 ISSN (Online): 3140-5312

Semantic Cohesion Analysis of the Chunk of Code with an LLM

Eunjin Jeong[1], Janghwan Kim[2], Chaeyun Seo[3], R. Young Chul Kim[4]

[1]Undergraduate Student, Dept. of Software and Communications Engineering, Hongik University, South Korea

[2]Ph.D Candidate, Dept. of Software and Communications Engineering, Hongik University, South Korea

[3]Ph.D, Dept. of Software and Communications Engineering, Hongik University, South Korea

[4]Professor, Dept. of Software and Communications Engineering, Hongik University, South Korea

Corresponding author: R. Young Chul Kim (bob@hongik.ac.kr)

Acknowledgement: This research was supported by the Ministry of Science and ICT (MSIT), Korea, under the Sejong SW Convergence Cluster 2.0 Project supervised by the National IT Industry Promotion Agency (NIPA) (Grant No. PJTS080126041007000100030200).

Abstract

Recent evaluations of software quality have raised concerns that structural metrics fail to capture design intent adequately. Metrics such as LCOM measure only the physical connectivity between code elements, missing the semantic notion of cohesion that reflects a developer's actual intent. To address this, we propose a Large Language Model (LLM)-based semantic cohesion evaluation model built on the 'LLM-as-a-Judge' paradigm, reformulating classical cohesion theory — grounded in Stevens' taxonomy and the Single Responsibility Principle — into five semantic rubric criteria evaluated through an automated process. This builds on prior findings that top-tier LLMs, such as Claude Sonnet and GPT-4o-mini, correlate strongly with human expert reviewers when interpreting identifiers, comments, and logic flow, letting them catch design-intent distortions that static analysis misses. We detail a five-stage pipeline — from preprocessing and AST-based context extraction to rationale-backed scoring — that converts subjective judgments into an objective, data-driven metric, laying groundwork for a semantics-centered code quality framework. Future work will validate the rubric across multiple LLMs and extend it into an automated refactoring recommendation tool.

Keywords: Large Language Model, Semantic Cohesion, LLM-as-a-Judge, Source Code Analysis

1. Introduction

As software systems grow more complex, managing cohesion has become increasingly important for preserving module independence and maintainability [1]. Cohesion measures the degree to which a module's components cooperate toward a single purpose, and high-cohesion design is central to improving readability and minimizing side effects [2]. However, existing structural metrics such as Lack of Cohesion in Methods (LCOM) rely solely on syntactic connectivity and therefore fail to capture the semantic notion of cohesion, leaving design intent and code context unreflected [3, 4].

To address this limitation, we propose a semantic cohesion measurement method based on the 'LLM-as-a-Judge' paradigm. Our goal is to compute a deeper level of cohesion — one that is difficult to capture through static analysis — by leveraging semantic information such as identifiers and comments. To this end, we re-establish traditional cohesion theory and build a five-criterion semantic rubric evaluation system optimized for LLMs' qualitative reasoning capabilities [5]. This enables quantification of subjective code interpretation against a

 ISSN (Online): 3140-5312

consistent standard, enabling semantic quality evaluation while providing an in-depth rationale for each judgment.

Section 2 reviews related work, Section 3 details the proposed methodology, and Section 4 concludes the paper with directions for future work.

2. Related Work

2.1 Cohesion Models in the Object-Oriented Paradigm

Traditional cohesion is a measure of the association among components within a module; in the object-oriented paradigm, it has been concretized primarily in terms of relationships between classes and methods. Prior work has proposed schemes that identify cohesion levels based on the syntactic structure of code, such as method call frequency, parameter sharing, and return-value usage [6], with sequential and communicational cohesion — where data flow is clear — widely used as key indicators of module reusability. As noted above, however, such purely structural measures cannot infer the logical consistency or design intent latent in the code, motivating the LLM-based semantic analysis model proposed in this paper.

2.2 LLM-as-a-Judge Based Rubric Evaluation Model

Recently in software engineering, the 'LLM-as-a-Judge' paradigm — which uses LLMs as the evaluating agent for software artifact quality — has drawn significant attention, building on the foundational LLM-as-a-Judge framework originally proposed for evaluating open-ended natural-language responses [7]. A subset of high-performing LLMs — such as Claude Sonnet and GPT-4o-mini — has been empirically shown to achieve high correlation with human expert reviewers when interpreting naming conventions, comment context, and logic flow [8], suggesting that this interpretive capability can be leveraged to analyze semantic cohesion that reflects design intent.

However, simple prompting approaches suffer from hallucination and inconsistent, subjective interpretation. To address this, our study adopts a rubric-based evaluation process (Fig. 1), since prior work shows that a concrete rubric improves both the validity and reproducibility of LLM-based evaluation [5].

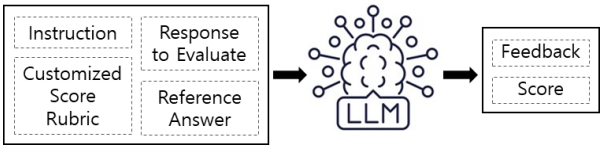

Fig. 1. LLM-as-a-Judge evaluation process

This process jointly analyzes the Instruction, the Response to Evaluate, the Reference Answer, and a Customized Score Rubric, generating both a score and accompanying feedback that grounds the evaluation in a clear, traceable rationale.

3. Proposed Methodology

3.1 Design of an LLM-Based Semantic Cohesion Evaluation Rubric

We accept the criticism raised in prior work that the traditional seven-level cohesion hierarchy, being merely an ordinal scale, is unsuited to automated quantitative measurement [2], and that applying it to object-oriented systems requires subjective semantic analysis, making automation difficult [1]. To address this, we restructure the complex hierarchy into five core semantic indicators that an LLM can evaluate in fine detail, and we design a 100-pt rating scale for each indicator. Throughout this rubric, the function is used as the primary unit of evaluation, while module refers to the class- or file-level container that groups related functions.

Table 1. LLM-Based Semantic Cohesion Evaluation Rubric

Evaluation Indicator	Evaluation Criteria
Responsibility Focus (RF)	Does the function exist to fulfill only a single, clearly defined domain purpose?
Functional Consistency (FC)	Do the multiple functions grouped within the module share a common domain context?
Data Cohesion (DC)	Do the functions within the module communicate closely around the same core data or state?
Flow Cohesion (FIC)	Does the execution flow between functions reflect an organic exchange of data, or merely an arbitrarily imposed order?
Role Clarity (RC)	Do identifiers (function/variable names) and comments clearly and truthfully describe the actual internal logic?

Table 1 presents the five evaluation indicators and their corresponding criteria. They are grounded in the classical cohesion taxonomy proposed by Stevens et al. [9] and the Single Responsibility Principle (SRP) from the SOLID principles [10]. Specifically, Responsibility Focus (RF)

 ISSN (Online): 3140-5312

and Functional Consistency (FC) correspond to functional cohesion and the SRP; Data Cohesion (DC) corresponds to communicational cohesion; and Flow Cohesion (FIC) corresponds to sequential cohesion. Role Clarity (RC) has no counterpart in that taxonomy — it was newly introduced so the LLM's natural-language reasoning can judge whether identifiers and comments truthfully reflect the code's actual logic, exposing 'distortions of design intent' that static analysis tools cannot detect.

Table 2. Deduction Severity Criteria for LLM-Based Scoring

Severity Level	Deduction Range	Description
Trivial	−1 to −5 pt	A minor flaw that barely deviates from the principle
Minor	−6 to −15 pt	Deviates from the principle but does not lead to a functional problem
Moderate	−16 to −30 pt	A clear violation with a substantive impact on maintainability
Major	−31 to −45 pt	Multiple principles violated at once, or impact spread across the module
Critical	−46 to −60 pt	A fundamental violation that effectively collapses cohesion

Table 2 summarizes the deduction severity criteria used to score each indicator. If multiple issues are found within a single indicator, the LLM deducts for each and accumulates the total, with a floor of 0. A single violation cannot exceed −60 pt; a more severe case must instead be expressed as multiple individually listed violation reasons whose deductions accumulate, thereby keeping the deduction rationale auditable and helping constrain cross-model variance.

3.2 Proposed Semantic Cohesion Evaluation Process

By applying the single-standard evaluation rubric defined in Section 3.1 to the system, we propose an automated ‘semantic cohesion evaluation process’ that spans from source-code input to the derivation of a final quality score. As shown in Fig. 3, the overall proposed architecture comprises five core processing stages.

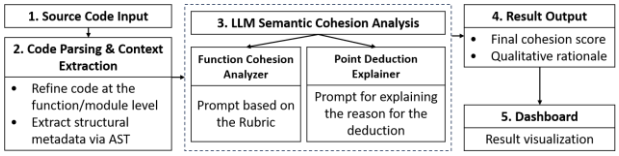

Fig. 3. Rubric-based semantic cohesion evaluation process

Step 1. Source Code Input: The source code to be analyzed is entered into the system.

Step 2. Preprocessing and Context Extraction: The code is refined at the function and module levels, with structural metadata extracted from the AST to capture both syntactic and semantic context.

Step 3. LLM Semantic Cohesion Analysis: An LLM injected with the rubric prompt compares the code's design intent against each of the five indicators. Within this single LLM call, two functional roles operate together: the Function Cohesion Analyzer assigns a numeric score to each rubric indicator, while the Point Deduction Explainer generates a natural-language rationale for every point deducted, referencing the severity levels defined in Table 2.

Step 4. Result Output: Structured data — including the cohesion score, matched indicators, and qualitative rationale — is produced for each function.

Step 5. Dashboard: The results are rendered on a web-based dashboard for intuitive interpretation by the developer.

4. Conclusion and Future Work

We proposed an LLM-based semantic cohesion evaluation model to overcome the limitations of existing static analysis, converting subjective, qualitative judgments into an objective, rubric-driven metric through an automated evaluation process. As future work, we plan to validate the rubric empirically across multiple LLMs and extend the system into an automated refactoring recommendation tool that directly improves code based on the evaluation results.

Conflict of Interest

The authors declare that there are no potential conflicts of interest related to this paper.

Acknowledgement

This research was supported by the Ministry of Science and ICT (MSIT), Korea, under the Sejong SW Convergence Cluster 2.0 Project supervised by the National IT Industry Promotion Agency (NIPA) (Grant No. PJTS080126041007000100030020).

ISSN (Online): 3140-5312

References

[1] L. C. Briand, J. W. Daly, and J. Wüst, “A unified framework for cohesion measurement in object-oriented systems,” Empirical Software Engineering, vol. 3, no. 1, pp. 65-117, 1998.

[2] J. M. Bieman and L. M. Ott, “Measuring functional cohesion,” IEEE Transactions on Software Engineering, vol. 20, no. 8, pp. 644-657, Aug. 1994.

[3] A. Marcus and D. Poshyvanyk, “The conceptual cohesion of classes,” in Proc. IEEE Int. Conf. Software Maintenance (ICSM’05), Budapest, Hungary, 2005, pp. 133-142.

[4] L. H. Etzkorn and H. S. Delugach, “Towards a Semantic Metrics Suite for Object-Oriented Design,” Journal of Object-Oriented Programming, vol. 12, no. 10, pp. 35-44, 2000.

[5] S. Bi et al., “Prometheus: Inducing Fine-Grained Evaluation Capability in Language Models,” in Proc. Int. Conf. Learning Representations (ICLR), 2024.

[6] E. Y. Byun, B. K. Park, W. S. Jang, and Y. C. Kim, “Best practices for cohesion level identification for code reuse in the object-oriented paradigm,” in Proc. KIPS (Korea Information Processing Society) Fall Conf., vol. 23, no. 2, pp. 455-458, 2016.

[7] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,” in Proc. 37th Int. Conf. Neural Information Processing Systems (NeurIPS), 2023.

[8] M. Kawalerowicz, M. Pietranik, and K. Stępniak, “LLMs as Code Review Agents: A Rapid Review and Experimental Evaluation with Human Expert Judges,” in Computational Collective Intelligence (ICCCI 2025), Lecture Notes in Computer Science, vol. 16138, Springer, Cham, 2026.

[9] W. P. Stevens, G. J. Myers, and L. L. Constantine, “Structured design,” IBM Systems Journal, vol. 13, no. 2, pp. 115-139, 1974.

[10] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices. Upper Saddle River, NJ: Prentice Hall, 2003.

Author Information

Eunjin Jeong
now: Undergraduate Student, Department of Software and Communications Engineering, Hongik University, South Korea
Email: jej031020@naver.com
Research Interest: Natural Language Processing, Machine Learning, Prompt Engineering

Janghwan Kim
now: Ph.D Candidate, Department of Software and Communications Engineering, Hongik University, South Korea
Email: lentoconstante@hongik.ac.kr
Research Interest: AI software validation, reinforcement learning-based software validation, software visualization, and requirements engineering.

Chaeyun Seo
2014: Ph.D, Department of Computer Engineering, Hongik University
Email: chaeyun@hongik.ac.kr
Research Interest: Metaverse, Metaverse Contents, Design Thinking, CBD, Software Engineering, BPM, AI, Business Process Modeling.

R. Young Chul Kim
2000: Ph.D. Degree, Software Engineering from the Department of Computer Science, Illinois Institute of Technology, USA
Email: bob@hongik.ac.kr
Research Interest: Test Maturity Model, Model-Based Testing, Metamodeling, Software Process Model, Software Visualization, AI software validation, reinforcement learning-based software validation.