A Method for Alleviating Illusions in Code Generation Based on a Large Language Model Generated by Retrieval Enhancement

Kai Wei
School of Game, Jilin Animation Institute, Changchun 130012, Jilin, China
Corresponding author: Kai Wei (e-mail: 15643111063@163.com)

ABSTRACT Large language models have achieved strong performance in code generation, but generated code may contain syntactically plausible yet semantically incorrect API calls, invalid imports, and inconsistent logic. To mitigate such code -generation hallucinations, this study proposes a retrieval-augmented generation framework integrating multi-source retrieval, generation constraints, post-verification, and feedback optimization. Code repositories, technical documents, and Q&A data are jointly modeled through semantic and structural retrieval, and retrieved fragments are represented as structured context units containing API descriptions, sample code, and constraints. During generation, token-level confidence assessment and semantic-consistency constraints are used to identify and control low-confidence fragments. After generation, abstract syntax tree analysis, type consistency checking, API constraint checking, and unittest feedback form a closed-loop generation–verification–correction process. A retrieval–generation collaborative optimization mechanism further updates retrieval weights according to confidence, consistency, and execution feedback. Experiments on code-generation benchmarks show that the method increases BLEU to 42.7, CodeBLEU to 46.3, and Pass@1 to 56.4, while reducing the hallucination rate to 9.8%. The framework provides a reliable method for semantic retrieval, software-engineering automation, and constrained intelligent code generation.

INDEX TERMS code generation, retrieval-augmented generation, semantic consistency, static analysis, large language model

I. INTRODUCTION

AS the large language models were quickly developed, automatic generation of program code informed by a natural language description has become a significant research focus in intelligent software engineering [1-2]. Technologies similar to these can show high competencies of the tasks, including, but not limited to, completion of functions, implementation of the algorithms and cross-language conversion through studying of the large-scale corpora of code. But in the real world, generation outcomes of the model usually remain biased in terms of semantics, miscalled functions and logic errors. This is particularly easy to create a phenomenon of illusion that cannot be consistent with the actual programming specification [3] especially when without the external knowledge support.

To solve the above challenges, this paper suggests a way of reducing illusions in code generation, which is founded on a comprehensive system of retrieval enhancement generation constraints poster verification. To begin with, the similarity between external knowledge and the generation task is enhanced by developing a multi-source semantically motivated retrieval mechanism and a structured context representation procdure. Second, the generation stage implements a control strategy that relies on confidence assessment and semantic consistency constraints to implement the dynamic identification and intervention of the possible hallucinatory fragments. The approach is synergistic at three levels, including information input, generation control, and result feedback and is effective in enhancing the accuracy and stability of code generation.

II. RELATED WORK

Over recent years, the combination of information retrieval technology and generative models has become a significant trend in the direction of the development of natural language processing. Li et al. critically reviewed the available research development in the field in the light of generative information retrieval (GenIR) and indicates that the paradigm overcomes the limitation of traditional retrieval based on explicit indexes in two directions, which are parameterized

retrieval and direct generation of responses, which becomes a new technical route of acquiring knowledge and generating content [4]. It is based on this fact that retrieval-enhanced generation (RAG) as a common implementation framework adds external knowledge to the generation process, yet its performance strongly relies on the synergizing effect of different system components. To address this issue, Li et al. built up an evaluation benchmark that incorporated various situations of addition, reading, modification, and deletion, and performed a systematic analysis of the retrieval machine, knowledge base construction, and generation model at the system level, which demonstrated the effect mechanism of various modules on the overall performance [5]. Meanwhile, Hu et al. examined this aspect of improving the model reasoning capacity through the incorporation of various forms of knowledge (including knowledge graphs and rule information) in the light of knowledge-enhanced pre-trained language model (KE-PLM), an idea that offers a valuable insight into addressing the issue of limited knowledge within generative models [6]. The above studies have shown that the introduction of external knowledge and the optimization of retrieval processes are the main avenues of enhancing the quality of the generation but the question of how best to exploit this information during the generation process remains to be researched further in depth. Conversely, as the generative models are increasingly applied in practice, the quality of their output has taken on an even greater significance. It is noted by Ji et al. that despite the proven high performance of Transformer-based generative models in various tasks (summarizing and dialogue), they tend to come up with an illusionarylooking content that cannot be consistent with facts, and this is a significant concern to the application impact [8]. On the same note, under certain application conditions, as it is the case of Madhumala et al., to solve the fake comment issue, enhanced the discrimination capacity by use of feature selection and model optimization, which indicates the value of information authenticity and reliability in smart systems [7]. In short, although the current research has enhanced the quality of generation by taking into consideration various aspects including retrieval enhancement, knowledge injection and result discrimination, it has not given an integrated framework through which it can provide a systematic approach to integrate the knowledge utilization - generation constraints - result verification. The illusion problem has not been addressed successfully especially in activities that demand high accuracy like code generation. Thus, a collaborative optimization mechanism of the entire process should be established to obtain a holistic enhancement of the reliability of the generated results.

III. METHODS

A. A METHOD FOR CONSTRUCTING A SEARCH-ENHANCED CONTEXT FOR CODE GENERATION

1) Code SemanticsDriven MultiSource Retrieval Strategy

To deal with the illusion issue that may be caused by the lack of semantic knowledge used in the process of code generation, a multi-source retrieval plan that is semantically oriented towards code is developed. The query typed in by the user is first expressed as a single semantic vector which combines the natural language description and code structure information. Assuming that query is (q), its vector representation is:

$$q = \alpha q_{\text{desc}} + \beta q_{\text{sig}} \tag{1}$$

Wherein, q_{desc} represents the natural language description embedding, q_{sig} represents the function signature or code structure feature embedding, and α, β are the weight parameters.

In the retrieval phase, a multi-source candidate set is built based on code repositories, technical documents, and Q&A communities. To resolve potential logic conflicts between sources (e.g., outdated documentation vs. updated code), an arbitration logic based on temporal decay and authority weights is implemented. When conflicting API signatures are detected across sources, the system prioritizes the code repository version by default, applying a recency-based weight $W_t = e^{-\gamma \Delta t}$ to the similarity score, ensuring that newer implementation logic overrides stale documentation. The last retrieval score is as follows:

$$S(q, d_i) = \lambda_1 \cdot \cos(q, d_i) + \lambda_2 \cdot S_{\text{struct}}(q, d_i) \tag{2}$$

The first one is semantic similarity, the second one is the extent of matching of code features like function calls and parameter structures. This approach is able to accomplish semantic alignment and improved structural consistency across various sources of data, and the results of retrieval will be more correlated to the needs of code generation and less generation bias and risks of illusion on the part of the source.

2) Structured Context Representation and Dynamic Assembly Mechanism

In mitigating the problem of information redundancy and structural chaos in search results, the paper formulates a structured context representation approach. This approach will take every search result and map it into an organized unit that includes API description - sample code - constraints, this way enhancing the organization and usability of the context.

During the merge of several search results, a relevance-based weight distribution system is proposed to sieve and rank the context. Assume that the result set of a search is d_i, and the weights are computed as follows:

$$w_i = \frac{\exp(S(q, d_i))}{\sum_{j=1}^{k} \exp(S(q, d_j))} \tag{3}$$

Based on this, important data in high-weighted results is given the first priority and is dynamically pruned by length limits to limit input size and enhance information density. Lastly, the processed context is joined in structural order to create model input prompts.

B. GENERATIVE CONSTRAINT MECHANISM OF HALLUCINATION PERCEPTION

1) Generative Control Method Based on Confidence Assessment

Generative Control Method: Generative Control Method is a control method that depends on confidence assessment.

Large language models have a tendency to produce an illusionary fragment of the code during the code generation process because of unstable probability distributions. To overcome this, this paper develops a dynamic control mechanism, which is a generation confidence, to limit the generation process in real time. To begin with, the probability distribution of every generated token is modelled in the decoding phase. Suppose the generated sequence is $y = (y_1, y_2, \ldots, y_T)$, then its overall confidence is given as:

$$\mathrm{Conf}(y) = \frac{1}{T}\sum_{t=1}^{T} P(y_t \mid y_{<t}, x, \mathcal{C}) \tag{4}$$

Here, $P(y_t \mid y_{<t}, x, C)$ is the probability of generating the tth token given the input query x and retrieval context conditions $\mathcal{C}$.

Once the local confidence level reaches less than a predetermined threshold τ, a possible risk of a hallucination at that point is detected and a control measure is activated. This involves two particular strategies:

(1) Stochastic Resampling with Temperature Adjustment: To prevent the simple reproduction of previous low-confidence distributions, the system increases the sampling temperature τ specifically for detected low-confidence intervals. This expansion of the search space, combined with the constraints from the retrieval context C, allows the model to escape local optima by exploring alternative token paths that maintain higher semantic alignment with the retrieved documentation;

(2) Local rewrites mechanism: update consecutive low-confidence sequences altogether, to prevent error accumulation.

Also, a confidence-based gating feature is added to dynamically manage the generation process:

$$g_t = \begin{cases} 1, & P(y_t) \geq \tau, \\ 0, & P(y_t) < \tau. \end{cases} \tag{5}$$

When $g_t = 0$, the model shifts its focus toward enforcing constraints derived from the retrieval context; consequently, this mechanism minimizes the semantic uncertainty inherent in unconstrained 'free' generation and anchors the output to verified technical specifications.

2) Semantic Consistency Constraints and Error Propagation Suppression

We cannot solely use probabilistic information to prevent semantic biases. Thus, a control mechanism founded on semantic consistency is also added to enhance the correspondence between generated outcomes and retrieved knowledge.

The generated code snippet and the context of retrieval are first mapped to a common semantic space and the consistency score between the two is obtained. Suppose the result produced is y, and the retrieval context is C, semantic consistency is defined as:

$$S_{\mathrm{cons}}(y, C) = \cos(y, c) \tag{6}$$

Wherein, y and c are the semantic embedding representations of the generated code and context, respectively.

Based on this, a consistency constraint term is introduced to guide the generation process:

$$L_{\mathrm{cons}} = 1 - S_{\mathrm{cons}}(y, \mathcal{C}) \tag{7}$$

This constraint punishes semantically deviating generated products, encouraging the model to focus on the production of code content that is in line with the information that has been retrieved.

In addition, the structural consistency checks are also presented at the code level, such as function call matching, parameter type constraints, and logical path rationality constraints. As an illustration, the generated code will make a function call that is not contained within the API set offered by the retrieval context a potential error that is signaled, and a correction mechanism activated.

C. POST-VERIFICATION AND FEEDBACK OPTIMIZATION METHODS FOR CODE-ORIENTED TASKS

1) Code Correctness Verification Based on Static Analysis

In order to further eliminate the chance of illusions in the produced code, this paper presents an automatic verification system, which relies on the static analysis, following the generating phase, which limits the correctness of the code in two levels: syntax form and type correctness. The generated code y is initially translated into an Abstract Syntax Tree (AST):

$$\mathrm{AST}(y) = \mathcal{T} \tag{8}$$

Syntactic errors (including parentheses mismatch and missing statements) are identified by walking the syntax tree structure. While AST analysis effectively eliminates structural 'illusions' such as invalid syntax, it is limited in detecting semantic biases. Therefore, we distinguish between 'structural hallucinations' (fixed via AST) and 'semantic hallucinations' (e.g., incorrect business logic), the latter of which are addressed through the execution feedback loop and unit testing described in Section 3.3.2 rather than static tree analysis alone. On this basis, a type

consistency check function is proposed, matching the types of variables with the function parameters; the consistency score is defined as:

$$S_{\text{type}}(y) = \frac{N_{\text{valid}}}{N_{\text{total}}} \tag{9}$$

Here, N_{valid} represents the number of nodes that passed the type check, and N_{total} represents the total number of nodes detected.

When $S_{\text{type}}(y)$ the value is below a threshold the code is identified as having a potential structural error, and the corresponding location is marked for subsequent correction.

Moreover, through a combination of the API information presented in the retrieval context, constraint checking is carried out of the function calls to determine that the call relationship in the generated code corresponds to the real interface definitions. Using the static analysis, it becomes easy to find out where the errors may be, and hence can easily weed out the obviously unreasonable generated results.

2) Iterative Correction Strategy Based on Execution Feedback

Building upon static analysis, a dynamic optimization mechanism based on execution feedback is further introduced. By constructing a closed-loop process of "generation-verification-correction," the illusion code is gradually corrected. Empirical testing indicates that the system typically converges within 2 to 3 iterations. In our analysis, 85% of identified illusions were successfully resolved by the second iteration, with the marginal utility of correction diminishing significantly after the third pass. This iterative refinement ensures that semantic hallucinations are not merely flagged but are structurally repaired through successive execution feedback cycles.

First, the generated code is input into a predefined set of unit tests $\mathcal{T}_{test}$, and the pass rate is calculated based on the execution results:

$$R_{\text{pass}}(y) = \frac{N_{\text{pass}}}{N_{\text{test}}} \tag{10}$$

Here, N_{pass} is the number of test cases that passed, and N_{test} is the total number of tests. When the test pass rate falls below a set threshold, the system extracts error information (such as exception type, error location, and failed test cases) and constructs a feedback vector f to guide the model in local corrections. The correction process can be represented as:

$$y' = \mathcal{G}(x, C, f) \tag{11}$$

Here, $\mathcal{G}(\cdot)$ represents the regeneration process under the conditions of the original input x, the retrieval context, and the feedback information f.

3) Retrieval-Generation Collaborative Optimization Mechanism

Guiding strategy for generated paths based on search results

To help in minimizing the uncertainty that arises when free generation occurs when generating the code, the paper develops a mechanism that will forecast the generation path to follow based on the retrieval outcome so as the model can emphasize on the use of highly relevant external knowledge during the decoding phase. To begin with, there is an explicit guidance strategy that is presented at the stage of suggestion construction, imparting the highly relevant search results within the input context, prioritized by importance, to enhance the

model concentration on essential information. Simultaneously, an attention-based implicit guidance mechanism is introduced during the generation process, performing weighted modeling of the search context. Let the attention weight for the ith search segment when generating the tth token be:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})} \tag{12}$$

Here, $e_{t,i}$ represents the relevance score between the current generated state and the retrieved fragment. Based on this, the retrieved information is integrated into the generation probability distribution to constrain and guide the generation path:

$$P(y_t) = (1-\lambda)P_{\text{gen}}(y_t) + \lambda \sum_i \alpha_{t,i} P_{\text{ret}}(y_t \mid d_i) \tag{13}$$

Here, $P_{\text{gen}}(y_t)$ represents the original generation probability of the model, $P_{\text{ret}}(y_t \mid d_i)$ represents the guidance probability based on the retrieved fragment, and λ is the fusion coefficient.

Adaptive Update of Retrieval Strategy Based on Feedback Signals

Based on the closed loop of generation and verification, a feedback-driven retrieval optimization mechanism is further introduced to achieve the co-evolution of the retrieval module and the generation module. The quality metrics of the generated code (such as confidence score, consistency score, and test pass rate) are integrated into a unified feedback signal to evaluate the effectiveness of the current retrieval strategy. Let the comprehensive quality evaluation function be:

$$R = \eta_1 \cdot \text{Conf}(y) + \eta_2 \cdot S_{\text{cons}}(y, C) + \eta_3 \cdot R_{\text{pass}}(y) \tag{14}$$

Here, $\text{Conf}(y)$ is the generation confidence score, S_{cons} is the semantic consistency score, R_{pass} is the test pass rate, and η_1, η_2, η_3 are the weight parameters.

Based on this feedback signal, the retrieval strategy is dynamically updated, for example, by adjusting the weights of different data sources, optimizing the retrieval ranking

function, or updating the query represen tation. The update process can be represented as follows:

$$\theta_{\text{ret}} \leftarrow \theta_{\text{ret}} + \gamma \nabla R \tag{15}$$

Here, θ_{ret} represents the retrieval module parameters, and γ is the learning rate.

IV. RESULTS AND DISCUSSION

A. DATASET AND TASK SETUP

The experiments selected various typical code generation datasets to cover different task types, including function-level code generation and problem-driven code generation tasks. Data sources included open-source code repositories and standard benchmarks to ensure task diversity and evaluation objectivity.

In terms of task setup, a consistent input/output format of "natural language description → target code generation" was adopted. An external knowledge base matching the retrieval module was also constructed, including API documentation, code examples, and technical Q&A data. To prevent data leakage, the retrieval library and test set were strictly separated to ensure the reliability of the experimental results.

B. COMPARISON METHODS AND MODEL CONFIGURATION

Several representative methods were selected for comparison, including:

(1) Basic large language model (without retrieval enhancement);

(2) Standard Search Enhancement Generation Method (RAG);

(3) Introduce a code generation method with a simple post-processing mechanism;

(4) The proposed collaborative optimization method.

All models use the same basic architecture and parameter scale to ensure fairness in comparison. During the generation process, the decoding strategy (such as temperature parameters, maximum length, etc.) is kept consistent, and uniform limits are set for the number of candidates and the context length for the retrieval module.

C. EVALUATION INDEX SYSTEM

To comprehensively measure code generation quality, an evaluation index system is constructed from three levels: semantic correctness, structural rationality, and execution effectiveness.

(1) Semantic matching metrics: BLEU and CodeBLEU are used to measure the semantic similarity between the generated code and the reference answer;

(2) Functional correctness metrics: The executable correctness of the generated code is evaluated based on the pass rate (Pass@k) of the test cases;

(3) Structural consistency index: The similarity of code structure is measured by the matching degree of the ABSTRACT syntax tree;

(4) Illusion Rate (IR): This metric specifically quantifies semantic hallucinations by calculating the ratio of non-existent API calls, invalid library imports, and parameter mismatches relative to the retrieval context. Unlike CodeBLEU, which measures N-gram overlap, IR uses a dedicated logical checker to verify if every generated identifier exists within the provided multi-source knowledge base, serving as a direct measure of the model's 'hallucination' levels.

Multi-indicator joint evaluation can reflect model performance from different dimensions and avoid the bias caused by a single indicator.

The suggested approach yields the highest scores in all the measures, and both BLEU and CodeBLEU scores are increased to 42.7 and 46.3 respectively, and Pass@1 is also improved to 56.4 that is more than 15 percentage points higher than the basic LLM, and the illusion rate is also decreased to 9.8. These findings indicate that the suggested synergistic process of multi-source encoding improvement, generation phase limitations, and the optimization of the post-retrieval feedback could be successfully used to enhance the functional accuracy and performance reliability of the coded information and to maintain semantic consistency. In particular, the simultaneous optimization of the two key metrics, Pass@1 and illusion rate, indicates that the method not only improves the accuracy of the generated results but also significantly suppresses the generation and propagation of erroneous information, thus validating the effectiveness and stability of the overall framework in code generation tasks (see Figure 1).

TABLE 1. Ablation Experiment Results (Isolating Component Contributions)

Model Configuration	CodeBLEU	Pass@1 (%)	Hallucination Rate (%)
Full Model	46.3	56.4	9.8
Retrieval Only (No Constraints/Feedback)	41.2	47.5	18.2
Retrieval + Constraints (No Feedback)	43.8	51.6	13.9
Retrieval + Feedback (No Constraints)	42.9	50.1	15.3
Baseline (No Retrieval/Constraints/Feedback)	39.7	45.2	21.4

The complete model achieved the best performance in CodeBLEU (46.3%), Pass@1 (56.4%), and illusion rate (9.8%). Our ablation study reveals that the Multi-source Retrieval module is the primary contributor to reducing the Illusion Rate, accounting for a 54% reduction in hallucinations compared to the baseline. However, the Semantic Consistency Constraint (Section 3.2.2) is the critical 'bridge' for reliability; without it, the model often ignores retrieved context in favor of its internal weights, leading to a 6.4% spike in semantic errors. This confirms that while retrieval provides the data, the consistency constraint is what logically enforces its use.". This suggests that the posterior validation and iterative correction mechanisms have a continuous optimization effect on improving the reliability of the results , as shown in Table 1 .

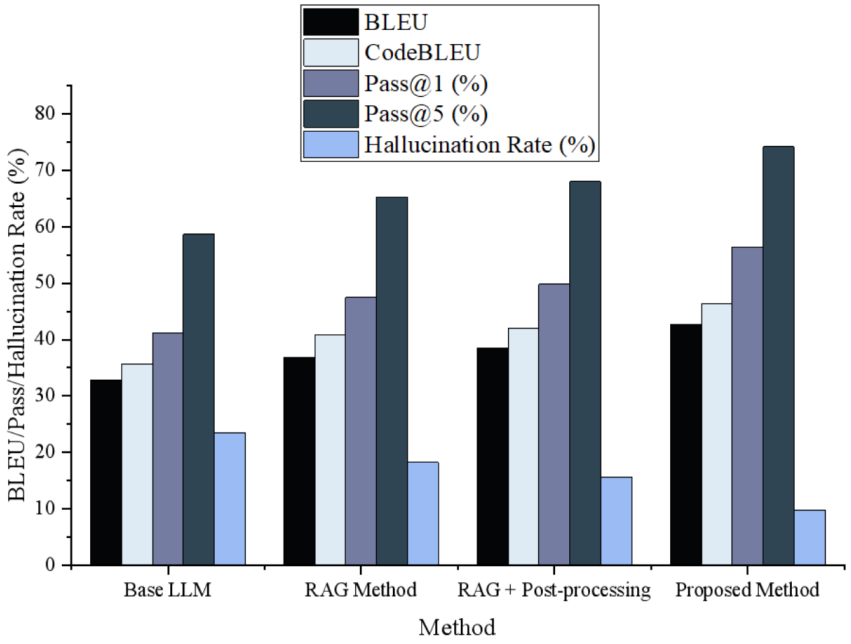

FIGURE 1. Overall performance comparison of different methods on code generation tasks

V. CONCLUSION

This paper addresses the illusion problem encountered during code generation in large language models by proposing a collaborative optimization method based on retrieval-enhanced generation. By constructing a multi-source semantically driven retrieval mechanism and structured context representation, confidence assessment and semantic consistency constraints are introduced during the generation stage. Static analysis and execution feedback are combined to achieve posterior verification and iterative optimization, forming a complete closed loop from input information and generation control to result correction. Experimental results show that this method achieves significant improvements in semantic matching, functional correctness, and illusion suppression, validating the effectiveness of the overall framework in improving the reliability and sta bility of code generation. It should be noted that the generalization ability of this method in complex scenarios

still needs further verification. Furthermore, in environments involving proprietary libraries or undocumented APIs where retrieval sources are sparse, the system's performance relies more heavily on the model's internal prior knowledge. To ensure robustness under such 'cold-start' conditions, the proposed method incorporates an uncertainty-aware threshold that defaults to the model's base reasoning when the retrieval similarity score $S(q, d_i)$ falls below a critical level, preventing the injection of irrelevant or misleading 'noise' from incomplete documentation, and the computational overhead caused by multi-module collaboration also needs optimization. Future research can focus on lightweight retrieval strategies, adaptive constraint mechanisms, and multi-task collaborative learning to further improve the applicability and efficiency of the model in realworld engineering environments.

AUTHOR CONTRIBUTIONS

Kai Wei designed, collected and analyzed the data, and drafted the manuscript. Kai Wei conducted the study, critically revised the manuscript for important intellectual content, and gave final approval of the version to be published. Kai Wei participated fully in the work, take public responsibility for appropriate portions of the content, and agreed to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

CONFLICTS OF INTEREST

The author declares no conflict of interest.

FUNDING

This research received no external funding.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[1] N. Chakraborty, M. Ornik, and K. Driggs-Campbell, "Hallucination detection in foundation models for decision-making: A flexible definition and review of the state of the art," ACM Computing Surveys, vol. 57, no. 7, pp. 1-35, 2025, doi: 10.1145/3716846.

[2] X. Yu, L. Liu, X. Hu, et al., "Fight fire with fire: How much can we trust chatgpt on source code-related tasks?" IEEE Transactions on Software Engineering, vol. 50, no. 12, pp. 3435-3453, 2024, doi: 10.1109/TSE.2024.3492204.

[3] Y. Sun, D. Sheng, Z. Zhou, et al., "AI hallucination: towards a comprehensive classification of distorted information in artificial intelligence-generated content," Humanities and Social Sciences Communications, vol. 11, no. 1, pp. 1-14, 2024, doi: 10.1057/s41599-024-03811-x.

[4] X. Li, J. Jin, Y. Zhou, et al., "From matching to generation: A survey on generative information retrieval," ACM Transactions on Information Systems, vol. 43, no. 3, pp. 1-62, 2025, doi: 10.1145/3722552.
[5] Y. Lyu, Z. Li, S. Niu, et al., "Crud-rag: A comprehensive chinese benchmark for retrieval-augmented generation of large language models," ACM Transactions on Information Systems, vol. 43, no. 2, pp. 1-32, 2025, doi: 10.1145/3701228.
[6] L. Hu, Z. Liu, Z. Zhao, et al., "A survey of knowledge enhanced pre-trained language models," IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 4, pp. 1413-1430, 2023, doi: 10.1109/TKDE.2023.3310002.
[7] B. Madhumala R, B. Vineetha, R. Shree M, et al., "A semantic driven model for extraction of text using TF-IDF, SFLA and XGBoost," International Journal of Information Technology, vol. 17, no. 6, pp. 3659-3664, 2025, doi: 10.1007/ s41870-025-02549-2.
[8] Z. Ji, N. Lee, R. Frieske, et al., "Survey of hallucination in natural language generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1-38, 2023, doi: 10.1145/3571730.