UDC 004.8:004.4
DOI https://doi.org/10.32782/2663-5941/2026.3.1/36

Syromiatnikov M.V.
https://orcid.org/0000-0002-0610-3639
Odesa Polytechnic National University

A CONTEXT-EFFICIENT EVIDENCE-BASED TECHNOLOGY FOR AUTOMATED REPOSITORY-LEVEL SOFTWARE QUALITY EVALUATION USING LARGE LANGUAGE MODELS

Large language models are increasingly applied to software engineering tasks that require reasoning over repository-scale context, including architecture and security assessments. For repositories that still fit into a single prompt, direct full-context ingestion provides a strong reference condition as it exposes the model to the broadest available set of raw repository evidence at once. However, such evaluation remains expensive, weakly interpretable, and sensitive to context limitations, especially when relevant signals are dispersed across source code, configurations, and workflows. Retrieval-only solutions partially reduce prompt size but often fragment architectural context, while deterministic tools provide transparent yet incomplete signals that do not form a coherent repository-level judgment.

This paper proposes a context-efficient evidence-based technology for automated repository-level software quality evaluation. The technology combines a weighted rubric graph, repository-adaptive activation, bounded evidence collection, deterministic tool-supported signals, LLM-assisted scoring, and deterministic hierarchical aggregation to produce final scores with confidence diagnostics. A distinctive feature is the use of SiMAL as a compact machine-readable representation to support activation and targeted evidence acquisition.

The experimental part compares the proposed technology with direct full-context evaluation. Results indicate that while full-context evaluation remains a strong baseline for repositories fitting into a single prompt, the proposed technology substantially reduces peak per-call context requirements by decomposing the assessment into bounded, inspectable scoring steps. Although this decomposition increases total token volume, it enables repository-level evaluation under stricter context budgets, improves evidence traceability, and provides a realistic foundation for adapting smaller open-weight models to scoring tasks.

Keywords: *repository-level evaluation, software quality assessment, large language model, evidence-based scoring, software architecture, static analysis, retrieval-augmented software engineering.*

Formulation of the problem. The rapid advancement of large language models has broadened the scope of artificial intelligence (AI) in software engineering. Early machine learning (ML) systems were mostly limited to code completion or other local file-level tasks, whereas recent models and agent-based systems can operate over multi-file repositories and support project-wide modification, repository summarization, and automated pull request review [1, pp. 2-3; 2, pp. 1-2; 3, pp. 1-3]. Existing AI-assisted review tools mainly provide comments or recommendations for selected fragments, but they rarely produce an end-to-end interpretable, evidence-grounded, and reproducible assessment of repository quality as an integral software artifact. This creates a practical need for methods that can systematically evaluate repositories, beyond isolated code snippets or pull request changes.

At the same time, repository-level quality evaluation remains difficult for several reasons. First, even very large context windows do not guarantee reliable use of all information contained in long inputs [4, pp. 163-167; 5, pp. 3639-3641]. Recent studies show that relevant evidence may be overlooked or used inconsistently when scattered within long prompts. This limitation is especially pronounced in large software projects, where quality-relevant signals are highly dispersed across diverse, heterogeneous artifacts rather than localized in single files. Second, pure retrieval-based approaches partially mitigate the context bottleneck, but they often expose only a subset of local artifacts. Such evidence may be insufficient for architecture-sensitive evaluation tasks, where module boundaries, interface contracts, deployment structure, and cross-component dependencies play a vital role [6, pp. 7150-7152; 7, pp. 1-2].

© Syromiatnikov M.V., 2026
Стаття поширюється на умовах ліцензії відкритого доступу CC BY 4.0

Third, black-box LLM-driven evaluation of a large repository is difficult to audit: it is often unclear which evidence influenced the result, how missing evidence was treated, and whether the obtained judgment can be reproduced given the non-deterministic nature of LLM.

Another practical difficulty is that repository review requires both deterministic and semantic assessment. Some repository properties can be checked directly, for example, the existence of specific files, CI workflows, dependency locks, or leaked secrets. Other properties require broader contextual judgment, such as modularity quality, separation of concerns, architectural consistency, or documentation usefulness. Therefore, a practical repository-level evaluation method should combine explicit deterministic evidence handling with selective model-based reasoning [8, pp. 1117-1120; 9, pp. 614-616].

While direct full-context ingestion serves as a natural baseline for smaller codebases, this approach becomes increasingly expensive, less interpretable, and harder to scale as repository size and artifact diversity grow. This motivates a repository-level evaluation technology that combines bounded-context assessment, evidence grounding, deterministic aggregation, and explicit uncertainty diagnostics. Such technology is relevant to both repository-level software engineering research and practical tasks such as engineering audits, technical due diligence, security triage, and quality monitoring across project portfolios. It is also important because reducing peak context requirements provides a more practical foundation for in-house deployment and adaptation of smaller, open-weight models for repository-level analysis tasks, which may improve data control and reduce security risks associated with external processing.

Analysis of recent research and publications. While recent systems demonstrate substantial progress in applying large language models to software engineering [1, pp. 2-3], repository-level tasks remain considerably more difficult than function-level ones as they require cross-file dependency understanding, navigation over configuration artifacts, and coherent reasoning over architecture and workflow structure [6, pp. 7150-7152; 7, pp. 1-2]. A major limitation in such tasks is the context bottleneck. Simply expanding the context window does not resolve this, as the model's performance frequently fluctuates when navigating weakly structured or highly dispersed project files. Consequently, evaluating quality at the repository scale remains a complex challenge, as cross-component dependencies and architectural evidence are rarely localized and often span multiple representation layers of the same system.

One established direction for mitigating context limits is retrieval- and planning-based repository interaction. RepoCoder applies iterative retrieval and generation for repository-level code completion [10, p. 1], while CodePlan formulates repository-level coding as a planning problem [11, p. 1]. SWE-agent and AutoCodeRover demonstrate that tool-augmented, repository-aware interaction can improve software engineering performance [2, p. 1; 3, p. 1]. However, these approaches primarily focus on code generation, patching, or issue resolution rather than repository-level quality evaluation. Moreover, retrieval-based interaction often fragments understanding by exposing only local snippets detached from their broader architectural context.

Another direction uses structure-aware repository representations. Approaches such as RepoGraph and the Code Graph Model use structure-aware repository representations to provide a global structural overview without relying on flat-file ingestion [7, pp. 1-2; 12, pp. 1-2]. Similarly, SiMAL provides a prompt-efficient domain-specific language for software representation [13, pp. 113-118]. However, these abstractions, which effectively expose links between components, typically enable navigation or bug localization rather than evidence-based scoring with deterministic aggregation.

Automated code review is another directly relevant area. Industrial studies indicate growing interest in LLM-based review systems, including review recommendation, review comment generation, and pull-request evaluation [14, p. 1; 15, p. 1; 16, p. 1]. At the same time, most existing review work remains centered on patches or pull requests rather than full-repository quality scoring with explicit evidence traces. In contrast to the limited scope of automated pull-request reviews, deterministic tooling remains an essential baseline for full-project coverage. Static application security testing, software composition analysis, secret scanners, linters, and complexity analyzers provide transparent and reproducible quantitative signals [9, pp. 614-616; 17, pp. 308-309; 18, p. 1], but are strictly limited to properties formalizable through predefined rules.

To address the semantic limitations of deterministic tools, recent judge-oriented frameworks (e.g., G-Eval, Prometheus, JudgeLM) increasingly rely on criteria-guided LLM evaluation [19, pp. 2511-2513; 20, pp. 1-2; 21, pp. 1-3]. In parallel, weighted aggregation of lower-level software metrics into higher-level quality estimates has been studied as a separate problem in software-quality evaluation [8, pp. 1117-1120; 22, pp. 1-3]. Collectively, these studies support the use

of explicit criteria, fixed scoring dimensions, and constrained outputs. However, a critical gap remains: the field lacks an integrated framework bridging deterministic static analysis with selective semantic reasoning. Specifically, there is a need for a structural representation that goes beyond simple summarization to actively drive routing and targeted evidence acquisition for reproducible, repository-level evaluation.

Task statement. The identified limitations of existing repository-level analysis approaches dictate the need for a technology that integrates explicit evidence handling with the semantic flexibility of LLM-based assessment under practical context constraints. The aim of this research is to develop and substantiate a context-efficient technology for automated repository-level software quality evaluation that synthesizes bounded evidence collection, deterministic tool-supported signals, selective large language model reasoning, and deterministic aggregation, while minimizing peak per-call token requirements and preserving the auditability of intermediate evaluation stages.

To achieve this aim, the following objectives are defined:

1) to design a staged repository-level evaluation architecture that unifies quality criteria, bounded evidence collection, and deterministic score formation;

2) to define an approach to repository analysis that fuses deterministic signals from software artifacts and analysis tools with LLM-based judgment;

3) to incorporate a compact structural repository representation based on SiMAL for repository routing and targeted evidence acquisition;

4) to implement and experimentally evaluate the proposed technology on real software repositories under repository-scale context constraints;

5) to compare the proposed approach with direct full-context repository evaluation, with an emphasis on assessment quality, the trade-off between peak per-call context and total token consumption, and evidence traceability.

Outline of the main material of the study. General architecture of the proposed technology. The proposed repository-level quality evaluation technology is implemented as a staged bounded-context evaluator that transforms rubric-defined criteria and evidence from repository artifacts into a final weighted quality score, along with confidence and coverage diagnostics (Fig. 1).

This includes the following artifacts: rubric groups that define what must be evaluated; evidence profiles that define how evidence must be gathered; the activation logic that selects applicable rubrics; the scoring logic that evaluates rubric criteria; and the aggregation logic that composes all partial judgments into a single final result. At the conceptual level, the technology consists of seven stages: (1) environment configuration, (2) structural context distillation, (3) rubric activation, (4) bounded evidence collection, (5) deterministic and/or LLM-assisted scoring, (6) bottom-up aggregation, and (7) structured report generation.

A key property of the proposed approach is auditability. Each stage produces explicit artifacts, including activation prompts and outputs, evidence bundles, tool outputs, cached LLM responses, token-usage records, and aggregation summaries. Therefore, the system does not operate as a single black-box prompt akin to LLM-as-a-Judge, but rather as an interpretable evaluation pipeline. The source code of the proposed technology is available at https://github.com/niksyromyatnikov/SAGES.

Rubric model and repository-adaptive activation. The evaluation target is formalized as a weighted rubric hierarchy with four distinct levels: repository, module, group, and leaf (Fig. 2). Modules represent broad quality dimensions such as architecture, security, observability, and domain-specific engineering concerns. Groups divide each module into narrower evidence families. Leaves are concrete evaluative statements scored on a bounded numeric scale. In this way, the rubric is used not only as a quality model, but also as an explicit constraint on evaluation, since all later scoring is performed only over predefined groups and leaf criteria rather than by free-form repository judgment.

The rubric is repository-adaptive. Core modules are usually applicable for all repositories, whereas domain-specific ones (e.g., library-, web app-, or ml-oriented) are activated only when supported by repository signals. The activation stage is therefore not a quality assessment stage, but a top-down routing stage. Its function is to determine which groups should be considered and which evidence should be retrieved for them.

The activation step is performed by LLM using a pre-built structured repository context built with SiMAL domain-specific language and produces: a list of programming languages used in repository, inferred repository signals (such as presence or absence of API, database interaction, containerization etc.), module and group activation decisions with rationale, confidence values, and structured evidence hints derived from SiMAL repo schema. These hints are later reused to guide targeted evidence collection. Therefore, activation serves two purposes simultaneously: applicability selection and downstream retrieval planning.

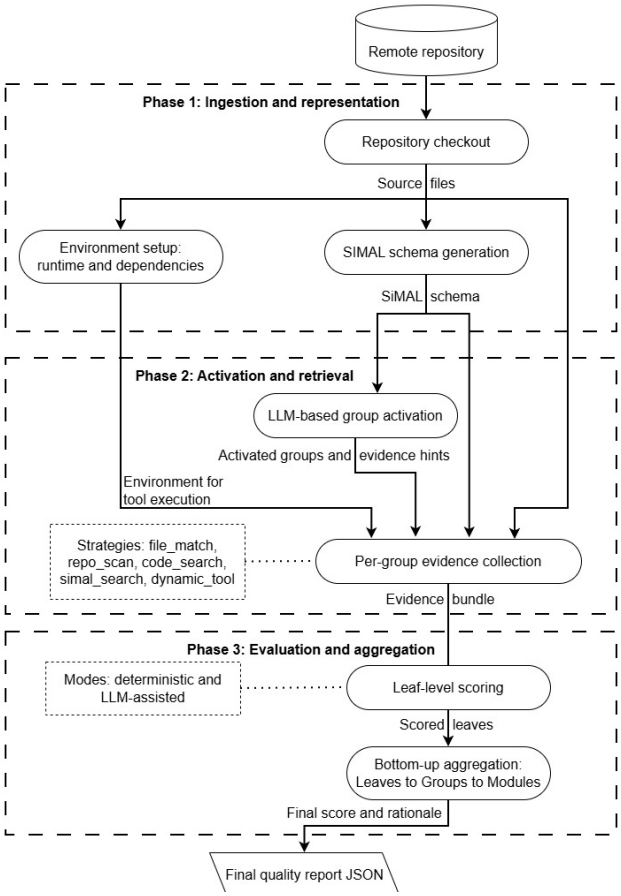

Fig. 1. General architecture of the proposed technology

Source: compiled by the authors

Evidence profiles and bounded evidence collection. Each rubric group references an evidence profile (Fig. 3) that specifies its grading mode and a list of evidence steps. The supported evidence strategies include file matching, repository scan, bounded code search, search over SiMAL schema, and dynamic execution of external analysis tools. Therefore, evidence profiles are the main mechanism that connects the abstract rubric to concrete repository inspection procedures.

The evidence layer is intentionally bounded. File matching detects expected artifacts such as README files, lockfiles, CI workflows, Dockerfiles, changelogs, or security-policy files. A repository scan provides lightweight structural metadata about the top-level layout, entry points, generated artifacts, large binary files, and general repository hygiene. Code search performs bounded regex and snippet retrieval across repository files, skipping generated, minified, binary-like, or overly large directories. SiMAL search performs a similar retrieval procedure over a compact structural schema instead of the raw source tree.

In addition, the proposed technology includes a second retrieval phase driven by activation-time evidence hints derived from the SiMAL schema (path annotations, component or method names, etc.). This phase can directly load path-anchored artifacts, expand globs, or run a targeted search based on components and keywords suggested during

```
modules:
  …
 - id: C
   name: Architecture & dependency structure
   module_weight: 0.10
   applies_if: always
   groups:
    - id: C1
      name: Modularity & boundaries
      group_weight: 0.45
      evidence_profile: simal_modularity_and_interfaces
      leaves:
       - C1.1: "Clear module/package boundaries exist (no 'god' module)."
       - C1.2: "Separation of concerns is evident (UI/business/data, layers, adapters)."
       - C1.3: "Configuration is separated from logic (no scattered magic env reads)."
       …
 …
```

Fig. 2. Excerpt of architectural group C from the rubric model

Source: compiled by the authors

activation. As a result, evidence collection becomes more group-specific and less dependent on generic retrieval. This design also reduces dependence on large repository prompts by constructing smaller evidence bundles tailored to the current scoring group.

Runtime-aware tool integration. To ensure the reliable execution of deterministic tools (e.g., linters, scanners), the technology performs preflight detection of repository languages and runtime dependencies, applying bounded resolution to mitigate environment-mismatch failures. Tools are executed via a unified registry that maps abstract evidence classes to concrete, cached commands. Crucially, a normalization layer converts heterogeneous raw outputs (JSON, XML, or unstructured text) into a compact structured summary, ensuring the scoring stage operates on normalized evidence.

Leaf-level scoring protocol. Following evidence collection, each active group is evaluated leaf by leaf. For each group, the system constructs a structured scoring prompt that includes repository identity, module and group context, the exact predefined leaves, the compact evidence bundle, and activation context. This design intentionally constrains the model to criteria-guided judgment: it does not produce an unconstrained repository review but evaluates only the predefined leaf statements within a fixed, structured output schema. As a result, repository-level evaluation is decomposed into multiple scoring calls rather than a single full-context judgment across the entire repository, effectively neutralizing the context bottleneck.

Two scoring paths are supported. The first path applies deterministic rules for selected evidence profiles. The second path uses an LLM under a strict JSON output contract. If LLM scoring fails, the system does not terminate the whole run. Instead, it emits explicit failure states for the affected leaves and lowers confidence.

Deterministic hierarchical aggregation. Final repository-level judgments are composed deterministically from lower-level leaf assessments. The score of a group is computed as:

$$score_{group} = \frac{\sum_i s_i w_i}{\sum_i w_i} \quad (1)$$

where i – leaf index; s_i – the score of an included leaf; w_i – weight of the i-th leaf.

Thus, the final repository score is not produced in one step, but by transparent bottom-up composition from leaf-level evidence-backed judgments to group-, module-, and repository-level estimates. Under the current policy, leaves with insufficient evidence or tool failure are excluded from the score numerator and denominator rather than automatically penalized. This prevents the system from conflating lack of evidence with poor quality. At the same time, confidence is reduced by evidence coverage and by the fraction of leaves that require human review.

```
evidence_profiles:
 …
 simal_modularity_and_interfaces:
  grading_mode: llm_assisted
  evidence:
   - strategy: simal_search
     max_snippet_lines: 50
     queries:
      - "service "
      - "components:"
      - "api:"
   - strategy: repo_scan
     checks: ["file_tree_top", "languages_detected", "main_entry_candidates"]
   - strategy: code_search
     queries:
      - "internal/"
      - "routes/"
      - "services"
 …
```

Fig. 3. Excerpt of evidence collection with profile for group C1

Source: compiled by the authors

Module scores are computed as weighted means of enabled group scores:

$$score_{module} = \frac{\sum_j score_{group_j} * w_{group_j}}{\sum_j w_{group_j}} \quad (2)$$

where j – group index; $score_{group_j}$ – the calculated score of the j-th enabled group; w_{group_j} – assigned weight of the j-th group.

In addition to numeric scores, the final report contains coverage ratio, evidence sufficiency ratio, counts of non-applicable leaves, insufficient-evidence leaves, failed-tool leaves, and leaves that still require human review. This makes the final result more interpretable than a single undifferentiated repository score.

Role of SiMAL in the proposed technology. Within this pipeline, SiMAL is used as more than a compact repository summary [13, pp. 117-118]. It acts as a machine-readable structural control layer that actively drives two critical bounded-context stages: semantic rubric activation (by exposing components, interfaces, and technology signals) and targeted evidence collection (via schema search and activation-derived hints). This structured routing substantially reduces the need to repeatedly expose the model to the full repository context.

Experimental design. The empirical study was conducted on a curated repository corpus (Table 1) selected to balance technological coverage, repository-type diversity, and practical feasibility of full-context evaluation.

Because the current implementation supports Python, Go, PHP, JavaScript/TypeScript, Docker, and Kubernetes-related artifacts, the corpus was restricted to repositories from these stacks. The final set includes frameworks, reusable libraries, application-style repositories, and infrastructure-oriented projects, while keeping single-call baseline evaluation within the practical context budget of the scoring model. Repository size was measured using the o200k_base tokenizer employed by the GPT model series.

The study compares two repository-level evaluation strategies. The first strategy is a direct full-context single-call baseline, in which the repository source context is passed to one LLM prompt along with the evaluation instructions, and the model directly returns rubric activation decisions and leaf-level judgments. The second strategy is the proposed staged technology, in which the same repository is evaluated through repository-adaptive activation, bounded evidence collection, group-level scoring, and deterministic aggregation. In both cases, final repository-level results are computed by aggregating leaf-level outputs through an identical deterministic procedure, thereby preserving strict comparability of the scores.

Both strategies used GPT-5.2 with medium reasoning effort as the scoring model. Evaluation quality was then assessed by an independent LLM-as-a-Judge protocol implemented with GPT-5.4 and high reasoning effort. The judge received the full repository source content, the evaluation rubric, the normalized candidate assessment object, and the normalized repository tool evidence. It scored each candidate along six rubric-based dimensions: structural score correctness, issue correctness, issue coverage, confidence calibration, interpretability, and activation correctness. In addition, the judge counted concrete error categories, including major and minor score mismatches, missing relevant groups, irrelevant groups, missed issues, overstated issues, and confidence errors. Structural score correctness reflects agreement of module-, group-, and leaf-level scores with repository evidence; issue correctness and issue coverage refer respectively to the validity and completeness of

Table 1

GitHub repositories sampled for evaluation

GitHub repository	Stack	Description	Tokens	Folders	Files
marshmallow-code/marshmallow	Python	data serialization library	193,097	14	98
encode/starlette	Python	ASGI web framework	195,400	15	118
go-chi/chi	Go	web framework	98,821	24	91
spf13/cobra	Go	CLI app framework	165,416	9	63
jesseduffield/lazydocker	Go	terminal application	169,913	29	128
slimphp/slim	PHP	web framework	136,434	26	144
spatie/laravel-permission	PHP	authorization package	168,245	36	160
laravel/lumen-framework	PHP	lightweight framework	59,691	27	60
koajs/koa	JavaScript	Node.js framework	78,977	11	99
pmndrs/zustand	TypeScript	state-management library	178,875	33	124
unjs/h3	TypeScript	server framework	195,922	30	192
dockersamples/example-voting-app	Polyglot / Docker	multi-service application	28,320	13	45

Source: compiled by the authors

identified weaknesses; confidence calibration measures whether confidence values are proportional to evidence strength; interpretability reflects traceability of judgments to evidence; and activation correctness measures whether relevant scoring groups were included and irrelevant ones excluded.

Results and discussion. The evaluation results are analyzed across two dimensions: token usage efficiency and assessed evaluation quality. Table 2 compares the token statistics of the direct full-context baseline against the proposed staged technology. The results show the proposed approach substantially reduces average and maximum per-call token requirements by decomposing repository-level assessment into bounded scoring requests. For example, the marshmallow repository's maximum LLM context size decreased from 291,655 tokens to 46,437 tokens - an 84% reduction in peak memory requirements. Such a reduction reduces dependence on large single-call context windows, making repository-level assessment computationally feasible in constrained deployment settings.

At the same time, this advantage is achieved at the cost of higher total token usage, since the repository is assessed repeatedly across rubric groups. In addition, iterative SiMAL schema generation introduces an extra one-time preprocessing cost, which is reported separately in Table 2. This separation is intentional, as schema generation is not part of the per-group scoring stage itself and may be reused as an intermediate artifact for repeated evaluation of the same repository. Moreover, its token footprint depends heavily on the batching policy and can be further optimized by adjusting the number of files processed per request.

Table 3

LLM-as-a-Judge evaluation results

Metric	Full context LLM	Proposed Technology
Average Scores (1-5, higher is better)	–	–
Score correctness	4	4
Issue correctness	**4.33**	4.17
Issue coverage	**4.08**	4
Confidence calibration	4	4
Interpretability grounding	**4.17**	4
Activation scope correctness	**4.75**	4.5
Total Counts (lower is better)	–	–
Major score mismatches	**3**	7
Minor score mismatches	**13**	17
Missed issues	8	**7**
Overstated issues	**1**	9
Confidence errors	6	9

Source: compiled by the authors

Table 3 presents the LLM-as-a-Judge comparison. The results indicate that direct full-context evaluation remains a strong baseline for repositories that comfortably fit into a single prompt. The baseline achieved slightly higher judged quality across most dimensions, particularly in minimizing overstated issues and major score mismatches. However, the proposed staged technology remained highly competitive, matching the baseline in structural score correctness and confidence calibration, while performing closely in issue coverage.

Eventually, the proposed approach proves it is possible to maintain precise evaluation while eliminating the peak context bottleneck. This, in turn, helps establish a practical foundation for evaluating massive repositories or adapting smaller, open-weight models to complex scoring tasks.

Table 2

Token usage statistics

Repository name	Full context	Proposed technology					
	Avg/Max/ Total tokens	Scoring			Schema generation		
		Avg tokens	Max tokens	Total tokens	Avg tokens	Max tokens	Total tokens
marshmallow	291,655	23,765	46,437	712,949	34,960	72,834	489,433
starlette	291,026	26,574	53,608	1,009,796	30,125	82,174	451,862
chi	179,783	25,277	59,347	960,505	26,121	67,828	600,769
cobra	258,061	23,317	44,864	722,805	38,173	96,316	343,556
lazydocker	251,922	27,591	55,747	938,091	28,332	77,020	764,956
slim	232,520	22,018	49,964	770,612	31,630	64,828	822,366
laravel-permission	269,624	27,096	67,215	948,337	34,289	114,042	994,374
lumen-framework	126,613	22,433	48,361	829,994	24,996	35,826	549,912
koa	147,714	22,256	35,903	801,198	20,784	37,854	228,624
zustand	277,865	27,983	56,789	1,035,368	29,364	70,214	910,303
h3	290,187	29,107	74,943	1,106,057	32,597	84,938	912,739
example-voting-app	58,723	16,991	28,110	713,586	16,110	34,492	209,435

Source: compiled by the authors

Conclusions. This research proposed a context-efficient, evidence-based technology for repository-level software quality evaluation. By leveraging a compact structural representation to drive bounded evidence collection and deterministic hierarchical aggregation, the architecture decomposes the assessment into multiple inspectable scoring steps. Experimental results confirm that while direct full-context evaluation remains a strong baseline with slightly higher judged quality, the proposed technology substantially reduces peak per-call token requirements while preserving an accurate usable quality assessment. Therefore, this approach serves as a highly auditable, bounded-context alternative designed for stricter memory budgets.

This architectural trade-off has significant practical implications. While a single full-context evaluation call regularly consumed between 150,000 and 300,000 input tokens, the largest staged scoring calls were substantially smaller. Although the proposed pipeline currently results in a higher total token consumption across its lifecycle, eliminating the peak context bottleneck establishes a realistic foundation for analyzing massive repositories and enabling secure, in-house deployments. Future research will focus on reducing the total token footprint, optimizing SiMAL generation costs, and fine-tuning medium-sized, open-weight models specifically for these scoring tasks.

Bibliography:

1. Hou X., Zhao Y., Liu Y. et al. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology*. 2024. Vol. 33, no. 8. Art. 220. DOI: 10.1145/3695988.

2. Yang J., Jimenez C.E., Wettig A. et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)* (Vienna, Austria, May 7-11, 2024). 2024.

3. Zhang Y., Pan R., Chen K. et al. AutoCodeRover: Autonomous Program Improvement. *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24)* (Vienna, Austria, September 16-20, 2024). 2024. DOI: 10.1145/3650212.3652131.

4. Liu N.F., Lin K., Hewitt J. et al. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*. 2024. Vol. 12. P. 157–173. DOI: 10.1162/tacl_a_00638.

5. Bai Y., Tu S., Zhang J. et al. LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Vienna, Austria, July 27-August 1, 2025). 2025. P. 3639-3664. DOI: 10.18653/v1/2025.acl-long.183.

6. Du J., Liu Y., Guo H. et al. DependEval: Benchmarking LLMs for Repository Dependency Understanding. *Findings of the Association for Computational Linguistics: ACL 2025*. 2025. P. 7150-7179. DOI: 10.18653/v1/2025.findings-acl.373.

7. Ouyang S., Yu W., Ma K. et al. RepoGraph: Enhancing AI Software Engineering with Repository-level Code Graph. *arXiv*. 2024. URL: https://arxiv.org/abs/2410.14684 (дата звернення: 24.03.2026).

8. Mordal K., Anquetil N., Laval J. et al. Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process*. 2013. Vol. 25, no. 10. P. 1117-1135. DOI: 10.1002/smr.1558.

9. Bennett G., Hall T., Winter E., Counsell S. Semgrep: Improving the Limited Performance of Static Application Security Testing (SAST) Tools. *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24)* (Salerno, Italy, June 18-21, 2024). 2024. P. 614-623. DOI: 10.1145/3661167.3661262.

10. Zhang F., Chen B., Zhang Y. et al. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. *arXiv*. 2023. URL: https://arxiv.org/abs/2303.12570 (дата звернення: 26.03.2026).

11. Bairi R., Sonwane A., Kanade A. et al. CodePlan: Repository-level Coding using LLMs and Planning. *arXiv*. 2023. URL: https://arxiv.org/abs/2309.12499 (дата звернення: 27.03.2026). DOI: 10.48550/arXiv.2309.12499.

12. Tao H., Zhang Y., Tang Z. et al. Code Graph Model (CGM): A Graph-Integrated Large Language Model for Repository-Level Software Engineering Tasks. *arXiv*. 2025. URL: https://arxiv.org/abs/2505.16901 (дата звернення: 27.03.2026). DOI: 10.48550/arXiv.2505.16901.

13. Syromiatnikov M.V., Ruvinskaya V.M. A system internals modeling and annotation language for large language model-driven software engineering. *Applied Aspects of Information Technology*. 2026. Vol. 9, no. 1. P. 103-121. DOI: 10.15276/aait.09.2026.08.

14. Li Z., Lu S., Guo D. et al. Automating Code Review Activities by Large-Scale Pre-training. *arXiv*. 2022. URL: https://arxiv.org/abs/2203.09095 (дата звернення: 24.03.2026). DOI: 10.48550/arXiv.2203.09095.

15. Cihan U., Haratian V., İçöz A. et al. Automated Code Review In Practice. *arXiv*. 2024. URL: https://arxiv.org/abs/2412.18531 (дата звернення: 27.03.2026).

16. Tufano R., Bavota G. Automating Code Review: A Systematic Literature Review. *arXiv*. 2025. URL: https://arxiv.org/abs/2503.09510 (дата звернення: 25.03.2026). DOI: 10.48550/arXiv.2503.09510.

17. McCabe T.J. A Complexity Measure. *IEEE Transactions on Software Engineering*. 1976. Vol. SE-2, no. 4. P. 308-320. DOI: 10.1109/TSE.1976.233837.

18. Basak S.K., Cox J., Reaves B., Williams L. A Comparative Study of Software Secrets Reporting by Secret Detection Tools. *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (New Orleans, USA, October 23-27, 2023). 2023. P. 1-12. DOI: 10.1109/ESEM56168.2023.10304853.

19. Liu Y., Iter D., Xu Y. et al. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (Singapore, December 6-10, 2023). 2023. P. 2511-2522. DOI: 10.18653/v1/2023.emnlp-main.153.

20. Kim S., Shin J., Cho Y. et al. Prometheus: Inducing Fine-grained Evaluation Capability in Language Models. *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)* (Vienna, Austria, May 7-11, 2024). 2024.

21. Zhu L., Wang X., Wang X. JudgeLM: Fine-tuned Large Language Models are Scalable Judges. *arXiv*. 2023. URL: https://arxiv.org/abs/2310.17631 (дата звернення: 26.03.2026). DOI: 10.48550/arXiv.2310.17631.

22. Ulan M., Löwe W., Wingkvist A. Weighted software metrics aggregation and its application to defect prediction. *Empirical Software Engineering*. 2021. Vol. 26. Art. 86. DOI: 10.1007/s10664-021-09984-2.

Сиром'ятніков М.В. КОНТЕКСТНО-ЕФЕКТИВНА ДОКАЗОВА ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА РІВНІ РЕПОЗИТОРІЮ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Великі мовні моделі дедалі ширше застосовуються для розв'язання задач інженерії програмного забезпечення, що потребують міркувань у масштабі всього репозиторію, зокрема для оцінювання архітектури та аналізу безпеки. Для репозиторіїв, які ще можуть бути подані в межах одного запиту, безпосереднє використання повного контексту є сильним еталонним підходом, оскільки надає моделі доступ до максимально повного набору доказів із репозиторію. Проте таке оцінювання залишається дорогим, слабко інтерпретованим і чутливим до обмежень контексту, особливо коли релевантні сигнали розосереджені між вихідним кодом, конфігураційними файлами та робочими процесами. Підходи, що базуються на пошуку, дозволяють частково зменшити обсяг контексту, але вони часто фрагментують архітектурну картину системи, тоді як детерміновані інструменти забезпечують прозорі, проте неповні сигнали, які самі по собі не формують цілісного судження про якість репозиторію.

У статті запропоновано контекстно-ефективну доказову технологію автоматизованого оцінювання якості програмного забезпечення на рівні репозиторію. Запропонована технологія поєднує зважений граф рубрик, адаптивну до властивостей репозиторію активацію, обмежене збирання доказів, детерміновані сигнали від інструментальних засобів, оцінювання за участю великої мовної моделі та детерміновану ієрархічну агрегацію для формування підсумкових оцінок із діагностикою впевненості. Відмінною особливістю підходу є використання SiMAL як компактного машинозчитуваного подання репозиторію, що підтримує активацію релевантних критеріїв і цілеспрямований збір доказів.

В експериментальній частині запропоновану технологію порівняно з прямим оцінюванням повного контексту. Одержані результати показали, що, хоча оцінювання повного контексту залишається сильним базовим підходом для репозиторіїв, які вміщуються в один запит, запропонована технологія істотно знижує пікові вимоги до контексту на один виклик завдяки декомпозиції процесу оцінювання на обмежені та контрольовані кроки. Попри збільшення загального обсягу токенів, такий підхід робить оцінювання на рівні репозиторію більш придатним для умов жорстких бюджетів контексту, підвищує відстежуваність доказів і створює практичне підґрунтя для адаптації менших моделей із відкритими вагами до задач оцінювання.

Ключові слова: *оцінювання на рівні репозиторію, оцінювання якості програмного забезпечення, велика мовна модель, доказове оцінювання, архітектура програмного забезпечення, статичний аналіз, інженерія програмного забезпечення з використанням пошукових підходів.*

Дата першого надходження статті до видання: 27.03.2026
Дата прийняття статті до друку після рецензування: 23.04.2026
Дата публікації (оприлюднення) статті: 19.05.2026