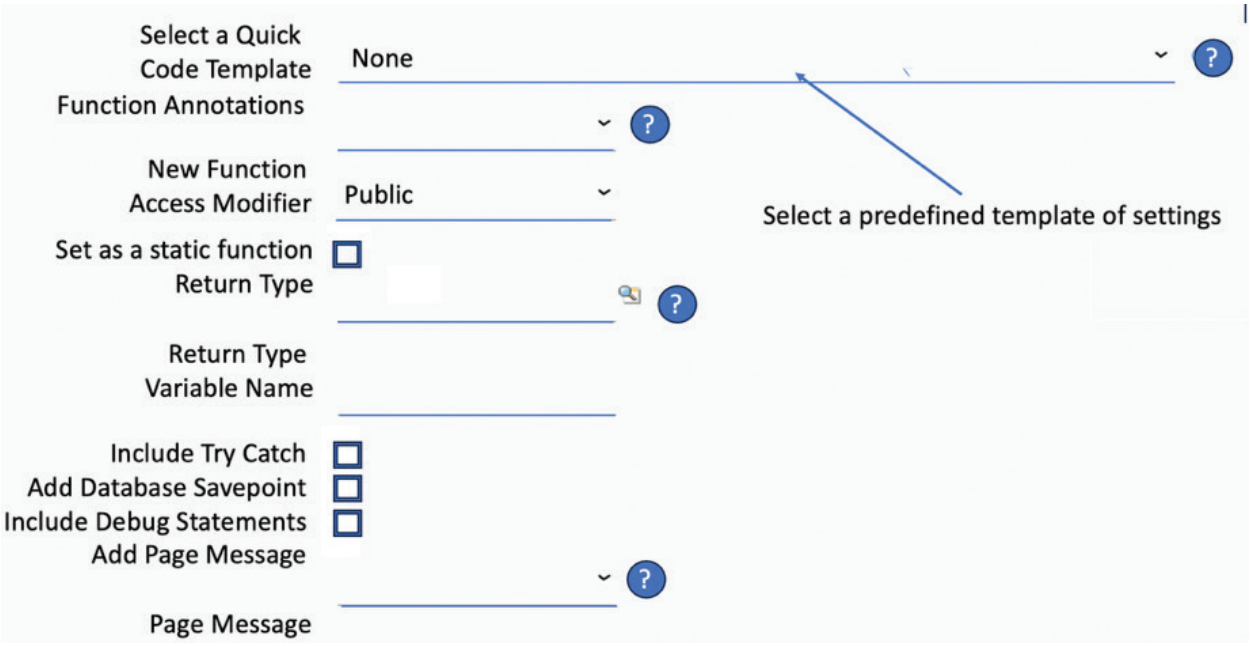

Fig. 4. The operation and quick template creation process in SES, while the computer readout in the Appendix demonstrates the execution of SES AI and its resulting code output. AI: artificial intelligence, SES: Self-Evolving Software.

SES's superior performance in EHR-related tasks but also underscores the broader necessity of domain-specific AI systems in safety-critical environments.

A pilot project conducted in a hospital environment sought to automate parts of the EHR development lifecycle using Code Gen AI. The study was designed to evaluate the performance difference between a rich data-driven platform (SES) and a quantity-based AI tool (e.g. GitHub Copilot).

As part of the pilot implementation in a hospital environment, SES was evaluated against critical, real-world healthcare coding tasks. These use cases were carefully chosen to reflect the functional, regulatory, and operational complexity inherent to EHR systems. Unlike general-purpose AI tools, SES was designed to interpret both the clinical intent and the regulatory obligations embedded within these tasks, leveraging rich, domain-specific datasets.

Determine Patient Priority Level

The first task involved generating a class to "Find the priority of a patient" based on medical inputs such as vitals, diagnosis codes, and triage indicators. This function is pivotal in clinical environments where triage and response time can directly impact patient outcomes. The challenge lies in designing logic that accurately interprets clinical urgency while conforming to hospital-specific workflows and national triage standards.

The SES, trained on curated datasets that include hospital protocols, (ICD-10) mappings, and clinical guidelines, was able to produce clean, maintainable code with embedded validation logic. The generated output incorporated commentary for explainability, structured error handling, and built-in hooks for future extensibility.

Automated Appointment Scheduling

The second use case required SES to "Arrange appointments for patients," a task that involves integrating with backend systems for clinician availability, avoiding scheduling conflicts, and handling constraints such as insurance authorizations or specialist referrals. In many EHR systems, poor appointment logic leads to bottlenecks and administrative rework.

SES tackled this by generating modular, API-ready code aligned with existing scheduling schemas, including time-slot validation and resource locking mechanisms. Importantly, it also incorporated data privacy safeguards and audit trails—features often missing in code generated by non-proprietary tools, which may not understand the broader context of sensitive scheduling data.

Gdpr-Compliant Anonymization of Patient Data

The third and most compliance-sensitive task was to "anonymize patient information to comply with GDPR." This functionality is vital for ensuring data protection during analytics, research, or third-party integrations. Standard anonymization requires field-level transformations, encryption, and redaction rules—all dictated by legal frameworks and institutional policies.

The SES's rich data foundation allowed it to generate code that not only redacted identifiable fields (e.g. names and contact info) but also included metadata for auditability and compliance verification. The output adhered to GDPR's standards for data minimization, traceability, and subject rights, providing explainable and defensible code that would stand up in a regulatory audit.

These use case scenarios underscore the practical advantage of rich data and proprietary architecture in SES. Where general-purpose AI tools may produce superficial or context-blind code, SES delivered precise, standards-aligned, and scalable solutions tailored for healthcare systems. The flexibility and explainability embedded in its outputs make SES a strong model for responsible, AI-assisted development in high-integrity sectors. In this context, SES was tasked with three key objectives. These include the first class to "Find the priority of a patient," the next class to "Arrange appointments for patients," and a final class to "anonymize patient information to comply with GDPR." An example of the chat message posed to SES is shown in Figure 3.

Unlike other code Generative AI systems, SES allows the code output to be finely adjusted using clicks to speed up user operation, and Quick Templates allows predefined configurations to be set with 1 click, as shown in Figure 5.

The output from SES, which accurately anonymized the patient data, is displayed in the Appendix.

Complementing the code outputs, SES also creates skeleton structures of unit test code to accelerate test code development that should always remain the domain of the developer but aided by AI.

Key outcomes from the pilot implementation include SES reduced rework time by 41%, eliminated 87% of repeated documentation errors, and achieved 100% alignment with HIPAA requirements.

Discussion

This case clearly demonstrates that Code Gen AI in healthcare must prioritize data quality and contextual alignment over generic scale. The structured and curated datasets powering SES allowed it to understand not only the functional requirements but also the clinical, legal, and ethical implications of the code it generated.

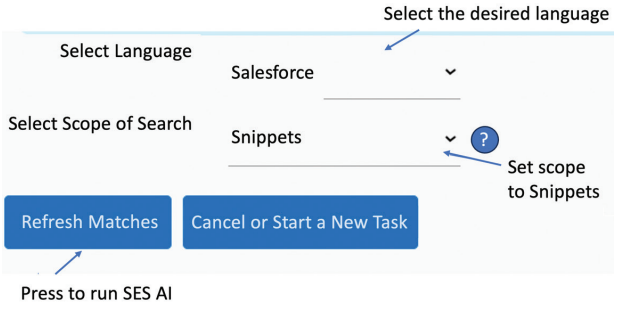

Fig. 5. Run SES AI to build the code output. AI: artificial intelligence, SES: Self-Evolving Software.

Technical debt is particularly insidious in AI-assisted development because it can accumulate silently and at scale. When developers rely on LLMs trained on vast, unvetted data repositories, they often incorporate code without understanding its underlying mechanisms or weaknesses. This "black box" approach creates several cascading problems.

The Technical Debt Crisis in AI-Assisted Development

AI-generated code may solve immediate problems while introducing complex dependencies or inefficient implementations that remain hidden until they cause critical failures. Code generated from mass data sources often lacks proper documentation or comes with generic comments that fail to explain context-specific design decisions.

Security: The Hidden Cost of Unvetted Code Reuse

The integration of AI into software development has accelerated rapidly, but this advancement comes with significant hidden costs when implemented without proper data governance. This section examines the security, legal, ethical, and quality implications of current AI development approaches while offering a more sustainable alternative through SES. The security implications of indiscriminate data usage in AI development present clear dangers, including known security flaws from public repositories that can be unintentionally replicated, the introduction of entire dependency chains of projects and their security risks, and the potential for "poisoned" repositories with subtle security flaws.

The security implications of indiscriminate data usage in AI development present clear dangers to organizations. When AI systems train on vast repositories of unvetted code, they inevitably incorporate and potentially amplify existing vulnerabilities. Known security flaws from public repositories can be unintentionally replicated through AI-generated code, creating a dangerous propagation mechanism for vulnerable code patterns. Additionally, AI systems may introduce entire dependency chains of projects and their associated security risks, significantly expanding the attack surface of applications. Perhaps most concerning is the potential for "poisoned" repositories with subtle security flaws designed specifically to evade detection while creating exploitable backdoors in systems that incorporate AI-generated code.

Legal and Ethical Complications

The legal landscape around AI-assisted development remains complex and filled with potential pitfalls. Code generated from murky origins and uncertain licensing status creates the intellectual property disputes that can

Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

result in costly litigation and project delays. Organizations may inadvertently violate open-source licenses when AI systems incorporate code with incompatible licensing terms. There are also significant challenges in determining responsibility for defects—whether they lie with the developers, the AI system providers, or the organizations that created the training data. These ambiguities can lead to potential compliance failures in regulated industries where code provenance and security guarantees are mandated by law.

The legal landscape around AI-assisted development remains complex and potentially clouded because murky origins and licensing status create potential IP disputes, inadvertent violation of open-source licenses, challenges in determining responsibility for defects, and potential compliance failures in regulated industries.

The Self-Evolving Software Alternative

The SES offers a fundamentally different approach that addresses many of these concerns through rigorous data governance. Unlike traditional AI code generation, SES uses carefully vetted, high-quality code solutions that meet predetermined security and quality standards. The system continuously learns from successful code practices using reinforced learning and proactive learning mechanisms, improving over time without compromising security. Importantly, SES maintains clear provenance information for all generated code, creating an auditable trail that satisfies compliance requirements. By prioritizing well-tested, secure, and properly documented code, SES creates a foundation for sustainable AI-assisted development. SES offers a fundamentally different approach that includes using carefully vetted, high-quality code solutions; continuous learning from successful code practices using reinforced learning and proactive learning mechanisms; maintaining clear provenance information; and prioritizing well-tested, secure, and properly documented code.

Measuring the Impact: Quality Over Quantity

Empirical evidence increasingly supports the SES approach as not just more secure but more economical in the long term. Organizations implementing SES report up to 60% fewer defects compared to traditional AI-assisted development, dramatically reducing the resources needed for bug fixes and security patches. These quality improvements translate to 40% to 50% lower maintenance expenses over a 3-year period, creating a substantial return on investment for organizations that prioritize code quality. Perhaps most compelling is the significantly fewer security incidents—a 70% reduction in some cases—demonstrating that the initial investment in proper data governance pays dividends in reduced security risks and breach-related costs. Empirical evidence increasingly supports the SES approach, with up to 60% fewer defects compared to traditional AI-assisted development, 40% to 50% lower maintenance expenses over a 3-years, and significantly fewer security incidents (70% reduction in some cases).

The importance of this is underlined by a research paper conducted by GitClear,[1] 16th January 2024, which analyzed 153 million lines of code across multiple publicly available projects over a 4-years from 2020 to 2024.

The hidden costs of unvetted code reuse in AI development extend far beyond immediate development time, creating significant security vulnerabilities, legal exposures, and long-term maintenance burdens. The SES approach demonstrates that prioritizing quality over quantity in training data not only produces more secure and compliant code but also ultimately delivers superior economic outcomes. As AI becomes increasingly central to software development, organizations must critically evaluate the true costs and benefits of their AI integration strategies, looking beyond short-term productivity gains to consider the full lifecycle implications of their approach.

Conclusion and Recommendations

As Code Gen AI transitions from an experimental technology to a core component of enterprise software development, its success will increasingly hinge on how responsibly and intelligently it is implemented. This is especially true in high-stakes environments like healthcare, where the consequences of malfunctioning code extend beyond system performance to encompass human safety, privacy, and legal accountability.

This article demonstrates that the richness and contextual relevance of data are more critical than dataset size alone. Through case studies and comparative metrics, we show that platforms like SES, which adopt domain-specific, curated, and legally compliant datasets, outperform quantity-based AI systems across virtually all key dimensions—code quality, maintainability, security, and compliance.

We recommend that organizations adopt rich data strategies when implementing Code Gen AI. These should include curated datasets, embedded regulatory logic, explainable outputs, and continuous feedback loops. While this approach may require more initial investment and configuration, it results in long-term benefits—reduced technical debt, enhanced trust, and significantly lower maintenance costs.

The future of trustworthy AI in software development lies not in the indiscriminate scaling of models but in scaling with intention, ethics, and contextual intelligence. This principle will form the cornerstone of sustainable and secure AI-driven engineering in the years to come.

Best Practice Guidelines for Code Gen AI in Regulated Sectors

The implementation of AI-driven code generation in regulated sectors presents unique challenges that demand

Table 4. Guidelines that offer a practical roadmap for implementing code generation AI responsibly

Guideline	Implementation
Prioritize rich, domain-specific data.	Use curated, annotated, and context-aware datasets that reflect real-world requirements and legal constraints.
Apply weighted decision matrices for data selection.	Assess training and operational data using multidimensional criteria.
Implement human-in-the-loop validation.	Maintain human oversight through subject-matter experts.
Choose architectures that embed explainability and traceability.	Select AI tools and platforms that support end-to-end traceability.
Incorporate continuous feedback loops.	Leverage performance metrics, error rates, and user feedback to iteratively improve outputs.
Ensure compliance by design.	Integrate legal and regulatory logic directly into the AI development lifecycle.
Monitor for technical debt accumulation.	Regularly evaluate AI-generated code for duplication, complexity, and maintainability.
Adopt platforms that align with ethical AI principles.	Use tools designed around fairness, accountability, transparency, and sustainability.

AI: artificial intelligence.

specialized approaches. Healthcare, finance, aerospace, and other highly regulated industries operate under strict compliance frameworks where software failures can have serious consequences, including harm to patients, financial losses, or threats to public safety. Traditional AI code generation techniques that prioritize quantity over quality create unacceptable risks in these environments.

This section presents a comprehensive framework of best practices specifically designed for organizations deploying Code Gen AI in contexts where regulatory compliance, security, and reliability are non-negotiable requirements. These guidelines represent a synthesis of emerging industry standards, lessons learned from early adopters, and principles derived from successful implementations in regulated environments. By following these recommendations, organizations can harness the productivity benefits of AI code generation while maintaining the high standards of quality, security, and compliance that their industries demand.

Table 4 presents guidelines that offer a practical roadmap for implementing Code Gen AI responsibly, with special attention to the unique requirements of healthcare and other regulated sectors where patient safety, data privacy, and regulatory adherence must remain paramount concerns throughout the development process.

The future of sustainable, safe, and responsible software engineering does not lie in scaling indiscriminately, but in scaling ethically and intelligently. Rich data, combined with thoughtful design and governance frameworks, will be the cornerstone of trustworthy AI-driven development—particularly in healthcare, where the stakes are too high for anything less.

Funding

There was no funding received for this research.

Conflicts Of Interest

No.

Contributors

Both authors wrote the article and reviewed by each other.

Data Availability Statement (DAS), Data Sharing, Reproducibility, and Data Repositories

The written code is available in the appendix.

Application of AI-Generated Text or Related Technology

ChatGPT was used to help with grammar checking and initial literature review. All AI-generated content was reviewed and verified by the authors.

Acknowledgments

We would like to thank Russo John, BHTY, for providing an excellent support for proofreading our article.

References

1. GitClear. The impact of AI on code duplication, churn and defects [Internet]. 2024 [cited 2025 Apr 20]. Available from: https://arc.dev/talent-blog/impact-of-ai-on-code/
2. EU AI Act & GDPR Regulations [Internet]. [cited 2025 Apr 20]. Available from: https://artificialintelligenceact.eu/
3. Tantithamthavor C, Cito J. Hemmati H, Chandra S. Explainable AI for SE. IEEE Software; 2023.
4. Self-Evolving Software [Internet]. [cited 2025 Apr 20]. Available from: https://www.selfevolvingsoftware.com
5. Brundage M. Lessons learned on language model safety and misuse. OpenAI; 2022, Available from: https://openai.com/index/language-model-safety-and-misuse/
6. SES Overview. Internal Whitepaper; 2024.
7. Nijkamp E, Zhao J, Poesia G, Xiong C. CodeGen: an open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2303.17568. 2023.
8. Perry N, Srivastava M, Kumar D, Vonneh D. Do users write more insecure code with AI assistants? arXiv preprint arXiv: 2211.03622. 2022. https://doi.org/10.1145/3576915.3623157
9. Bell E. Generative AI vs. Large Language Models (LLMs): what's the difference? [Internet]. 2024 [cited 2025 Apr 20].

Available from: https://appian.com/blog/acp/process-automation/generative-ai-vs-large-language-models
10. IBM. AI code-generation software: what it is and how it works? [Internet]. 2023 [cited 2025 Apr 20]. Available from: https://www.ibm.com/think/topics/ai-code-generation
11. American Psychological Association. Publication manual of the American Psychological Association. 7th ed. Washington, DC; 2019.
12. Bommasanin R, Hudson DA, Adeli E, Altman R, Arora S, von Arx S. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258. 2021.
13. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P. Language models are few-shot learners. Adv Neural Inf Process Syst. 2020;33:1877–901.
14. Chen M, Tworek, J, Jun H, Yuan Q, Pinto P, Kaplan J. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. 2021.
15. Fried D, Aghajanyan A, Lin J, Wang S, Wallace E, Shi F. InCoder: a generative model for code infilling and synthesis. arXiv preprint arXiv:2204.05999. 2022.
16. Li Y, Choi D, Chung J, Kuhman N, Shrittwieser J, LeBlond R, Eccles T. Competition-level code generation with AlphaCode. Science. 2022;378(6624):1092–7. https://doi.org/10.1126/science.abq1158
17. Sherje N. Enhancing software development efficiency through AI-powered code generation. Res J Comput Syst Eng. 2024;5(1):01–12.

Appendix

The output from SES, which accurately anonymized the patient data.

```
public static void anonymizePatient(Id patientId) {
        //Anonymize patient details for GDPR compliance
        Patient__c patient = [SELECT Id, Name, Email__c, Phone__c FROM
Patient__c WHERE Id = :patientId LIMIT 1];

        patient.Name = 'Anonymized';
        patient.Email__c = null;
        patient.Phone__c = null;

        update patient;
    }
```

The other 2 requirements produced the following code outputs:
"Find the priority of a patient"

```
public static Integer calculatePriority(Patient__c patient) {
        //calculate the priority of a patient
        Integer score = 0;

        if (patient.Urgency__c == 'High') {
            score += 50;
        } else if (patient.Urgency__c == 'Medium') {
            score += 30;
        } else {
            score += 10;
        }

        if (patient.Age__c != null) {
            score += Integer.valueof(Math.min(patient.Age__c, 40)); // Age
contributes up to 40 points
        }

        return score;
    }
```


Citation: Blockchain in Healthcare Today 2025, **8**: 396 - https://doi.org/10.30953/bhty.v8.396

"To arrange appointments for patients"

```
public static void scheduleAppointments(List<Patient__c> patients,
List<Doctor__c> doctors) {
        //schedule appointments for patients
        List<Appointment__c> appointmentsToInsert = new List<Appointment__c>();

        Map<integer,Patient__c[]> patientScoreMap = new Map<integer,
Patient__c[]>();
        for (Patient__c p : patients){
            Integer scoreP = PatientPriorityHandler.calculatePriority(p);
            if (!patientScoreMap.containskey(scoreP))
patientScoreMap.put(scoreP, new Patient__c[]{});
            patientScoreMap.get(scoreP).add(p);
        }

        // Sort patients by priority score descending
        List<integer> sortLst = new List<integer>();
        sortLst.addAll(patientScoreMap.keySet());
        sortLst.sort();

        Integer doctorIndex = 0;
        for (Integer scoreVal : sortLst){
            for (Patient__c patient : patientScoreMap.get(scoreVal)) {
                if (doctorIndex >= doctors.size()) {
                    doctorIndex = 0;
                }

                Appointment__c appt = new Appointment__c(
                    Patient__c = patient.Id,
                    Doctor__c = doctors[doctorIndex].Id,
                    Appointment_Date__c = Date.today().addDays(doctorIndex), //
simplistic logic
                    Status__c = 'Scheduled'
                );

                appointmentsToInsert.add(appt);
                doctorIndex++;
            }
        }

        insert appointmentsToInsert;
    }
```