INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 7,854 Volume 13, issue 08 (2026)

EvoGraph AI : An Evolutionary Graph-Based Framework for Adaptive Software Engineering and Intelligent Code Optimization

Nur Aisyah Rahman
Department of Intelligent Systems and Machine
National Institute of Digital Technology, Penang Malaysia

Abstract: The increasing complexity of modern software systems has created a need for intelligent engineering approaches capable of representing software dependencies, identifying structural relationships, and adapting optimization decisions to changing development conditions. Conventional code optimization methods frequently operate on isolated source files, functions, or predefined metrics, limiting their ability to reason about the broader structural and behavioral context of software systems. This research proposes Evo GraphAI, an evolutionary graph-based framework for adaptive software engineering and intelligent code optimization. The framework conceptualizes software as a dynamic graph in which source-code components, dependencies, execution relationships, interfaces, and optimization objectives are represented as interconnected entities. Evolutionary search mechanisms are then applied to graph-derived representations to identify improved code configurations while preserving functional and structural constraints. The methodological foundation integrates graph-based software representation, learning-oriented pattern extraction, adaptive candidate generation, fitness-based evaluation, and iterative optimization. The provided literature, although primarily focused on wearable sensing and human activity recognition, contributes transferable concepts concerning multimodal representation, personalization, sensor reliability, and adaptive classification. These principles are interpreted as methodological analogies for constructing context-aware software representations rather than as direct evidence of software optimization performance. The framework is further positioned in relation to EvoGraphCoder, which establishes an evolutionary graph-reasoning direction for self-adaptive software engineering (Ramamurthy et al., 2026). The proposed analysis indicates that graph-centered evolutionary reasoning can provide a coherent foundation for dependency-aware optimization, personalization of software transformations, and adaptive decision-making. However, computational overhead, graph construction complexity, fitness-function design, and preservation of semantic correctness remain important limitations requiring empirical validation.

Keywords: evolutionary computing, graph-based software engineering, intelligent code optimization, adaptive software systems, graph reasoning, artificial intelligence, software optimization, program analysis, evolutionary search

INTRODUCTION

1.1 Background

Software engineering has progressively shifted from manually controlled development processes toward intelligent and increasingly automated environments. Contemporary software systems contain large numbers of interacting modules, libraries, interfaces, services, configuration elements, and execution dependencies. Consequently, optimizing one component without considering its relationships with other components may produce locally favorable but globally undesirable outcomes. An optimization that reduces execution time in

one module, for example, may increase memory consumption elsewhere or introduce undesirable dependency interactions.

A graph-based representation offers a natural mechanism for addressing this structural complexity. Software can be modeled as a graph in which nodes represent functions, classes, files, services, variables, or other software entities, while edges represent relationships such as calls, inheritance, data dependencies, imports, or communication. Such representation changes optimization from a purely textual transformation problem into a relational reasoning problem.

The importance of context-aware representation is supported indirectly by the provided literature. Research on human activity recognition demonstrates that information derived from multiple sensing modalities can improve the representation and interpretation of complex activities (Ihianle et al., 2020; Lara and Labrador, 2013). Similarly, software optimization can benefit from integrating multiple forms of evidence, including source structure, dependency relationships, historical optimization outcomes, runtime characteristics, and developer-defined constraints.

Personalization is another important consideration. Ferrari et al. (2020) examined personalization of classification models for human activity recognition, demonstrating the broader importance of adapting computational models to the characteristics of their operating context. In software engineering, this principle can be translated into optimization policies that adapt to project-specific architectures, coding practices, performance requirements, and operational constraints.

The proposed EvoGraphAI framework therefore combines graph representation with evolutionary optimization. Its central premise is that software optimization should not be treated solely as a sequence of predefined transformations but as an adaptive search process over a structured software state space.

1.2 Problem Statement

Existing optimization processes may suffer from three related limitations. First, isolated code-level analysis can overlook dependencies between software components. Second, static optimization rules may not adequately respond to different project contexts. Third, optimization objectives can conflict, requiring systematic exploration of alternative solutions rather than a single deterministic transformation.

These challenges motivate an architecture in which graph-based structural information guides evolutionary search. The framework must simultaneously preserve semantic correctness, evaluate competing objectives, and adapt its search strategy according to the characteristics of the software graph.

1.3 Research Relevance

The relevance of this approach lies in its potential to unify structural software representation and adaptive optimization. The evolutionary graph-reasoning direction represented by EvoGraphCoder provides a closely related conceptual foundation for self-adaptive software engineering (Ramamurthy et al., 2026). EvoGraphAI extends this research direction conceptually toward intelligent code optimization by emphasizing graph-aware candidate generation, fitness evaluation, and adaptive transformation.

The provided studies also reveal transferable methodological principles. Wearable-sensor research emphasizes reliability, multimodal integration, personalization, and context-dependent analysis (Guignard et al., 2021; Seshadri et al., 2019). These principles are relevant to intelligent software engineering because optimization decisions likewise depend on the quality and combination of heterogeneous information sources.

1.4 Objectives

The primary objective is to develop a conceptual framework for evolutionary graph-based software optimization. Specifically, the research aims to:

1. formulate a graph representation of software structure and dependencies;
2. develop an evolutionary mechanism for generating optimization candidates;
3. integrate multiple software-level objectives into a fitness model;
4. establish an adaptive feedback mechanism for iterative optimization;
5. identify theoretical advantages, limitations, and future empirical requirements.

1.5 Scope and Significance

The study focuses on conceptual architecture and methodological formulation rather than claiming experimentally validated performance. Its significance is the integration of graph reasoning and evolutionary optimization into a unified adaptive software-engineering framework. Such integration may support optimization scenarios involving performance, maintainability, resource utilization, dependency reduction, and architectural restructuring.

LITERATURE REVIEW

The literature supplied for this study originates primarily from wearable sensing, sports analysis, and human activity recognition. Although these studies do not directly investigate software optimization, their methodological characteristics provide useful transferable concepts for constructing adaptive computational frameworks.

Lara and Labrador (2013) presented a broad survey of human activity recognition using wearable sensors, highlighting the importance of extracting meaningful activity information from heterogeneous sensor observations. This perspective is relevant to EvoGraphAI because software systems similarly contain heterogeneous signals. Source-code structure, dependency information, runtime behavior, historical changes, and quality metrics can be considered different observational channels describing software state.

Ihianle et al. (2020) investigated deep learning for human activity recognition from multimodal sensing devices. Their emphasis on multimodal information suggests that intelligent systems can benefit from combining complementary representations rather than relying on a single information source. For EvoGraphAI, this supports a multimodal software-state model in which graph topology is combined with semantic, behavioral, and performance information.

The importance of reliable measurement is demonstrated by Guignard et al. (2021), who investigated the validity, reliability, and accuracy of inertial measurement units for angular measurement in swimming. The methodological implication for software optimization is significant: optimization cannot be reliable when the underlying measurements are unstable or poorly characterized. Performance metrics, memory measurements, execution traces, and code-quality indicators must therefore be treated as potentially noisy observations rather than unquestionable ground truth.

Seshadri et al. (2019) examined wearable sensors for monitoring physiological and biochemical characteristics

of athletes. Their work reinforces the value of continuous monitoring in computational decision-making. In an adaptive software environment, continuous monitoring can similarly provide feedback concerning performance and resource behavior.

Muniz-Pardos et al. (2018) examined integration of wearable sensors into the evaluation of running economy and foot mechanics. The integration perspective is particularly relevant to EvoGraphAI because optimization requires combining structural and behavioral evidence. A graph alone describes relationships, but optimization quality improves when graph information is linked with observable system behavior.

Malawski (2021) compared depth and inertial sensors in real-time sports analysis, illustrating that different sensing mechanisms provide different forms of information and may involve trade-offs. This principle maps naturally to software optimization, where static analysis and dynamic profiling provide complementary advantages. Static analysis offers broad structural coverage, while dynamic analysis provides execution-specific evidence.

Struber et al. (2021) examined the reliability of human running analysis using low-cost inertial and magnetic sensor arrays. Their focus on reliability reinforces the importance of robust measurement pipelines. In EvoGraphAI, unreliable measurements can distort the evolutionary fitness function and consequently direct the search toward inappropriate code transformations.

Wang et al. (2019) investigated wearable sensors for capturing lumbar-spine posture during competitive swimming. The study demonstrates how sensor-derived representations can encode complex physical relationships. The analogous software-engineering principle is that graph representations can encode relationships among software components that are difficult to capture through isolated textual inspection.

Tabrizi et al. (2021) applied deep learning to table-tennis forehand-stroke evaluation using an IMU sensor. This illustrates the use of learned representations for evaluating complex actions. EvoGraphAI similarly proposes learned or computationally derived representations to evaluate software states and candidate transformations.

Ferrari et al. (2020) specifically addressed personalization of classification models. This is particularly relevant to adaptive software engineering because optimization policies should not necessarily be identical across applications. A transformation beneficial for a computationally intensive service may be inappropriate for a memory-constrained embedded application.

Worsey et al. (2019) systematically reviewed performance analysis in rowing using inertial sensors. Their work highlights the value of systematic performance analysis, which is central to evolutionary software optimization. Candidate solutions must be evaluated according to measurable criteria rather than subjective assumptions.

Finally, Eline and Marco (2018) reviewed human motion capture systems for sports applications and emphasized the broader challenge of accurately representing physical motion. The conceptual parallel is the representation challenge in software engineering: the usefulness of an optimization algorithm depends strongly on whether its underlying representation captures the relationships relevant to the optimization objective.

Taken together, the literature establishes several transferable methodological foundations: multimodal representation, measurement reliability, personalization, continuous monitoring, comparative evaluation, and structured representation. These concepts complement the graph-evolutionary software-engineering direction

identified by Ramamurthy et al. (2026).

A major research gap remains evident. The supplied literature does not directly establish an experimentally validated graph-based evolutionary code-optimization framework. Therefore, EvoGraphAI should be understood as a proposed conceptual synthesis rather than as a directly empirically confirmed extension of the wearable-sensing studies. The framework's primary theoretical contribution is connecting adaptive representation and evolutionary search to software graphs.

METHODOLOGY

3.1 Research Design

This study adopts a framework-development methodology. The objective is to construct a technically coherent architecture by synthesizing graph representation, adaptive computational analysis, evolutionary search, and feedback-driven optimization.

The methodology consists of five interconnected layers:

Software Graph Construction → Multimodal Feature Integration → Evolutionary Candidate Generation → Fitness Evaluation → Adaptive Feedback

The architecture treats software as a dynamic state rather than a static artifact. At iteration (t), the software state can be represented as:

[

G_t=(V_t,E_t,X_t)

]

where (V_t) represents software entities, (E_t) represents relationships among entities, and (X_t) represents associated attributes such as complexity, execution behavior, resource usage, semantic information, or historical characteristics.

3.2 Software Graph Construction

The first stage transforms software artifacts into a graph. Nodes may represent functions, classes, files, modules, services, or APIs. Edges represent calls, data dependencies, inheritance, imports, composition, communication, or other relevant relationships.

For example, if function (A) calls functions (B) and (C), the graph contains edges (A\rightarrow B) and (A\rightarrow C). If (B) shares data with (D), another relationship can be added. The resulting graph exposes structural dependencies that isolated code optimization may overlook.

Graph construction should also attach attributes to nodes and edges. A function node may contain cyclomatic complexity, execution frequency, code size, or historical modification frequency. An edge may contain dependency type, call frequency, or data-transfer characteristics.

3.3 Multimodal Software-State Representation

The second layer integrates multiple information sources. The principle follows the methodological value of

multimodal sensing demonstrated by Ihianle et al. (2020). Instead of relying solely on source code, EvoGraphAI combines:

[

X = [X_{static},X_{dynamic},X_{semantic},X_{historical}]

]

where static features describe source structure, dynamic features describe runtime behavior, semantic features characterize code meaning, and historical features represent previous changes and optimization outcomes.

This representation improves contextual reasoning but also introduces measurement complexity. As demonstrated by research emphasizing sensor reliability and accuracy (Guignard et al., 2021; Struber et al., 2021), computational decisions are sensitive to the quality of the observations supplied to the decision process.

3.4 Evolutionary Candidate Generation

The evolutionary engine generates candidate modifications to the software graph. Candidate transformations may include function restructuring, redundant-code removal, dependency simplification, algorithm substitution, or graph-level refactoring.

A candidate solution can be represented as:

[

C_i = (G_i,T_i)

]

where (G_i) is the resulting graph and (T_i) is the sequence of transformations applied to the original graph.

The evolutionary population consists of multiple candidates:

[

P_t={C_1,C_2,\ldots,C_n}

]

Selection retains candidates with superior fitness, while mutation and recombination generate new alternatives. The iterative process enables exploration of a larger optimization space than a single deterministic transformation rule.

3.5 Fitness Evaluation

Fitness must capture multiple objectives because software quality is multidimensional. A conceptual fitness function can be defined as:

[

F(C)=w_pP(C)+w_mM(C)+w_rR(C)+w_dD(C)-w_cC_x(C)

]

where (P) represents performance improvement, (M) maintainability, (R) resource efficiency, (D) dependency quality, and (C_x) transformation complexity. The weights (w) determine project-specific priorities.

This formulation also supports personalization. Ferrari et al. (2020) demonstrate the relevance of adapting computational models to individual contexts. Analogously, EvoGraphAI can adjust fitness weights according to application requirements. A real-time application may prioritize execution latency, whereas a long-lived enterprise system may prioritize maintainability and dependency stability.

3.6 Semantic and Structural Constraints

Optimization cannot be considered successful merely because a candidate improves a numerical metric. Functional equivalence and architectural validity must be preserved.

Therefore, each candidate passes through constraint validation:

[

Valid(C_i)=

\begin{cases}

1,& \text{if semantic and structural constraints are satisfied}\\

0,& \text{otherwise}

\end{cases}

]

Invalid candidates are rejected regardless of their apparent performance advantage.

This constraint mechanism is particularly important because evolutionary search can discover unusual transformations that exploit weaknesses in the evaluation process. Consequently, testing, static verification, and dependency validation must operate alongside the optimization engine.

3.7 Adaptive Feedback Mechanism

The final methodological component is feedback. After each evolutionary cycle, performance measurements are collected and compared against previous states. The graph and optimization policy can then be updated.

Continuous monitoring is conceptually consistent with the monitoring orientation found in wearable-sensor research (Seshadri et al., 2019). For software, feedback may include execution latency, memory utilization, CPU consumption, failure rates, code complexity, and regression-test outcomes.

The framework therefore operates as:

[

G_t \rightarrow P_t \rightarrow F_t \rightarrow G_{t+1}

]

This cycle enables the software representation and optimization strategy to evolve together.

3.8 Proposed Algorithmic Workflow

The proposed EvoGraphAI workflow begins by ingesting source repositories and associated software metadata. A graph parser constructs the initial software graph. Feature extraction then attaches static, dynamic, semantic, and historical attributes. The evolutionary engine generates candidate transformations, evaluates them through a multi-objective fitness function, removes candidates violating semantic or structural constraints, and retains high-quality solutions. The resulting optimized software is subsequently monitored, and new observations update the graph and optimization policy.

The conceptual framework is aligned with the evolutionary graph-reasoning direction of EvoGraphCoder, which provides a relevant foundation for self-adaptive software engineering (Ramamurthy et al., 2026). EvoGraphAI emphasizes the additional optimization dimension by treating code transformations as candidate graph states within an evolutionary search process.

RESULTS

The framework analysis produces several principal findings concerning the feasibility and theoretical value of graph-based evolutionary software optimization.

First, graph representation provides a stronger structural basis for optimization than isolated source-level analysis. Software dependencies, call relationships, and architectural connections become explicit optimization variables. This enables the system to identify transformations whose consequences extend beyond individual functions or files. The approach is particularly valuable for large systems where local changes can propagate through multiple dependency paths.

Second, multimodal software-state representation appears theoretically advantageous. The provided literature repeatedly demonstrates that complex recognition and analysis tasks benefit from integrating multiple sources of information (Ihianle et al., 2020; Lara and Labrador, 2013). Applied to software, this suggests that source structure, runtime behavior, semantic characteristics, and historical information should be jointly considered when determining optimization priorities.

Third, personalization emerges as a significant mechanism for adaptive optimization. Ferrari et al. (2020) demonstrate the value of personalized computational models, while the proposed framework applies the same general principle to software environments. EvoGraphAI can theoretically assign different fitness priorities according to application requirements, making optimization context-sensitive rather than universally fixed.

Fourth, measurement reliability is a critical determinant of optimization quality. The attention to accuracy and reliability in sensor-based research (Guignard et al., 2021; Struber et al., 2021) provides a useful methodological analogy. If runtime profiling or quality measurements are unstable, evolutionary selection may favor candidates based on noise. Consequently, repeated measurements, robust aggregation, and validation are necessary components of an operational implementation.

Fifth, the evolutionary mechanism provides a natural method for handling conflicting objectives. Instead of forcing one deterministic optimum, the system can maintain multiple candidate solutions representing different trade-offs among execution speed, memory utilization, maintainability, and dependency complexity.

This is especially useful when no single solution simultaneously maximizes all software-quality dimensions.

Finally, the framework identifies important implementation challenges. Graph construction can become computationally expensive for very large repositories. Evolutionary search may require substantial evaluation cycles because every candidate must be tested. Fitness-function design can also introduce unintended optimization behavior. These findings indicate that EvoGraphAI is conceptually promising but requires controlled empirical experimentation before performance claims can be generalized.

DISCUSSION

The findings suggest that graph-based evolutionary reasoning can provide an appropriate theoretical foundation for adaptive software engineering because software is inherently relational. Traditional code optimization frequently emphasizes individual functions or localized transformations, whereas graph representation captures the dependencies through which local changes influence global system behavior. This distinction is important for intelligent optimization because a transformation that appears beneficial at source level may have negative architectural consequences.

The strongest theoretical contribution of EvoGraphAI is therefore the integration of structural representation and evolutionary exploration. Ramamurthy et al. (2026) establish an evolutionary graph-reasoning direction for self-adaptive software engineering, and the proposed framework builds upon this conceptual trajectory by emphasizing code optimization as a graph transformation problem. The evolutionary component is particularly useful when the optimization space contains multiple competing possibilities and no deterministic rule can reliably identify the best transformation.

The literature on multimodal sensing further supports the proposed architecture. Ihianle et al. (2020) and Lara and Labrador (2013) demonstrate the importance of integrating heterogeneous information for complex recognition tasks. In software engineering, graph topology alone may not reveal whether a component is a performance bottleneck. Runtime evidence, execution frequency, semantic information, and historical changes can provide complementary evidence. Consequently, a multimodal graph is theoretically more informative than a purely structural graph.

Personalization also creates an important distinction between generic optimization and adaptive optimization. Ferrari et al. (2020) illustrate how model personalization can improve contextual relevance. EvoGraphAI translates this concept into project-specific optimization policies. However, personalization creates a trade-off: a highly customized optimizer may perform effectively within one project but generalize poorly to unfamiliar software architectures. Therefore, adaptive models should retain general optimization principles while allowing project-specific calibration.

Measurement reliability presents another important limitation. Guignard et al. (2021) and Struber et al. (2021) show that measurement validity and reliability are fundamental issues in sensing systems. The same logic applies to software optimization. Performance metrics can vary with workload, hardware, compilation state, concurrency, and environmental conditions. An evolutionary system that treats a single noisy measurement as ground truth may select inferior candidates. Robust evaluation protocols are therefore essential.

The principal practical trade-off concerns optimization quality versus computational cost. More sophisticated graph representations and larger evolutionary populations may improve search capability but increase computational requirements. Similarly, evaluating every candidate through comprehensive testing may substantially increase optimization time. A practical implementation should therefore use staged evaluation, where inexpensive structural checks eliminate unsuitable candidates before costly dynamic validation.

The framework also faces a semantic-preservation challenge. Evolutionary search is inherently exploratory, and optimization pressure can encourage transformations that improve measured objectives without preserving intended behavior. Strong testing and semantic constraints are therefore not optional components but fundamental safeguards.

Overall, EvoGraphAI should be interpreted as a structured research framework rather than an experimentally proven optimization engine. Its theoretical value lies in connecting graph-based software representation, adaptive multimodal analysis, evolutionary search, and feedback-driven optimization. Empirical validation across heterogeneous software projects remains necessary to determine whether these conceptual advantages translate into measurable improvements.

CONCLUSION

This research proposed EvoGraphAI, an evolutionary graph-based framework for adaptive software engineering and intelligent code optimization. The framework models software as a dynamic graph, integrates heterogeneous software-state information, generates candidate transformations through evolutionary search, evaluates candidates using multi-objective fitness functions, and applies continuous feedback to support adaptation.

The literature synthesis indicates that multimodal representation, personalization, measurement reliability, continuous monitoring, and structured analysis provide useful methodological foundations for adaptive computational systems. Although the supplied references primarily concern wearable sensing and sports analysis, their principles can be transferred conceptually to software optimization. The evolutionary graph-reasoning direction represented by Ramamurthy et al. (2026) provides the most direct conceptual foundation for positioning graph-based evolutionary intelligence within self-adaptive software engineering.

The principal contribution of EvoGraphAI is its treatment of code optimization as an evolutionary graph-transformation problem rather than solely as localized source-code rewriting. This perspective can potentially support dependency-aware optimization and project-specific adaptation while simultaneously handling competing software-quality objectives.

Nevertheless, the framework remains subject to significant limitations, particularly graph-construction overhead, evolutionary search cost, fitness-function sensitivity, measurement instability, and semantic-preservation requirements. Future empirical research should implement the framework and evaluate it across diverse software repositories using controlled benchmarks, multi-objective performance measures, regression testing, and ablation studies. Such investigations would establish whether graph-aware evolutionary optimization can produce consistently superior and reliable software transformations.

REFERENCES

1. V. D. K. Eline and M. R. Marco, “Accuracy of human motion capture systems for sport applications; state-of-the-art review,” European Journal of Sport Science, vol. 18, no. 6, pp. 806–819, 2018.

2. A. Ferrari, D. Micucci, M. Mobilio and P. Napoletano, “On the personalization of classification models for human activity recognition,” IEEE Access, vol. 8, pp. 32066–32079, 2020.

3. B. Guignard, O. Ayad, H. Baillet, F. Mell, E. D. Simbaña, J. Boulanger et al., “Validity, reliability and accuracy of inertial measurement units (IMUs) to measure angles: Application in swimming,” Sports Biomech, Advance Online Publication, pp. 1–33, 2021.

4. I. K. Ihianle, A. O. Nwajana, S. H. Ebenuwa, R. I. Otuka, K. Owa et al., "A deep learning approach for human activities recognition from multimodal sensing devices," IEEE Access, vol. 8, pp. 179028–179038, 2020.

5. O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," IEEE Communications Surveys & Tutorials, vol. 15, no. 3, pp. 1192–1209, 2013.

6. F. Malawski, "Depth versus inertial sensors in real-time sports analysis: A case study on fencing," IEEE Sensors Journal, vol. 21, no. 4, pp. 5133–5142, 2021.

7. B. Muniz-Pardos, S. Sutehall, J. Gellaerts, M. Falbriard, B. Mariani et al., "Integration of wearable sensors into the evaluation of running economy and foot mechanics in elite runners," Current Sports Medicine Reports, vol. 17, no. 12, pp. 480–488, 2018.

8. S. S. Tabrizi, S. Pashazadeh and V. Javani, "A deep learning approach for table tennis forehand stroke evaluation system using an IMU sensor," Computational Intelligence and Neuroscience, vol. 9, pp. 1–15, 2021.

9. L. Struber, S. Ledouit, O. Daniel, P.-A. Barraud and V. Nougier, "Reliability of human running analysis with low-cost inertial and magnetic sensor arrays," IEEE Sensors Journal, vol. 21, no. 13, pp. 15299–15307, 2021.

10. D. R. Seshadri, R. T. Li, J. E. Voos, J. R. Rowbottom, C. M. Alfes et al., "Wearable sensors for monitoring the physiological and biochemical profile of the athlete," Npj Digital Medicine, vol. 2, no. 72, pp. PMC6646404, 2019.

11. M. T. Worsey, H. G. Espinosa, J. B. Shepherd and D. V. Thiel, "A systematic review of performance analysis in rowing using inertial sensors," Electronics, vol. 8, no. 11, pp. 1304, 2019.

12. Z. Wang, J. Wang, H. Zhao, S. Qiu, J. Li et al., "Using wearable sensors to capture posture of the human lumbar spine in competitive swimming," IEEE Transactions on Human-Machine Systems, vol. 49, no. 2, pp. 194–205, 2019.

13. K. Ramamurthy, R. K. Konduru and N. Amanmadov, "EvoGraphCoder: An Evolutionary Graph-Reasoning Framework for Self-Adaptive Software Engineering," in IEEE Access, vol. 14, pp. 63063-63076, 2026, doi: 10.1109/ACCESS.2026.3686019.