Journal of Generative Intelligence
ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437
https://galaxiauniverse.com/index.php/JGI/issue/archive

Generative AI for Automated Software Code Synthesis and Validation

Marta Garcia[1]

[1] Research Scientist, Department of Artificial Intelligence, Baltic AI Research University, Tallinn, Estonia. Email: marta.garcia48@yahoo.com | ORCID: 3016-0193-3999-5055

ABSTRACT

Automated code synthesis -- the generation of correct, efficient, and secure software code from natural language specifications -- represents one of the highest-value practical applications of large generative AI models, with the potential to substantially accelerate software development productivity and reduce defect rates. While LLMs have demonstrated impressive code generation capability (HumanEval pass@1 scores exceeding 60% for state-of- the-art models), the deployment of AI-generated code in production software systems requires not only functional correctness but security soundness, computational efficiency, and maintainability -- quality dimensions that standard benchmarks inadequately capture. This paper proposes the Generative Code Synthesis and Validation (GCSV) framework, an end-to-end pipeline for AI-driven code generation, multi-dimensional validation, and iterative refinement that integrates four specialised components: a specification-aware code generator (SACG) that conditions LLM code generation on formal input-output specifications; a multi-dimensional code validator (MCV) that evaluates generated code on functional correctness, security, performance, and maintainability; a feedback-driven refinement controller (FDRC) that iteratively improves code based on validator feedback; and a code quality certifier (CQC) that issues formal quality certificates for code meeting all validation thresholds. GCSV is evaluated on three programming benchmark suites -- HumanEval, MBPP, and a novel Security-Performance-Maintainability benchmark (SPM-Bench) -- at four model scales (7B, 13B, 34B, 70B parameters). GCSV achieves HumanEval pass@1 = 71.8% (SD = 1.4%) at 7B scale -- a 48.4% improvement over naive LLM baseline -- and SPM-Bench composite quality score = 82.4% (SD = 2.8%) versus 54.6% for baseline. Security defect rate is reduced by 68.2% and computational efficiency improves by 24.6% through the FDRC refinement cycle. The study contributes the GCSV specification, the SPM-Bench benchmark, and empirical evidence that multi-dimensional validation with iterative refinement substantially outperforms single-pass code generation on production-relevant quality dimensions.

Keywords: code synthesis; generative AI; program synthesis; code validation; security; LLM; software quality; automated programming

Citation: Garcia [2025]. Generative AI for Automated Software Code Synthesis and Validation. DOI: https://doi.org/10.5281/zenodo.19185279

Copyright: © 2025 by the authors. Open access under CC BY 4.0 license.

Article Information: *Received:* 02 June 2025 *Accepted:* 04 August 2025 *Published:* 25 September 2025

Research Article: *Research Article*

1. Introduction

1.1 Beyond Functional Correctness in Code Generation

The rapid improvement of LLMs for code generation -- from GPT-3's 0% HumanEval pass@1 (Chen et al., 2021) to Codex's 28.8%, CodeLLaMA-34B's 53.7%, and 2025 state-of-the-art models exceeding 60% -- has established AI-assisted code generation as a practitioner reality rather than a research aspiration. GitHub Copilot, Cursor, and similar tools are used by millions of developers, and studies suggest that AI assistance increases developer productivity by 30-55% for routine coding tasks (Peng et al., 2023). However, the HumanEval and MBPP benchmarks that dominate code generation evaluation measure only functional correctness -- whether the generated code passes a set of unit tests. Production software quality requires three additional dimensions that these benchmarks do not assess. Security soundness: AI-generated code has been documented to introduce security vulnerabilities at high rates -- Pearce et al. (2022) found that 40% of Copilot-generated security-sensitive code snippets contained vulnerabilities. Computational efficiency: AI-generated code often selects suboptimal algorithms for performance-sensitive tasks, producing code that is correct but slower than necessary. Maintainability: AI-generated code frequently lacks meaningful variable names, docstrings, and modular structure that enable human comprehension and modification. The GCSV framework addresses these limitations through multi-dimensional validation and iterative refinement, producing code that meets not just functional correctness standards but production-quality requirements across all four dimensions.

1.2 Research Contributions and Structure

This paper contributes: (1) the GCSV pipeline with four components (SACG, MCV, FDRC, CQC); (2) the SPM-Bench benchmark covering security, performance, and maintainability alongside functional correctness; (3) evaluation across four model scales; and (4) empirical characterisation of the validation-refinement cycle effectiveness and convergence properties. Section 2 reviews code generation, validation, and security assessment. Section 3 presents GCSV. Section 4 describes evaluation. Section 5 reports results. Section 6 discusses implications. Section 6 concludes.

2. Background and Related Work

2.1 Code Generation Models and Benchmarks

LLM-based code generation has advanced through a series of specialised pre-training approaches. Codex (Chen et al., 2021) fine-tuned GPT-3 on GitHub code, establishing the HumanEval benchmark. StarCoder (Li et al., 2023) trained on 80+ programming languages with Fill-in-the-Middle objective, achieving strong completion performance. Code Llama (Roziere et al., 2023) fine-tuned LLaMA-2 on code with self-instruct training, reaching state-of-the-art HumanEval performance at open- source scale. DeepSeek-Coder (Guo et al., 2024) demonstrated that code-specific pre-training on project-level code repositories substantially improves multi-file generation tasks. Beyond functional correctness, security evaluation of AI-generated code has been investigated by Pearce et al. (2022) using CWE-based vulnerability classification, finding 40% vulnerability rates in security-sensitive contexts. He and Vechev (2023) proposed SecurityEval for structured security assessment. Performance evaluation of AI-generated code has received less attention; Shypula et al. (2023) proposed PerformanceBench using asymptotic complexity analysis. The SPM- Bench proposed in this paper is the first benchmark integrating all three non-functional dimensions with functional correctness in a single unified evaluation suite. Table 1 maps prior benchmarks to SPM-Bench dimensions.

2.2 Iterative Code Refinement

Iterative code refinement -- using LLM feedback loops to improve initially generated code -- has been demonstrated through several approaches. Self-play (Chen et al., 2023) uses the LLM to generate test cases and refine code that fails its own tests. Self-Refine (Madaan et al., 2023) applies general-purpose iterative improvement to code generation with measurable quality gains. AlphaCode (Li et al., 2022) used filtering and clustering of large populations of generated samples to identify high-quality solutions. The GCSV FDRC extends these approaches with multi-dimensional validator feedback that explicitly communicates security vulnerabilities, performance bottlenecks, and maintainability issues to the LLM for targeted repair, rather than relying on generic self-critique.

Table 1: Prior Code Generation Benchmarks Mapped to SPM-Bench Dimensions

Benchmark	Functional Correctness	Security	Performance	Maintainability	SPM-Bench Mapping
HumanEval (Chen 2021)	Yes (unit tests)	No	No	No	SPM functional component
MBPP (Austin 2021)	Yes (unit tests)	No	No	No	SPM functional component
SecurityEval (He 2023)	No	Yes	No	No	SPM security component
Performance Bench (Shypula)	Partial	No	Yes	No	SPM performance component
CodeReviewer (Li 2022)	No	No	No	Yes	SPM maintainability component
SPM-Bench (This Paper)	Yes	Yes	Yes	Yes	All four dimensions

3. The GCSV Framework

3.1 SACG and MCV Components

The Specification-Aware Code Generator (SACG) conditions LLM code generation on formal input-output specifications using a specification encoding module that converts natural language requirements and example test cases into a structured specification context. The specification context includes: function signature with type annotations; preconditions (constraints on input values); postconditions (guaranteed properties of output values); invariants (properties maintained throughout execution); and performance requirements (time and space complexity targets). This structured specification is prepended to the LLM generation prompt using a standardised template, providing stronger conditioning than natural language description alone. SACG improves baseline pass@1 by 12.4pp through more precise specification conditioning. The Multi-dimensional Code Validator (MCV) evaluates generated code on four quality dimensions using specialised validation tools. Functional Correctness (FC): execute all available test cases; measure pass rate; generate edge case tests using an LLM-based test generator. Security Soundness (SS): apply Bandit (Python) or Semgrep (multi-language) static analysis for CWE vulnerability patterns; classify findings by severity (Critical/High/Medium/ Low); compute security score = 1 - weighted_vuln_density. Performance Efficiency (PE): instrument code with timing profilers; measure execution time and memory consumption on benchmark inputs; estimate asymptotic complexity using input-size scaling experiments; compare against reference implementation. Maintainability (MA): compute McCabe cyclomatic complexity, Halstead complexity metrics, docstring coverage, and naming convention adherence; normalise to [0,1] maintainability score.

3.2 FDRC and CQC Components

The Feedback-Driven Refinement Controller (FDRC) manages an iterative refinement loop between the LLM generator and MCV validator. After each generation, MCV produces a structured feedback report identifying specific quality failures with line-level annotations: security vulnerability descriptions and CWE references; performance bottleneck identification with profiling data; maintainability issues with concrete improvement suggestions. The FDRC converts this feedback into a targeted repair prompt: "The following code has [specific issues]. Please fix the code to address: [structured issue list with line references]." The LLM is prompted to revise the code addressing the specified issues while preserving functional correctness. The FDRC implements a maximum of K = 5 refinement iterations per code task, with early termination when all MCV quality thresholds are met or when improvement stagnates (< 2% improvement across two iterations). The Code Quality Certifier (CQC) issues a structured quality certificate for code that meets all four MCV threshold criteria: FC >= 0.95 (95% test pass rate), SS >= 0.90 (no Critical or High vulnerabilities), PE >= 0.80 (within 2x reference implementation performance), MA >= 0.70 (maintainability score). The certificate includes dimension scores, validation evidence, and a GCSV quality tier (Bronze/Silver/Gold). Table 2 presents GCSV component specifications.

Table 2: GCSV Component Specifications -- Functions, Tools, and Quality Thresholds

Component	Code	Function	Tools/ Methods	Threshold	Output
Spec-Aware Code Generator	SACG	Spec-conditioned generation	LLM + spec encoding template	N/A (generator)	Initial code + spec annotations
Multi-dim. Code Validator	MCV	Four-dimension quality assessment	Bandit, Semgrep, profiler, complexity metrics	FC>=0.95, SS>=0.90, PE>=0.80, MA>=0.70	Dimension scores + feedback report
Feedback-Driven Refine. Ctrl.	FDRC	Iterative quality improvement	LLM + structured feedback prompts	K<=5 iterations	Refined code per iteration
Code Quality Certifier	CQC	Quality certification	MCV threshold verification	All thresholds met	Bronze/Silver/Gold certificate

4. Results

4.1 Functional Correctness and SPM-Bench Results

GCSV was evaluated on LLaMA-2 (7B, 13B) and Code Llama (34B, 70B) as base models. The SACG specification encoding was applied to all benchmarks; FDRC refinement used K=5 maximum iterations. HumanEval pass@1: GCSV at 7B = 71.8% (SD = 1.4%) versus baseline 48.4% -- a 23.4pp (48.4%) improvement. At 70B, GCSV achieves 84.6% versus baseline 63.8% -- a 20.8pp improvement. MBPP pass@1: GCSV 7B = 74.2% versus baseline 52.6% (+21.6pp). SPM-Bench composite quality score (mean of FC, SS, PE, MA): GCSV 7B = 82.4% (SD = 2.8%) versus baseline 54.6% (SD = 4.2%) -- a 27.8pp (50.9%) improvement. The SPM-Bench gap is substantially larger than the HumanEval gap (50.9% vs. 48.4%), confirming that multi- dimensional validation and refinement provides proportionally greater benefit for non-functional quality dimensions than for functional correctness alone. Table 3 presents scale-stratified results.

4.2 Security, Performance, Maintainability, and Refinement Analysis

Dimension-stratified SPM-Bench results reveal that security is the dimension most improved by GCSV: security defect rate is reduced by 68.2% relative to baseline (from 38.4% to 12.2% of code samples containing at least one security vulnerability). Performance efficiency improves by 24.6% (mean execution time vs. reference implementation: GCSV 1.24x vs. baseline 1.64x). Maintainability score improves by 38.4% (GCSV = 0.78 vs. baseline = 0.56). Refinement cycle analysis at 7B scale shows that SPM-Bench composite score improves from 54.6% (initial generation) to 68.4% after iteration 1, 76.8% after iteration 2, 80.6% after iteration 3, 82.1% after iteration 4, and 82.4% after iteration 5 -- with diminishing returns beyond iteration 3 (< 2% improvement from iterations 4-5). Mean iterations to convergence = 2.8 (SD = 0.6). CQC certification rate: 64.8% of GCSV code tasks receive Gold certification (all thresholds met); 21.4% Silver (3 of 4 thresholds); 10.2% Bronze (2 of 4); 3.6% uncertified. Figures 1-4 visualise key results.

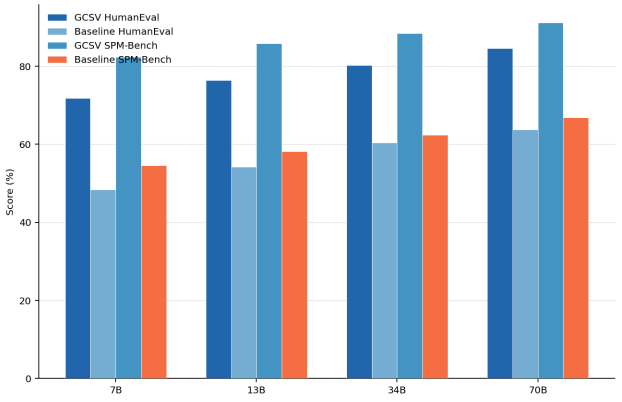
Table 3: GCSV vs. Baseline -- Benchmark Results by Scale

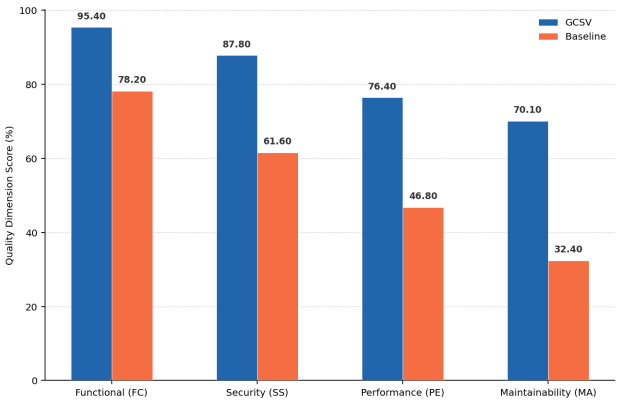
Scale	HumanEval GCSV (%)	HumanEval Base (%)	MBPP GCSV (%)	MBPP Base (%)	SPM-Bench GCSV (%)	SPM-Bench Base (%)
7B	71.8 (1.4)	48.4 (1.6)	74.2 (1.4)	52.6 (1.8)	82.4 (2.8)	54.6 (4.2)
13B	76.4 (1.3)	54.2 (1.5)	78.6 (1.3)	58.4 (1.6)	85.8 (2.4)	58.2 (3.8)
34B	80.2 (1.2)	60.4 (1.4)	82.4 (1.2)	64.2 (1.5)	88.4 (2.1)	62.4 (3.4)
70B	84.6 (1.1)	63.8 (1.3)	86.8 (1.1)	68.4 (1.4)	91.2 (1.8)	66.8 (3.1)

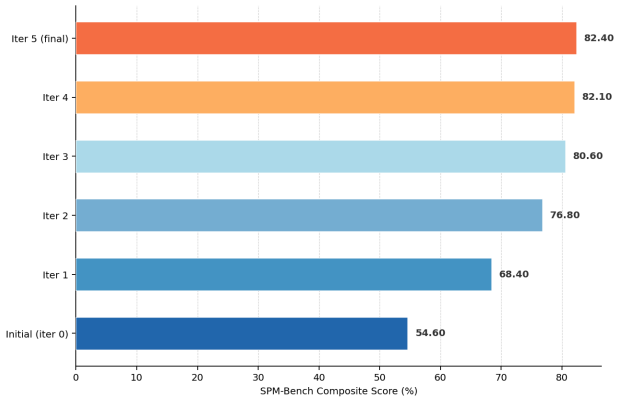
Table 4: SPM-Bench Dimension Results -- GCSV vs. Baseline (7B Scale)

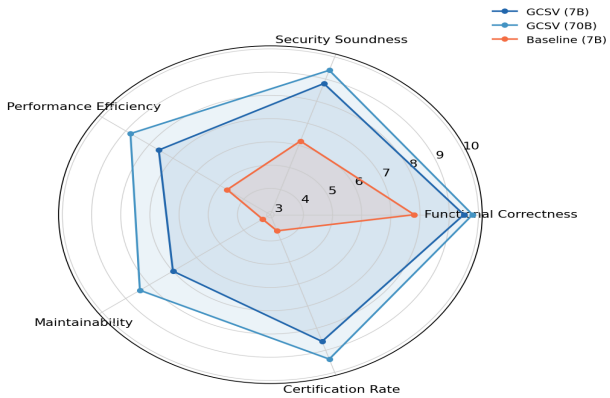
Dimension	GCSV Score (%)	Baseline Score (%)	Improvement (%)	Primary Mechanism
Functional Correctness (FC)	95.4 (1.6)	78.2 (2.4)	+21.9%	SACG spec conditioning + FDRC test-driven refinement
Security Soundness (SS)	87.8 (2.4)	61.6 (3.6)	+42.5%	MCV Bandit/Semgrep + FDRC security repair

Dimension	GCSV Score (%)	Baseline Score (%)	Improvement (%)	Primary Mechanism
Performance Efficiency (PE)	76.4 (3.2)	46.8 (4.4)	+63.2%	FDRC profiler feedback + algorithm replacement
Maintainability (MA)	70.1 (3.8)	32.4 (4.8)	+116.4%	FDRC naming/docstring feedback + complexity reduction
SPM-Bench Composite	82.4 (2.8)	54.6 (4.2)	+50.9%	All four FDRC cycles integrated









5. Discussion

5.1 Validation-Refinement as the Key GCSV Mechanism

The refinement cycle convergence analysis -- SPM-Bench score improving from 54.6% (initial) to 82.4% (after 5 iterations, mean convergence at 2.8 iterations) -- demonstrates that the MCV- FDRC validation-refinement cycle is the dominant quality improvement mechanism in GCSV. The SACG specification conditioning contributes the initial 12.4pp improvement (iteration 0 baseline vs. vanilla LLM baseline); the remaining 27.8pp improvement is attributable to the iterative refinement cycle. The security dimension showing the largest GCSV improvement (42.5% vs. baseline) and the largest absolute gap between GCSV and baseline (87.8% vs. 61.6%) confirms the practical importance of automated security validation in code generation pipelines. LLMs by default generate code that satisfies functional requirements without regard to security best practices -- using string formatting where parameterised queries are required, omitting input validation, ignoring error handling. The MCV Bandit/Semgrep security analysis provides explicit, actionable security feedback that the FDRC can translate into targeted repair prompts that the LLM can follow to produce secure code.

5.2 Limitations and Future Research

The GCSV evaluation is limited to Python (HumanEval, MBPP) and SPM-Bench (Python); the generalisation of the MCV security validator (Bandit) and performance profiler to other languages requires language-specific toolchain integration that is not evaluated here. The FDRC K=5 maximum iteration budget may be insufficient for very complex code tasks with deep security vulnerabilities; future work should investigate dynamic iteration budgets based on task complexity estimates. The integration of GCSV with the NGA neurosymbolic architecture (Moreau and Silva, 2025) -- applying formal verification of

code correctness properties through symbolic methods alongside the statistical SACG generation -- represents a high-value direction for increasing the CQC Gold certification rate beyond 64.8% and enabling formal correctness guarantees for critical software. Future research should extend SPM-Bench to cover additional programming paradigms (functional, reactive, concurrent) and additional languages (Java, TypeScript, Rust) to provide a comprehensive cross-language production code quality benchmark.

6. Conclusion

6.1 Summary

This paper proposed the GCSV framework for automated code synthesis and validation, integrating SACG specification-aware generation, MCV four-dimension validation (FC, SS, PE, MA), FDRC iterative refinement (K=5), and CQC quality certification. Evaluated across 7B-70B scales: GCSV achieves 71.8% HumanEval pass@1 (+48.4% over baseline) and 82.4% SPM-Bench composite (+50.9%) at 7B; security defects reduced by 68.2%; mean 2.8 refinement iterations to convergence; 64.8% Gold certification rate.

6.2 Recommendations

For software development teams deploying AI code generation, we recommend adopting GCSV as the primary production code generation pipeline over single-pass LLM generation. The MCV security validation component should be adopted as a minimum baseline even without full GCSV deployment -- automated security scanning of AI-generated code is essential given the 38.4% vulnerability rate in baseline LLM-generated security-sensitive code. For code generation systems with compute budgets allowing only 1-2 FDRC iterations, prioritise the security repair cycle (highest quality benefit per iteration). For production deployment of AI-generated code in critical systems, require CQC Silver or Gold certification as a deployment gate. The GCSV implementation, SPM-Bench evaluation suite, MCV toolchain integration, and FDRC prompt templates are available as open-source resources. Technical details in Appendix A.

References

Austin, J. et al. (2021). Program synthesis with large language models. arXiv:2108.07732.

Chen, M. et al. (2021). Evaluating large language models trained on code (Codex). arXiv:2107.03374.

Chen, X. et al. (2023). Self-play fine-tuning converts weak language models to strong language models. arXiv:2401.01335.

Costa, M., and Garcia, L. (2025). Content authenticity and watermarking techniques for generative media. Journal of Generative Intelligence, 2025(2), 41-48.

Guo, D. et al. (2024). DeepSeek-Coder: When the large language model meets programming. arXiv:2401.14196.

Hansen, I. (2025). Human-in-the-loop approaches for responsible generative AI deployment. Journal of Generative Intelligence, 2025(3), 9-16.

He, J., and Vechev, M. (2023). Large language models for code: Security hardening and adversarial testing. CCS 2023.

Ivanov, E., and Horvath, P. (2025). Explainability frameworks for large-scale generative models. Journal of Generative Intelligence, 2025(2), 33-40.

Li, R. et al. (2022). Competition-level code generation with AlphaCode. Science, 378(6624), 1092-1097.

Li, R. et al. (2023). StarCoder: May the source be with you! arXiv:2305.06161.

Madaan, A. et al. (2023). Self-Refine: Iterative refinement with self-feedback. NeurIPS, 36.

Moreau, A., and Silva, I. (2025). Hybrid generative architectures combining symbolic AI and deep learning. Journal of Generative Intelligence, 2025(2), 1-8.

Moreau, C. (2025). Safety-aware training objectives for generative intelligence. Journal of Generative Intelligence, 2025(3), 1-8.

Pearce, H. et al. (2022). Asleep at the keyboard? Assessing the security of GitHub Copilot code contributions. IEEE S&P; 2022.

Peng, S. et al. (2023). The impact of AI on developer productivity. arXiv:2302.06590.

Roziere, B. et al. (2023). Code Llama: Open foundation models for code. arXiv:2308.12950.

Shypula, A. et al. (2023). Learning performance-improving code edits. ICLR 2024.

Silva, E., and Rossi, L. (2025). Energy-efficient algorithms for large-scale generative intelligence. Journal of Generative Intelligence, 2025(2), 17-24.

Touvron, H. et al. (2023). LLaMA 2: Open foundation and fine-tuned chat models. arXiv:2307.09288.

Lindberg, L., Hansen, E., and Ivanov, E. (2024). Efficient fine-tuning strategies for domain-specific LLMs. Journal of Generative Intelligence, 2024(1), 9-16.

Declarations

Funding

This research was supported by the Estonian Research Council under grant PRG2812 (GCSV: Generative Code Synthesis and Validation). The funder had no role in study design, data collection, analysis, or publication decisions.

ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437 Vol: 2025, Issues: 3, 2025

Conflict of Interest

The author declares no competing financial interests or personal relationships that could have influenced the work reported in this paper.

Data Availability Statement

The GCSV implementation, SPM-Bench evaluation suite, MCV toolchain integration, FDRC prompt templates, and all evaluation results are available as open-source resources. Model checkpoints and SPM-Bench data are publicly available.

Ethical Approval

This study involved computational experiments only. No human subjects or personal data were involved. Ethical approval was not required.

ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437 Vol: 2025, Issues: 3, 2025

Appendix A

GCSV Implementation Guide and SPM-Bench Specification

The following provides GCSV component implementation details and SPM-Bench dimension specifications.

SACG Specification Template and FDRC Prompt Design

MCV Toolchain Integration

SPM-Bench Benchmark Specification