SARV: Structure-Aware Retrieval and Multi-Agent Verification for Software Traceability

Tongjia Ma[1,2], Yuan Huang[1], Ziwei Lei[2*], Chen Ling[2], Yanfei Lv[2], Huihong He[3*]
[1]School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056038, China
[2]Center for Information Research, Academy of Military Sciences, Beijing 100142, China
[3]China Mechanical and Electrical Equipment Tendering Center, Beijing 100142, China
[*]Corresponding authors: leiziwei08@nudt.edu.cn (Ziwei Lei), hehuihong@miitcntc.org.cn (Huihong He)

***Abstract*—In software engineering, establishing accurate traceability links between requirements and source code is essential for many development and maintenance tasks, yet it remains challenging in large and structurally complex software systems. Recent retrieval-augmented generation approaches based on large language models have introduced new opportunities for automated traceability link recovery. However, their effectiveness is often constrained by retrieval quality and the vulnerability of single-pass decision mechanisms to reasoning errors and hallucinations. This paper proposes SARV, a framework that explicitly decouples candidate link generation from link validation. In the retrieval stage, a structure-aware hybrid retrieval strategy is employed to identify potential links. In the validation stage, SARV introduces a multi-agent collaborative mechanism guided by auxiliary information, which applies a reflect-and-correct strategy to transform traceability decisions from a one-shot classification process into a verifiable and error-correctable validation process. Experimental results on multiple public datasets show that SARV achieves competitive and stable performance across diverse project settings.**

***Index Terms*—Software Traceability, Traceability Link Recovery, Retrieval-Augmented Generation, Large Language Models, Multi-Agent Systems**

I. Introduction

In complex software development processes, developers and other stakeholders are required to handle a large number of heterogeneous software artifacts, such as requirements, source code, documentation, and models. As a result, the success of software development largely depends on understanding the relationships among these artifacts. In particular, establishing accurate traceability links between requirements and source code—commonly referred to as Traceability Link Recovery (TLR)—is crucial for downstream tasks such as change impact analysis, compliance verification in safety-critical systems, and software maintenance [1]. However, maintaining traceability links manually is both labor-intensive and error-prone [2].

To address this issue, a wide range of automated TLR approaches have been proposed, including methods based on information retrieval (IR), deep learning, pre-trained models, and heuristic strategies [3], [4]. Among these approaches, textual similarity computation has consistently played a central role. Nevertheless, due to the substantial semantic gap between requirements documents written in natural language and source code expressed in programming languages in terms of lexical expression and abstraction levels, methods that rely solely on textual similarity often fail to capture deeper logical relationships. As a result, the performance of existing approaches remains insufficient to meet the demands of real-world industrial applications [5].

Large language models (LLMs) have demonstrated strong potential in natural language understanding and have proven effective in various software engineering tasks, such as code generation [6], [7], code summarization [8], and API recommendation [9]. However, directly applying LLMs to the TLR task presents several limitations. In large-scale software systems, when models are required to reason over tens of thousands of requirements, design documents, and source code files, they are vulnerable to input length constraints and contextual noise. Moreover, full-scale inference leads to high computational costs and low response efficiency, making it difficult to satisfy the timeliness requirements of real-world engineering scenarios. Consequently, retrieval-augmented generation (RAG) [10] has emerged as a promising solution, in which information retrieval techniques are used to filter relevant artifacts and provide contextual support for LLM-based reasoning.

Despite its potential, existing RAG-based LLM approaches still face several challenges when applied to complex real-world projects. First, the overall performance of current methods is constrained by the recall quality of the retrieval stage, as low-quality retrieved contexts directly determine the upper bound of subsequent LLM-based decisions. In terms of code representation, most existing studies treat source code as isolated textual units. Although some approaches incorporate call relationships or dependency information, such structured knowledge is often not explicitly transformed into reasoning evidence that can be effectively understood by LLMs. As a result, when a requirement is implemented through the collaboration of multiple classes, models struggle to capture the overall business logic and consequently miss critical indirect traceability links. From a reasoning perspective, existing frameworks typically treat LLMs as one-pass decision makers. This lack of reflection and self-correction mechanisms makes them highly susceptible to hallucinations and superficial textual similarity, leading to high false-positive rates in scenarios where domain concepts overlap but the underlying logic differs fundamentally.

To address these challenges, we propose SARV, a multi-

agent verification framework for retrieval-augmented software traceability. SARV explicitly separates candidate link generation in the retrieval stage from reliable link verification. In the retrieval stage, a structure-aware hybrid retrieval mechanism is introduced, in which LLM-based semantic structure parsing is used to transform requirements and source code into unified natural language descriptions, thereby bridging the semantic gap between heterogeneous artifacts. On this basis, SARV combines BM25-based lexical retrieval with relevance propagation over code dependency graphs to obtain candidate traceability links. In the verification stage, SARV moves beyond simple one-pass LLM decisions and instead constructs an auxiliary-information-guided dual-agent collaborative strategy, in which auxiliary information, including call relationships and contextual descriptions, is explicitly incorporated as reasoning evidence, transforming single-pass decisions into iterative reasoning with self-correction capabilities.

The main contributions of this paper are summarized as follows:

- We propose SARV, a multi-agent verification framework for retrieval-augmented software traceability. By explicitly decoupling candidate link generation from link verification, SARV provides a new approach for constructing stable and reliable TLR pipelines in complex software systems.
- We design a structure-aware hybrid retrieval strategy that leverages LLM-based semantic parsing to unify the representations of requirements and source code, and combines lexical retrieval with dependency propagation to enhance retrieval quality.
- We introduce an auxiliary-information-guided reflect-and-correct multi-agent verification mechanism. By incorporating auxiliary information as reasoning evidence and employing dual-agent collaboration to review and correct decisions, this mechanism effectively reduces hallucinations and false positives arising from one-pass LLM reasoning in complex scenarios.
- We conduct systematic evaluations on multiple public requirement–code traceability datasets. Experimental results demonstrate that SARV achieves superior overall performance and stability across diverse project settings compared to state-of-the-art approaches.

II. Related Work

This section focuses on state-of-the-art techniques for Requirement-to-Code Traceability Link Recovery (TLR), systematically reviews the landscape of existing methods, and elaborates on recent advancements in the application of Large Language Models (LLMs) within this domain.

A. Classical Approaches for Requirement–Code Traceability Link Recovery

Requirement–code Traceability Link Recovery (TLR) aims to identify semantic correspondences among different software artifacts and serves as a fundamental basis for tasks such as change impact analysis, software maintenance, and consistency checking. Early traceability studies primarily relied on information retrieval (IR) techniques, such as VSM, LSI, and LDA, to recover links by computing textual similarity [11]–[15]. Although these approaches exhibit good scalability, their effectiveness is often limited by the lexical gap between requirements and source code [16].

Beyond IR-based methods, several representative approaches model TLR directly as a link validity determination problem and incorporate stronger reasoning capabilities or fine-grained semantic modeling. For example, COMET, proposed by Moran et al. [17], integrates similarity evidence across multiple stages using Bayesian reasoning and enhances results by leveraging feedback and transitive relationships. Fine-Grained Trace Link Recovery (FTLR), introduced by Hey et al. [2], adopts "requirement statement–code method" pairs as fine-grained analysis units and utilizes word embeddings to capture more detailed semantic associations, thereby improving the precision of sentence-level traceability link recovery.

To alleviate the impact of lexical mismatch, Gao et al. [18], [19] enhance artifact representations by incorporating co-occurring term pairs from requirements and code, thereby improving semantic consistency within traditional IR pipelines. In addition, some studies extend traceability relationships by introducing intermediate artifacts or exploiting transitive links. Nishikawa et al. [20] propose leveraging intermediate artifacts to enable indirect traceability across heterogeneous artifact types. Rodriguez et al. [21] conduct a systematic analysis of path aggregation strategies under multiple source–intermediate–target paths, comparing functions such as maximum, summation, and weighted summation. With the advancement of pre-trained language models, researchers have further introduced Transformer-based pre-trained models, such as BERT [22] and its variants, to obtain context-sensitive semantic representations [23], [24].

B. Applications of Large Language Models in TLR

With the emergence of large language models (LLMs), researchers have begun to explore their potential applications in TLR tasks. Rodriguez et al. [25] evaluate the capability of LLMs to directly determine traceability links through natural language prompting. Their results indicate that LLMs are able to understand domain-specific knowledge, such as abbreviations and implicit semantics, but their performance is highly sensitive to prompt design. Zou et al. [26] further conduct a systematic evaluation of multiple mainstream LLMs on TLR tasks and point out that relying solely on textual similarity is insufficient, while structural information and semantic reasoning capabilities play a critical role in achieving better performance.

However, directly applying LLM reasoning to all requirement–code pairs in large-scale projects faces challenges related to context length limitations and high computational costs. To address these issues, retrieval-augmented generation (RAG) [10] has been introduced into the TLR domain to improve reasoning quality while maintaining scalability. Ali et al. [27]

propose the RARTG approach, which combines keyword-based indexing with vector-based retrieval to identify relevant classes and code fragments from codebases and uses them as contextual input for LLM-based decision making. Their method achieves superior performance compared to traditional IR approaches on multiple public datasets.

The LiSSA framework, proposed by Fuchß et al. [28], [29], further validates the effectiveness and generality of RAG in TLR tasks. LiSSA constructs retrieval contexts using code chunking strategies at different granularities, such as fixed-line segmentation or method-level splitting, and employs LLMs as decision makers for candidate traceability links. The framework demonstrates favorable precision and scalability across multiple requirement-to-code tasks, and its accompanying datasets have been publicly released to facilitate subsequent comparative studies.

Different from the aforementioned studies, existing LLM- and RAG-based TLR approaches typically treat source code as raw programming language text during the retrieval stage, failing to effectively bridge the representational heterogeneity between requirements and source code, and rarely model structured dependencies among code artifacts explicitly. In the reasoning stage, these methods often employ LLMs as one-pass decision makers, lacking systematic mechanisms for reviewing and correcting reasoning outcomes, and are therefore susceptible to hallucinations and superficial textual similarity. To address these limitations, this paper unifies the representations of requirements and source code through semantic structure parsing during retrieval and generates high-quality candidate links by combining lexical retrieval with structured dependency propagation. On this basis, a multi-agent collaborative verification mechanism is introduced, which improves traceability precision and result interpretability through a "preliminary reasoning–review and correction" workflow, rather than relying on prompt engineering techniques.

III. Method

This section introduces the two core components of the SARV automated traceability link recovery framework: a structure-aware hybrid retrieval mechanism and an auxiliary-information-guided multi-agent collaborative verification mechanism.

A. Structure-Aware Hybrid Retrieval Stage

As illustrated in Fig. 1, the structure-aware hybrid retrieval stage generates a high-confidence candidate set through a cascaded strategy. Specifically, heterogeneous textual representations are first unified via semantic parsing; preliminary filtering is then performed based on hybrid similarity computation; finally, code dependency propagation is introduced to "activate" potential implicit traceability links.

1) Data Preprocessing and Semantic Structure Extraction: In requirement–code traceability tasks, requirements documents and source code differ substantially in their forms of expression. Therefore, the primary objective of this subsection is to normalize both requirements and code into alignable natural language representations. Meanwhile, we construct a BM25 index to preserve precise sparse retrieval capability for program symbols.

To bridge the heterogeneity between requirements and code, we further introduce semantic structure extraction. Leveraging the strong comprehension and abstraction capabilities of LLMs, unstructured requirement texts and semi-structured code snippets are unified into a structured representation along four semantic dimensions: *core functionality* w_c, which captures high-level business intent for goal alignment; *key entities* w_e, which identify core data objects and roles to ensure accurate alignment between domain concepts and code classes; *operational process* w_p, which abstracts away low-level syntactic details while focusing on logical transitions and algorithmic steps to capture deeper behavioral consistency; and *business context* w_s, which defines the contextual constraints under which a functionality is triggered. Through this structured extraction, complex code implementations are transformed into requirement-isomorphic natural language descriptions, providing a stable and interpretable foundation for subsequent similarity computation.

Different types of requirements exhibit distinct information distributions across semantic dimensions. For example, data modeling or persistence-related requirements rely more heavily on entity and attribute information, whereas algorithmic or business-rule-oriented requirements emphasize operational processes and conditional constraints. Motivated by this observation, we further introduce an adaptive weight generation mechanism.

Specifically, during semantic structure extraction, the LLM simultaneously acts as a domain expert and dynamically assigns a set of normalized importance weights $W_R = \{w_c, w_e, w_p, w_s\}$ based on the semantic characteristics of the requirement.

For interaction-oriented requirements such as "when the user clicks the delete button," the model automatically increases the weight of the business context dimension w_s. For computation-oriented requirements such as "calculating interest using a specific formula," the weight of the operational process dimension w_p is significantly amplified. This mechanism enables the model to simulate the cognitive behavior of human developers during traceability analysis, adaptively focusing on the most critical semantic dimensions according to task type, and thereby achieving robust similarity computation across diverse query scenarios.

2) Vector Generation and Hybrid Similarity Computation: To jointly capture deep semantic relevance and precise lexical matching, we design a hybrid scoring mechanism. Specifically, based on the adaptive weights W_R obtained in the previous stage, we first compute a weighted cosine similarity between requirement and code vectors. Meanwhile, the BM25 score, normalized using Min–Max normalization and denoted as S'_{BM25}, is incorporated as a complementary signal. The final hybrid relevance score S_{final} is defined as:

$$S_{\text{final}} = \alpha \sum_{d \in D} w_{(R,d)} \cdot \cos\big(u_{(R,d)}, v_{(C,d)}\big) + (1-\alpha) S'_{\text{BM25}} \quad (1)$$

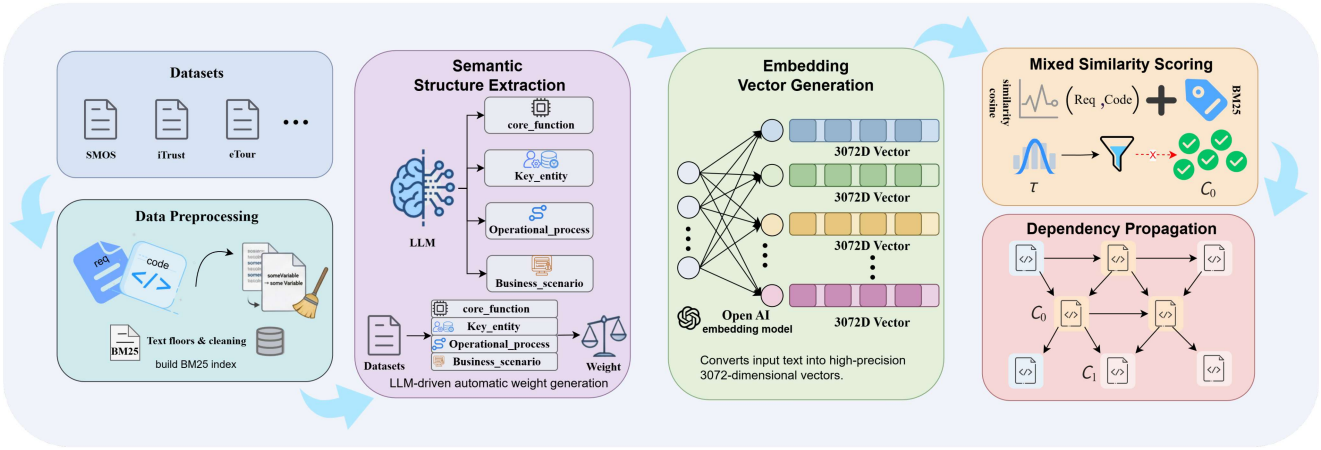

Fig. 1. Overview of the structure-aware hybrid retrieval stage.

where $u_{(R,d)}$ and $v_{(C,d)}$ denote the vector representations of the requirement and the code, respectively, under semantic dimension d, and $\alpha \in [0, 1]$ controls the relative contribution of semantic similarity and lexical matching signals.

Based on the overall distribution of S_{final}, we further introduce an adaptive threshold $\tau = \mu + \lambda\sigma$ to dynamically truncate low-confidence candidates, thereby generating the initial candidate set C_0.

3) Dependency Propagation: Although some target code artifacts exhibit low direct textual similarity with the requirement, they are often connected to high-confidence code through invocation or dependency relationships. To exploit such structural cues, we construct a code dependency graph $G = (V, E)$ and introduce a structure-aware score propagation mechanism.

Starting from the initial candidate set C_0 as seed nodes, relevance scores are propagated to neighboring nodes along dependency edges. To suppress noise propagation, only the Top-k neighboring nodes with the highest hybrid relevance scores are selected for weighted enhancement. The propagated score for a candidate node c' is defined as:

$$S_{\text{prop}}(c') = S_{\text{final}}(c') + \frac{\beta}{k} \sum_{c \in N_{\text{top-}k}(c')} S_{\text{final}}(c) \tag{2}$$

where c' denotes a candidate node being enhanced during the propagation process, $N_{\text{top-}k}(c')$ represents the set of Top-k neighbors of c' ranked by S_{final}, and β controls the strength of structural propagation.

After propagation, nodes whose scores exceed the predefined threshold are incorporated into the expanded candidate set.

B. Auxiliary-Information-Guided Multi-Agent Collaborative Verification

As illustrated in Fig. 2, this stage adopts a dual-agent collaborative architecture to perform auditable verification of candidate traceability links. By incorporating auxiliary information derived from code structure and employing a "reflection–correction" mechanism, the proposed approach enables systematic review and refinement of verification decisions.

1) Auxiliary Information Generation: We adopt a lightweight static analysis approach to extract structural dependency relationships among code files from the source code repository. For each code file to be verified, we characterize its *invocation context* (i.e., which code files invoke this file within the business workflow) as well as its *external dependencies* (i.e., which other code files and key methods are invoked by this file). These structured dependency relations are organized into semi-structured natural language descriptions—specifically detailing 'Context: This code is called by [Callers]' and 'Dependencies: [Callee] (uses: [methods])'—which serve as auxiliary inputs for the multi-agent reasoning module.

By incorporating such structured auxiliary information, the verification stage is able to analyze requirement–code relationships from the perspective of a code artifact's functional role and business position within the overall system, rather than relying solely on local code snippets or superficial keyword overlap.

2) Agent A: Agent A serves as the first analytical role in the verification stage and is responsible for performing preliminary association reasoning on requirement–code candidate pairs. Rather than directly producing a final decision, its objective is to generate plausible linkage hypotheses through systematic analysis, while simultaneously outputting intermediate reasoning results that can be inspected and reviewed in subsequent stages.

At the input level, Agent A receives the requirement text, the code text, and the auxiliary structural information generated via static analysis. During inference, the agent is explicitly guided to adopt step-by-step Chain-of-Thought reasoning, integrating information from the requirement, the code, and the auxiliary inputs in a progressive manner. The reasoning process is constrained along three dimensions: (1) consistency between the functional objectives of the requirement and the

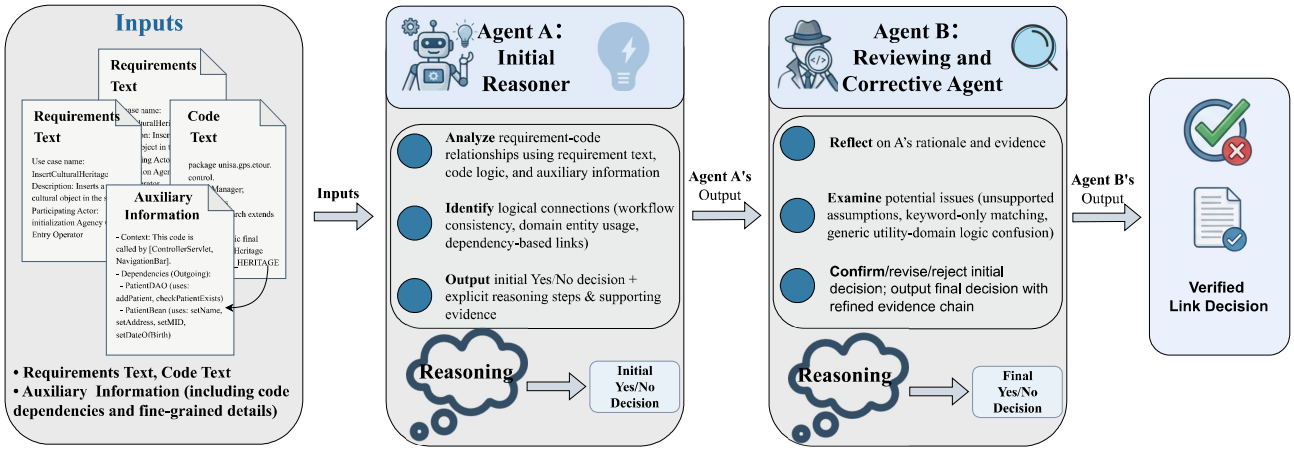

Fig. 2. Overview of the auxiliary-information-guided multi-agent collaborative verification stage.

core behaviors implemented in the code; (2) the supportive or connective role played by the code within the overall business workflow; and (3) whether the structural dependency information provides concrete evidence for the hypothesized association.

Under these constraints, Agent A outputs not only a preliminary decision, but also the corresponding reasoning process, key supporting evidence, and potential sources of uncertainty, thereby facilitating subsequent review and correction by other agents in the collaborative verification framework.

3) Agent B: After Agent A produces a preliminary judgment, Agent B intervenes in the verification process as an independent reviewing role. Rather than regenerating association reasoning from scratch, the primary responsibility of Agent B is to conduct a systematic review and correction of the existing reasoning process.

To this end, Agent B receives the complete requirement text, the code text, the auxiliary structural information, as well as the decision output by Agent A together with its corresponding Chain-of-Thought reasoning. The review focuses on three key aspects: (1) whether the key evidence cited in the reasoning truly reflects alignment at the level of business logic or system functionality, rather than remaining at superficial semantic similarity; (2) whether the reasoning steps exhibit issues such as logical leaps, evidence over-generalization, or unwarranted conclusion expansion; and (3) whether the structural position of the code within the invocation and dependency network is consistent with the functional module implicated by the requirement.

By treating the Chain-of-Thought reasoning itself as an explicit object of scrutiny, Agent B is able to identify and correct potential logical deficiencies in the preliminary analysis, thereby enabling more robust and reliable verification of candidate traceability links.

IV. Evaluation

This section presents a systematic evaluation of the proposed approach. The experiments are conducted on the task of requirement-to-code Traceability Link Recovery (TLR) and compare our method against multiple representative baseline approaches, with the goal of analyzing its effectiveness and robustness across different project scenarios.

A. Datasets

We evaluate SARV on four widely used datasets: SMOS, eTour, iTrust, and Dronology, which cover diverse domains (education, tourism, healthcare, and aerospace) as summarized in Table I. For Dronology, we employ both RE→Code and DD→Code mappings [28], where Design Definitions (DD) are considered low-level requirements closer to implementation details.

TABLE I
Overview of the Datasets

Dataset	Language[a]		Statistics[b]		
	NL	Code	Req	Code	TL
SMOS	IT	Java	67	100	1044 (15.6%)
eTour	EN	Java	58	116	308 (4.6%)
iTrust	EN	Java	131	226	286 (1.0%)
Dronology (RE)	EN	Java, Python	99	423	602 (1.4%)
Dronology (DD)	EN	Java, Python	211	423	740 (0.8%)

[a]IT = Italian, EN = English.
[b]Req = Requirements, TL = Traceability Links.

B. Evaluation Metrics

To assess the performance of the proposed approach, we adopt evaluation metrics that are commonly used in the TLR literature, including Precision, Recall, and the F_β score. In fully automated TLR settings, the F_1 score is used as the primary evaluation metric. In semi-automated scenarios, where coverage of true traceability links is emphasized, we additionally report the F_2 score, which assigns a higher weight to recall.

These metrics enable direct and fair comparison between the proposed approach and existing traceability link recovery methods.

TABLE II
PERFORMANCE COMPARISON ON INDIVIDUAL DATASETS AND OVERALL AVERAGES (MERGING ABLATION STUDY).

Approach		SMOS				eTour				iTrust				Dronology (RE)				Dronology (DD)				Avg.		w. Avg.	
		P	R	F_1	F_2	P	R	F_1	F_2	P	R	F_1	F_2	P	R	F_1	F_2	P	R	F_1	F_2	F_1	F_2	F_1	F_2
VSM_{opt}		.430	.414	.422	.417	**.557**	.427	.483	.448	.208	.227	.217	.223	**.844**	.087	.158	.106	**.846**	.071	.131	.087	.282	.256	.283	.257
LSI_{opt}		.415	.430	.422	.427	.452	.453	.453	.453	.251	.255	.253	.254	.333	.107	.162	.124	.757	.074	.135	.090	.285	.270	.285	.268
$COMET_{opt}$		.195	.572	.291	.413	.410	.468	.437	.455	**.361**	.231	.282	.249	–	–	–	–	–	–	–	–	–	–	–	–
FTLR		.444	.331	.380	.349	.379	.633	.474	.558	.165	.339	.222	.280	.183	.161	.172	.165	.129	.154	.140	.148	.278	.300	.273	.277
$FTLR_{opt}$		.314	.588	.409	.501	.505	.597	.548	.576	.234	.241	.238	.240	.184	.170	.177	.173	.140	.147	.144	.146	.303	.327	.294	.329
RARTG		**.608**	.126	.209	.150	.543	.242	.334	.272	.289	.292	.290	.291	–	–	–	–	–	–	–	–	–	–	–	–
LiSSA		.590	.195	.294	.226	.409	.734	.526	**.633**	.199	.451	.276	.360	.226	.344	**.273**	**.311**	.177	.380	.241	.309	.322	.368	.299	.319
Ours	*R1*	.225	.770	.348	.528	.129	.935	.227	.407	.025	**.899**	.049	.111	.041	**.789**	.077	.170	.075	**.664**	.135	.252	.167	.294	.199	.335
	R2	.296	.767	.427	.582	.136	**.938**	.238	.431	.057	.787	.107	.222	.032	.581	.064	.131	.088	.568	.144	.272	.196	.328	.233	.364
	Agent A	.414	.652	.506	**.585**	.242	.906	.382	.585	.101	.745	.177	.327	.056	.558	.103	.201	.108	.555	.181	.304	.270	.400	.300	.413
	Agent B	.499	.554	**.525**	.542	.552	.640	**.593**	.620	.221	.521	**.311**	**.410**	.214	.310	.253	.284	.220	.353	**.271**	**.315**	**.391**	**.434**	**.394**	**.429**

TABLE III
REPRESENTATIVE CASES WHERE AGENT B CORRECTS FALSE POSITIVES PRODUCED BY AGENT A (REASONING SUMMARIZED).

Case	Requirement–Code Pair	Agent A (Initial Reasoning)	Agent B (Correction Rationale)
C1	UC1 ↔ ICulturalHeritageCommonManager	Judged as related due to shared domain entity and package-level proximity.	Rejected because the requirement specifies a delete operation, while the code only implements read and feedback-modification logic.
C2	UC1 ↔ TagAgencyOperatorManager	Considered related due to the presence of a delete-related method and invocation proximity.	Rejected because the code deletes tag entities rather than cultural heritage objects required by the use case.

TABLE IV
FILTERING PERFORMANCE OF AGENT B (ACCURACY BREAKDOWN).

Dataset	Correctly Filtered	Incorrectly Filtered	Accuracy
SMOS	385	103	78.89%
eTour	715	82	89.71%
iTrust	1378	64	95.56%
Dronology (RE)	4938	150	97.05%
Dronology (DD)	2475	150	94.29%
Overall	**9891**	**549**	**94.74%**

C. Model Configuration

1) Configuration of the Structure-Aware Hybrid Retrieval Stage: For semantic parsing, we employ the DeepSeek-V3.2 Chat model with the temperature set to 1.0, and use the text-embedding-3-large model to generate vector representations. To balance semantic matching and precise lexical matching, the fusion coefficient α is set to 0.7, allowing semantic similarity to serve as the dominant signal while preserving the advantages of BM25 in matching program identifiers and abbreviated terms. The adaptive threshold parameter λ is set to 1.2. For the Dronology (DD) dataset, which has a significantly larger search space, λ is increased to 2.5 in order to control the candidate set size and reduce the computational cost of subsequent LLM-based verification. This adjustment is made primarily due to computational cost constraints rather than performance-oriented tuning.

During the dependency propagation (graph propagation) stage, to achieve a balance between incorporating structural information and suppressing noise, the number of neighbors is set to $k = 3$, and the propagation coefficient β is set to 0.3. This configuration helps prevent semantic drift caused by excessive reliance on neighboring nodes.

2) Agent Model Configuration: To balance performance and computational cost, we adopt heterogeneous model collaboration in the verification stage. Agent A, responsible for preliminary reasoning, employs the `doubao-seed-1-6-lite` model, while Agent B, which performs review and correction, uses the DeepSeek-V3.2 Chat model. The temperature parameter is set to 1.0 for both agents, following the official recommendations of the corresponding models for data processing tasks.

D. Baselines

1) Internal Baselines and Ablation Variants: To quantify the contribution of semantic structure extraction and the roles of different agents, we construct a series of progressively enhanced internal baselines as follows:

Unstructured Retrieval (denoted as R1): As an ablation baseline, this setting removes semantic structure extraction. The full texts of requirements and code artifacts are directly fed into the embedding model to generate global vector representations. This baseline is used to assess the effectiveness of semantic structure parsing.

Structure-Aware Hybrid Retrieval (denoted as R2): This baseline adopts the complete retrieval pipeline of SARV but does not introduce any agent-based verification. It serves as a benchmark for evaluating the performance of retrieval alone.

Agent A: In this setting, based on the candidate set generated by R2, only Agent A is introduced to perform preliminary reasoning and decision-making on requirement–code candidate pairs.

Agent B: This baseline corresponds to the full proposed method (SARV), in which Agent B is introduced on top of Agent A to conduct review and correction.

2) Requirement-to-Code Traceability Link Recovery Baselines: In addition to the internal baselines, we also include several representative external methods for comparison. The technical details of these methods are described in Section II.

For FTLR, we report the best-performing configuration provided in the original work (denoted as FTLR_OPT). The results of COMET on the SMOS, eTour, and iTrust datasets are

directly taken from the reproduction package released by Hey *et al.* [30]; due to the lack of a reproducible implementation, COMET is not evaluated on the Dronology dataset. For VSM and LSI, we directly adopt the experimental results reported by Dominik Fuchs *et al.* in the LiSSA study [28]. This work is based on standard reproduction code with systematic parameter search, and its results can be regarded as the performance upper bounds of these methods on the corresponding datasets.

In addition, the comparison results of LiSSA are taken from its original paper using the GPT-4o model with Chain-of-Thought prompting under the best-performing configuration [28]. The results of RARTG are directly adopted from the best parameter variant (RARTG-C1) reported in the original paper [27].

E. Results

1) Overall Performance and Ablation Analysis: The experimental results in Table II demonstrate the progressive performance improvements contributed by different components of the SARV framework. Compared to the unstructured retrieval baseline (R1), the structure-aware hybrid retrieval (R2) achieves consistent gains in both weighted average F1 and F2 scores (e.g., w. Avg F1 increases from 0.199 to 0.233). This validates that unifying artifact representations via semantic structure parsing effectively bridges the representational gap and improves candidate quality. Furthermore, R2 reduces the candidate search space by 59.6% to 94.7%, significantly lowering the computational overhead for subsequent reasoning.

Building upon the retrieval stage, the introduction of multi-agent verification further improves overall performance. Agent A effectively eliminates a large portion of obvious false positives through preliminary reasoning, leading to notable precision gains. More importantly, the complete SARV configuration with Agent B achieves a more favorable precision–recall trade-off on most datasets and attains the best weighted-average F_1 score of 0.394, indicating the effectiveness of the proposed collaborative verification mechanism.

Wilcoxon signed-rank tests confirm that these improvements are statistically significant ($p < 0.05$). Given the substantial costs of LLM inference, we focused this validation on the comparison between the configuration with the best average F_1 score (i.e., Agent B) and the baselines.

2) Insights into Review and Correction Behaviors: To analyze the correction effectiveness of Agent B, we conduct both quantitative and qualitative analyses of its rejection behavior. As summarized in Table IV, Agent B correctly filtered 9,891 out of 10,440 rejected candidate links, achieving a correction accuracy of 94.74%, while introducing only a small number of incorrect rejections.

Further inspection of representative cases in Table III reveals that Agent B effectively acts as a "logical auditor." Specifically, it is capable of identifying superficial traps caused by domain-term overlap—such as cases where keyword similarity exists without actual functional alignment—and correcting hallucinated judgments produced by Agent A. Through such review and correction, Agent B not only improves precision but also enhances the interpretability and reliability of the final traceability results.

V. Conclusion and Future Work

This paper proposes the SARV framework, which effectively addresses the semantic gap and model hallucination issues in automated traceability link recovery by explicitly decoupling candidate generation from link verification. Experimental results demonstrate that the proposed structure-aware hybrid retrieval strategy compresses the search space by 59.6% to 94.7% while maintaining high recall, thereby significantly reducing computational cost. Moreover, the reflection–correction-based multi-agent collaborative verification mechanism further improves the precision–recall trade-off and enhances the interpretability of the recovered traceability links.

Future work will primarily focus on two directions. First, *cross-domain extension*: we plan to extend the proposed verification framework to heterogeneous traceability scenarios such as requirement–requirement and model–code traceability, in order to support complex development paradigms including Model-Driven Development and Model-Based Systems Engineering. Second, adaptive parameter optimization: for key hyperparameters that are currently manually specified in the framework, we will explore automated configuration mechanisms that dynamically search for optimal parameter combinations based on project-specific code characteristics and data distributions. This direction aims to substantially improve the generality and practical applicability of the proposed approach.

Acknowledgment

This paper was supported by the National Key Research and Development Program of China (No. 2024YFB3312905).

References

[1] W. van Oosten, R. Rasiman, F. Dalpiaz, and T. Hurkmans, "On the effectiveness of automated tracing from model changes to project issues," *Inf. Softw. Technol.*, vol. 160, no. C, Aug. 2023. [Online]. Available: https://doi.org/10.1016/j.infsof.2023.107226

[2] T. Hey, F. Chen, S. Weigelt, and W. F. Tichy, "Improving traceability link recovery using fine-grained requirements-to-code relations," in *37th IEEE International Conference on Software Maintenance and Evolution (ICSME), 27.9.-01.10.2021 Luxemburg*. Institute of Electrical and Electronics Engineers (IEEE), 2021, p. 12–22.

[3] B. Wang, Z. Zou, H. Wan, Y. Li, Y. Deng, and X. Li, "An empirical study on the state-of-the-art methods for requirement-to-code traceability link recovery," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 36, no. 6, Jul. 2024. [Online]. Available: https://doi.org/10.1016/j.jksuci.2024.102118

[4] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 14–24. [Online]. Available: https://doi.org/10.1145/3387904.3389251

[5] G. Antoniol, J. Cleland-Huang, J. H. Hayes, and M. Vierhauser, "Grand challenges of traceability: The next ten years," *CoRR*, vol. abs/1710.03129, 2017. [Online]. Available: http://arxiv.org/abs/1710.03129

[6] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Evaluating large language models in class-level code generation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639219

[7] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3660810

[8] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, and X. Liao, "Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3608134

[9] Q. Huang, Z. Wan, Z. Xing, C. Wang, J. Chen, X. Xu, and Q. Lu, "Let's chat to find the apis: Connecting human, llm and knowledge graph through ai chain," in *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '23. IEEE Press, 2024, p. 471–483. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00075

[10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[11] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery: A ten-year retrospective," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1. [Online]. Available: https://doi.org/10.1145/3387904.3394491

[12] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation: A retrospective," *IEEE Trans. Softw. Eng.*, vol. 51, no. 3, p. 825–832, Mar. 2025. [Online]. Available: https://doi.org/10.1109/TSE.2025.3534027

[13] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 95–104. [Online]. Available: https://doi.org/10.1145/1806799.1806817

[14] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 133–142.

[15] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requir. Eng.*, vol. 20, no. 3, p. 281–300, Sep. 2015. [Online]. Available: https://doi.org/10.1007/s00766-013-0199-y

[16] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk, "Enhancing software traceability by automatically expanding corpora with relevant documentation," in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 320–329.

[17] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, and J. Johnson, "Improving the effectiveness of traceability link recovery using hierarchical bayesian networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 873–885. [Online]. Available: https://doi.org/10.1145/3377811.3380418

[18] H. Gao, H. Kuang, W. K. G. Assunção, C. Mayr-Dorn, G. Rong, H. Zhang, X. Ma, and A. Egyed, "Triad: Automated traceability recovery based on biterm-enhanced deduction of transitive links among artifacts," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639164

[19] H. Gao, H. Kuang, K. Sun, X. Ma, A. Egyed, P. Mäder, G. Rong, D. Shao, and H. Zhang, "Using consensual biterms from text structures of requirements and code to improve ir-based traceability recovery," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3556948

[20] K. Nishikawa, H. Washizaki, Y. Fukazawa, K. Oshima, and R. Mibe, "Recovering transitive traceability links among software artifacts," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 576–580.

[21] A. D. Rodriguez, J. Cleland-Huang, and D. Falessi, "Leveraging intermediate artifacts to improve automated trace link retrieval," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 81–92.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423/

[23] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," in *Proceedings of the 43rd International Conference on Software Engineering*, ser. ICSE '21. IEEE Press, 2021, p. 324–335. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00040

[24] T. Hey, J. Keim, and S. Corallo, "Requirements classification for traceability link recovery," in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, 2024, pp. 155–167.

[25] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 2023, pp. 455–464.

[26] Z. Zou, B. Wang, P. Liang, T. Bi, and H. Jin, "Natural language-programming language software traceability link recovery needs more than textual similarity," 2025. [Online]. Available: https://arxiv.org/abs/2509.05585

[27] S. J. Ali, V. Naganathan, and D. Bork, "Establishing traceability between natural language requirements and software artifacts by combining rag and llms," in *Conceptual Modeling*, W. Maass, H. Han, H. Yasar, and N. Multari, Eds. Cham: Springer Nature Switzerland, 2025, pp. 295–314.

[28] D. Fuchß, T. Hey, J. Keim, H. Liu, N. Ewald, T. Thirolf, and A. Koziolek, "Lissa: Toward generic traceability link recovery through retrieval-augmented generation," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1396–1408.

[29] D. Fuchß, S. Corallo, J. Keim, J. Speit, and A. Koziolek, "Establishing a benchmark dataset for traceability link recovery between software architecture documentation and models," in *Software Architecture. ECSA 2022 Tracks and Workshops: Prague, Czech Republic, September 19–23, 2022, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 455–464. [Online]. Available: https://doi.org/10.1007/978-3-031-36889-9_30

[30] T. Hey, "Fine-grained traceability link recovery (FTLR)," 2023, zenodo. [Online]. Available: https://doi.org/10.5281/zenodo.8367343.