GGC: Selective Query Correction for Reliable Text-to-SPARQL Generation

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated strong capabilities in structured query generation, making them a natural choice for Text-to-SPARQL, which translates natural language questions into executable SPARQL queries over knowledge graphs. However, their initial outputs remain unreliable: generated queries may be executable yet semantically misaligned with input questions, leading to incorrect retrieval. To address this issue, we propose Generator–Gate–Corrector (GGC), a framework for reliable LLM-based Text-to-SPARQL generation. GGC first uses a Generator to produce an initial query, then applies a Gate to predict whether correction is needed, and finally invokes a Corrector only for selected high-risk queries. This selective correction mechanism avoids unnecessary modifications and reduces the risk of degrading originally correct queries. Experiments on MCQA show that GGC improves query-level accuracy from 90.23% to 98.33% while reducing inference overhead by 45% compared with correcting all generated queries. Ablation studies show that the Gate is robust across thresholds and that Corrector training data composition affects correction effectiveness and stability. Overall, the results demonstrate that selective correction enhances the accuracy, reliability, and efficiency of LLM-based text-to-SPARQL generation.

1 Introduction

Knowledge graph question answering (KGQA) aims to answer natural language questions (NLQs) by grounding them in the structured facts of knowledge graphs (KGs). Existing KGQA methods are typically categorized into semantic parsing (SP)-based methods and information retrieval (IR)-based methods (Lan et al., 2023). The former generates logical forms or query structures, while the latter obtains answers through entity, relation, or path retrieval over the graph.

In the context of RDF-based (Kellogg et al., 2026) KGs, Text-to-SPARQL is an important approach to KGQA. RDF-based KGs represent knowledge as subject-predicate-object triples, and SPARQL (Harris and Seaborne, 2013) is the standard query language for RDF data. Therefore, Text-to-SPARQL aims to translate NLQs into executable and semantically consistent SPARQL queries. Compared with direct natural language answer generation, this approach retrieves answers through explicit query execution over an external KG, making the intermediate query and retrieved graph facts inspectable and facilitating more verifiable and interpretable results (Pan et al., 2024).

In recent years, large language models (LLMs) have demonstrated strong capabilities in natural language understanding, generation, and code-related tasks (Brown et al., 2020; Chen et al., 2021). These capabilities make them a natural choice for Text-to-SPARQL, where models must understand user questions and generate semantically consistent and executable structured queries.

However, Text-to-SPARQL is not merely a text generation task. Generated queries must be executable while faithfully encoding the entities, relations, constraints, variable bindings, and reasoning paths expressed in the NLQ (Banerjee et al., 2022). Since executability alone does not guarantee semantic correctness, LLM-based Text-to-SPARQL requires mechanisms to verify consistency with the original question, especially as LLMs may produce plausible but unfaithful outputs (Xu et al., 2023).

Our experiments further show that this problem is particularly prominent in Text-to-SPARQL. While the fine-tuned Generator achieves high query-level accuracy, many of its remaining errors are executable yet semantically flawed. We define Generator errors as initial SPARQL queries produced by the Generator that are judged incorrect against the ground-truth queries. On the Movie Complex Question Answering (MCQA) (Hoang

et al., 2024) dataset, semantic errors account for 77.70% of Generator errors, while syntactic errors account for only 21.07%. This indicates that relying solely on execution failures to trigger corrections will miss many semantic errors.

The key insight of this paper is that in LLM-based Text-to-SPARQL, the issue is not only how to generate queries, but also when to correct them. Correcting all generated results may fix incorrect queries, but it also incurs higher inference overhead and may degrade originally correct queries. In contrast, selective correction concentrates computation on high-risk samples, improving accuracy while reducing unnecessary rewriting. Experimental results show that the proposed Generator–Gate–Corrector (GGC) framework improves query-level accuracy from 90.23% to 98.33%, with only about 10% additional inference time over Generator-only inference. Compared with correcting all samples, selective correction achieves higher accuracy while reducing inference time by about 45%.

In summary, our contributions are as follows:

- We analyze LLM-based Text-to-SPARQL from a reliability perspective and show that executable but semantically flawed queries are a significant source of errors affecting system performance.
- We propose the GGC framework, which performs selective query correction through a detect-then-correct strategy and achieves a better balance between accuracy and inference efficiency.
- We conduct systematic experiments and ablation analyses on MCQA to examine the impact of the Generator, Gate, Corrector, KG execution feedback, and Corrector training data composition, providing empirical evidence for building more reliable Text-to-SPARQL systems. Additional results on SciQA (Auer et al., 2023), reported in Appendix D.2, further support these findings.

2 Related Work

2.1 Knowledge Graph Question Answering

Existing KGQA methods can generally be divided into SP-based methods and IR-based methods. SP-based methods convert natural language questions into logical forms, query graphs, or structured queries that can be executed on a knowledge graph to obtain answers (Lan et al., 2023). These methods provide relatively explicit reasoning processes, but their performance depends heavily on the quality of the generated intermediate structures. In contrast, IR-based methods retrieve relevant entities, relations, or subgraphs from topic entities and obtain answers through path search or neural reasoning. They are flexible for multi-hop reasoning but are often less interpretable than explicit query-based methods.

Query Graph Generation (QGG) (Lan and Jiang, 2020) and Neural State Machine-hybrid ($\mathrm{NSM_h}$) (He et al., 2021) are representative methods of these two directions respectively. QGG constructs query graphs with entities, relations, constraints, and answer variables, while $\mathrm{NSM_h}$ improves multi-hop reasoning by learning intermediate supervision signals. Unlike these methods, we focus on the reliability of LLM-based Text-to-SPARQL generation, especially how to identify and correct high-risk SPARQL queries after generation. Since QGG and $\mathrm{NSM_h}$ are important reference baselines on the MCQA dataset, they are included in the experiments for comparison.

2.2 Text-to-SPARQL and SPARQL Semantic Parsing

With the development of pre-trained language models, Text-to-SPARQL has shifted from rule-based, template-based, and task-specific semantic parsing methods toward generation-based approaches (Lan et al., 2023). Prior work has compared BART, T5, and pointer-generator models on LC-QuAD 1.0 and LC-QuAD 2.0, showing that pre-trained models provide strong baselines for SPARQL semantic parsing (Banerjee et al., 2022).

Recent studies have further applied LLMs to SPARQL generation (Xu et al., 2023; D'Abramo et al., 2025) over Wikidata (Vrandečić and Krötzsch, 2014). By translating NLQs into executable SPARQL queries, these methods retrieve answers from KGs through an explicit query execution process, improving verifiability compared with direct answer generation. More recently, FIRES-PARQL introduced a modular LLM-based framework for SPARQL generation over scholarly knowledge graphs (Pan et al., 2025), combining fine-tuning, optional retrieval-augmented context, and query correction. These studies demonstrate the potential of LLMs for SPARQL generation, but they pay less attention to the reliability of generated queries, especially executable but semanti-

cally flawed queries. In contrast, our work focuses on when post-generation correction should be triggered. The proposed Generator–Gate–Corrector framework uses a learned Gate to selectively route only high-risk queries to the Corrector, reducing unnecessary rewriting and inference overhead.

2.3 Constrained Generation and Error Correction in Structured Query Generation

Text-to-SPARQL and Text-to-SQL are both natural-language-to-structured-query generation tasks that require syntactically valid and semantically correct outputs. Therefore, constrained decoding and error correction methods in Text-to-SQL provide useful references for Text-to-SPARQL.

For constrained decoding, PICARD (Scholak et al., 2021) constrains autoregressive decoding through incremental parsing and rejects tokens that violate SQL syntax. For error correction, Text-to-SQL works (Chen et al., 2023; Shi et al., 2025) have shown that post-generation correction can improve semantic parsing accuracy, and that structured clause-level editing is often more suitable than token-level editing for query repair.

These studies show that improving structured query generation does not only rely on stronger initial generators, but can also benefit from constraint, validation, and correction modules.

2.4 Motivation for Selective Correction in LLM-based Text-to-SPARQL

Existing work has advanced KGQA through graph reasoning, semantic parsing, and structured query generation. However, LLM-based Text-to-SPARQL methods still face reliability issues, where executable queries may be semantically inconsistent with the original question. Inspired by recent post-correction studies in Text-to-SQL, we explore post-correction for LLM-based Text-to-SPARQL. Unlike prior work focusing mainly on query generation, our approach emphasizes post-generation error detection and selective correction to determine when and how corrections should be applied for improved semantic consistency.

3 Methodology

We propose a Generator–Gate–Corrector framework to improve the reliability of LLM-based Text-to-SPARQL generation. Instead of correcting all generated queries, this framework first determines whether the initial SPARQL query is likely to contain errors, and then triggers corrections only for high-risk queries. The overall goal is to improve query accuracy while reducing unnecessary correction and additional inference overhead.

3.1 Task Formulation

Given an NLQ, q, the objective of Text-to-SPARQL is to generate an executable SPARQL query, s. The answer set can be retrieved from the KG by executing the query. Compared with general text generation tasks, Text-to-SPARQL requires the generated queries to satisfy the following requirements: First, queries must conform to the syntax of SPARQL, and second, queries must be semantically consistent with original questions.

The initial query is generated by the Generator, denoted as $s^{(0)}$. If the query is determined to have potential errors, a corrected query, $s^{(c)}$, is generated by the Corrector. The final output query is denoted as $\hat{s}$. Therefore, we focus on how to generate $s^{(0)}$ and on how to determine whether it needs to be corrected.

3.2 Generator-Gate-Corrector Framework

We separate the Text-to-SPARQL process into three stages: initial generation, error detection, and selective correction. The overall process is as follows:

$$s^{(0)} = G(q)$$

$$g = Gate(q, s^{(0)})$$

$$\hat{s} = \begin{cases} s^{(0)}, & \text{if } g = 0 \\ C(q, s^{(0)}), & \text{if } g = 1 \end{cases}$$

where $G(\cdot)$ denotes the Generator, $C(\cdot)$ denotes the Corrector, and $g \in \{0, 1\}$ is the binary output of the Gate. The framework is shown in Figure 1.

Since not all initial SPARQL queries need to be corrected, triggering the Corrector on all queries would significantly increase inference overhead and could also corrupt queries that were initially correct. Therefore, the Gate in the framework acts as a query risk assessor, ensuring that the Corrector is triggered only on high-risk queries.

3.3 Generator

The objective of the Generator is to translate an NLQ, q, into an initial SPARQL query, $s^{(0)}$. We use a supervised fine-tuned LLM as the Generator. During training, the input is the NLQ, and the target output is the corresponding ground-truth SPARQL

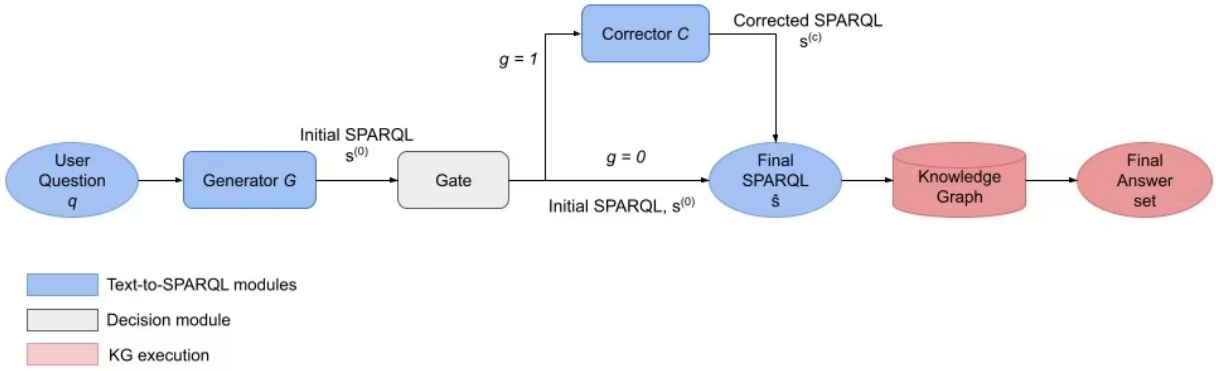

Figure 1: The proposed Generator-Gate-Corrector framework for Text-to-SPARQL generation.

query. During inference, the Generator generates an initial query $s^{(0)}$ based on the input question.

The Generator aims to produce an initial candidate query that is as accurate as possible. However, due to SPARQL's strict requirements for entities, relations, constraints, and variable bindings, the Generator's output may still contain syntactic or semantic errors. Therefore, the initial query is not directly considered the final result but is further passed to the Gate for reliability assessment.

3.4 Gate

The Gate is a binary classification module used to determine whether the initial SPARQL query needs correction. Its input consists of an NLQ and the initial query produced by the Generator:

$$x_{gate} = [q; s^{(0)}]$$

The Gate outputs the probability that correction is required, and the final decision is obtained based on the threshold τ:

$$g = \mathbb{I}\left[P\left(g = 1 \mid x_{gate}\right) \geq \tau\right]$$

where, $g = 1$ indicates that the Corrector is triggered, and $g = 0$ indicates that the initial SPARQL query is kept. The default threshold is $\tau = 0.5$.

Compared with methods that rely solely on execution feedback, the objective of the Gate is to identify queries that may contain syntactic or semantic errors, including executable queries that are semantically inconsistent with the NLQ. This is important because many incorrect SPARQL queries are executable but return answers that do not match the question. Therefore, the Gate needs to identify not only syntactic or execution-related errors, but also potential semantic errors.

3.5 Corrector

The Corrector aims to revise SPARQL queries identified as high-risk by the Gate. Unlike the Generator, the input to the Corrector contains an NLQ and a corresponding initial SPARQL query.

The objective of the Corrector is to fix potential errors in entities, relations, constraints, variable bindings, or query structure while preserving the correct parts of the initial query. During training, the Corrector takes an NLQ and an initial query produced by the Generator as input, and the corresponding ground-truth SPARQL query as the target output.

We further consider different training data compositions for the Corrector. Using only incorrect samples can provide a more direct correction signal, while adding a certain proportion of originally correct samples can help the Corrector learn to maintain stable output when the query is already correct. The impact of different training data compositions on correction effectiveness and stability will be examined in section 4.3.

3.6 Training and Inference

Both the Generator and the Corrector are trained using standard supervised fine-tuning, and training samples are constructed using an instruction-response format. For the Generator, the instruction includes a task description and an NLQ, while the response is a ground-truth SPARQL query. For the Corrector, the instruction includes a task description, an NLQ, and an initial SPARQL query, while the response is a ground-truth SPARQL query. During training, the language modeling loss is applied only to the response portion.

The Gate is trained as a separate binary classi-

fier. Its training samples consist of an NLQ, an initial SPARQL query, and a binary label. If the initial query is incorrect, the label is "Correction Required". If the initial query is correct, the label is "No Correction Required".

The inference procedure is summarized in Algorithm 1:

Algorithm 1 Generator-Gate-Corrector Inference

Require: Natural language question q
Ensure: Final SPARQL query $\hat{s}$

$s^{(0)} \leftarrow G(q)$
$g \leftarrow \mathrm{Gate}(q, s^{(0)})$
if $g = 1$ **then**
  $s^{(c)} \leftarrow C(q, s^{(0)})$
  $\hat{s} \leftarrow s^{(c)}$
else
  $\hat{s} \leftarrow s^{(0)}$
end if
return $\hat{s}$

Through this pipeline, the system selectively corrects high-risk queries, achieving a better balance between accuracy, reliability, and inference efficiency.

4 Experiments and Results

This section first introduces the experimental setup and compared systems, then presents the end-to-end framework results and ablation studies on key components, including the Generator, Gate and its threshold sensitivity, KG execution feedback, and Corrector training data composition.

4.1 Experimental Setup

Dataset The experiments were conducted on the MCQA dataset (Hoang et al., 2024). MCQA is a complex KGQA dataset for the movie domain constructed over iMKG, a KG based on Wikidata and MovieKG. Each sample contains an NLQ, question type, topic entity, answer, and corresponding ground-truth SPARQL query. This dataset is built on a movie-domain KG and is suitable for evaluating Text-to-SPARQL and KGQA methods.

The dataset includes training, validation, and test sets. The training set contains 119,409 samples, the validation set contains 15,785 samples, and the test set contains 31,570 samples. We use a portion of the training data to fine-tune the Generator, enabling it to learn the mapping from NLQs to SPARQL queries. Then, the trained Generator is used to generate initial queries on the remaining training data, and training samples for the Gate and Corrector are constructed accordingly. If the initial query generated by the Generator is incorrect, the corresponding sample is labeled "Correction Required". Otherwise, it is labeled "No Correction Required".

Setting	Description
Generator-only	No correction; output $s^{(0)}$.
Gen. + Corr.	Correct all initial queries.
Gen. + Gate + Corr.	Selectively correct initial queries.
$\mathrm{NSM_h}$	IR baseline reported in MCQA.
QGG	SP baseline reported in MCQA.

Table 1: Evaluated settings and baselines. Gen. denotes the Generator, Corr. denotes the Corrector.

Compared Settings Table 1 summarizes the compared settings in the experiments. Generator-only evaluates the initial generation capability, while Gen. + Corr. tests the effect of correcting all initial queries. Gen. + Gate + Corr. represents the proposed selective correction framework. We also include $\mathrm{NSM_h}$ and QGG as traditional KGQA reference baselines to assess end-task answer retrieval performance. Since these methods do not directly generate SPARQL queries or report query-level accuracy, their results are not strictly comparable to ours at the query level. Existing LLM-based Text-to-SPARQL methods are not included as main baselines because standardized implementations and evaluation settings remain limited.

Implementation Details Both the Generator and Corrector are supervised fine-tuned based on Llama-3.2-3B-Instruct (Grattafiori et al., 2024), employing LoRA (Hu et al., 2022) and 4-bit quantized loading (Dettmers et al., 2023) to reduce training costs. Both used an instruction-response format to construct training samples and applied the language modeling loss only in the response portion.

The Gate is modeled as a binary classifier. We compare the RoBERTa-based (Liu et al., 2019) Gate and the SBERT-based (Reimers and Gurevych, 2019; Wang et al., 2020) Gate, and use the RoBERTa-based Gate, which performed better, as the default setting.

Evaluation Metrics For the Generator, Corrector, and framework, we report both query-level and item-level metrics. Query-level accuracy measures the matching between the generated SPARQL and the ground-truth SPARQL, and is the primary evaluation metric. Item-level precision, recall, and F1

System	Query Acc. (%)	Item Prec. (%)	Item Rec. (%)	Item F1 (%)	Infer. Time (Total / Q.)
$\mathrm{NSM_h}$	–	59.20	70.06	59.44	–
QGG	–	11.65	25.13	12.32	–
Generator-only	90.23	37.96	85.76	52.63	∼30h / ∼3.4s
Gen. + Corr. (correct-all)	92.34	91.62	93.43	92.51	∼60h / ∼6.8s
Gen. + Gate + Corr. (selective)	**98.33**	**99.62**	**98.70**	**99.16**	∼33h / ∼3.7s

Table 2: Main results on MCQA. The proposed selective correction framework achieves the best query-level accuracy while requiring much less inference time than the correct-all baseline. Gen. denotes the Generator and Corr. denotes the Corrector. Infer.Time (Total/Q.) refers to both the total inference time on the full test set, and the average time per test question, respectively.

score measure the matching between the answer set obtained by the generated SPARQL and the ground-truth answer set, and are used for comparison with traditional KGQA baselines.

For the Gate, we report accuracy, precision, recall, F1 score, false positive rate (FPR), and trigger rate. "Correction Required" is considered the positive class, and the trigger rate represents the proportion of samples that the Gate determines need to proceed to the Corrector.

The detailed experimental setup and prompt templates are summarized in Appendices A and B.

4.2 End-to-End Framework Results

We evaluate the end-to-end GGC framework and compare it with traditional KGQA baselines on MCQA. The results are shown in Table 2.

The proposed framework achieves strong overall performance, reaching 98.33% query-level accuracy and 99.16% item-level F1. Compared with traditional KGQA baselines, GGC obtains better item-level performance, showing that explicit SPARQL generation combined with selective correction can improve final answer retrieval quality.

However, this comparison should be interpreted carefully because the baselines and our method optimize different objectives. Traditional KGQA baselines such as $\mathrm{NSM_h}$ and QGG are designed for answer retrieval, while our framework explicitly generates executable SPARQL queries. Since these baselines do not report query-level accuracy, they cannot be directly compared in terms of SPARQL generation quality. Therefore, we use them as reference baselines for MCQA answer retrieval, rather than as direct Text-to-SPARQL generation baselines.

Overall, the results show that the proposed GGC framework achieves competitive answer retrieval performance by additionally producing explicit SPARQL queries, improving accuracy, reliability, and inference efficiency.

4.3 Ablation Study on the Impact of each Component

To understand how each component contributes to the framework, we conduct ablation studies on them. We analyze the Generator-only performance, evaluate the Gate's ability to identify high-risk queries, and study key design choices including Gate threshold, KG execution feedback, and Corrector training data composition.

Generator-only Performance Generator-only achieves a query-level accuracy of 90.23%, which indicates that the fine-tuned LLM already has a considerable Text-to-SPARQL generation capability. However, the item-level precision is only 37.96%, significantly lower than the recall of 85.76%. These results in Table 2 indicate that although some generated queries may retrieve the correct answers, they also return a large number of irrelevant answers. In other words, Generator errors are not primarily non-executable queries, but are more likely to be semantic errors such as overly broad query scope, incorrect relation selection, missing constraints, or inaccurate variable binding.

To further analyze the error source of the Generator, we categorized the incorrect samples into syntax error, semantic error, and ground-truth SPARQL error. The results are shown in Table 3. As can be seen, semantic errors account for 77.70% of the Generator's errors, significantly higher than the 21.07% for syntax errors. This indicates that the main challenge of LLM-based Text-to-SPARQL is not merely generating executable queries, but rather generating queries that are semantically consistent with the NLQs. Therefore, relying solely on execution failure to trigger correction is insufficient, as this approach misses a large number of executable but semantically incorrect queries.

This result supports the necessity of introduc-

ing the Gate. The goal of the Gate is to determine whether a query can be executed, as well as to determine whether the initial query may have semantic inconsistencies, thereby deciding whether the Corrector should be triggered.

Error Type	Count	Percentage (%)
Syntax Error	650	21.07
Semantic Error	2,397	77.70
Gold SPARQL Error	38	1.23
Total	3,085	–

Table 3: Error distribution of the Generator.

Gate Performance To evaluate whether the Gate can identify which queries require correction, we compare the Gate with two backbones, and the results are shown in Table 4.

Both backbones achieved high classification performance, indicating that strong error signals were already present between the NLQ and the initial SPARQL query. The RoBERTa-based gate is better than the SBERT-based gate in recall and F1 score. RoBERTa's recall reached 92.11%, meaning it could cover most queries that truly needed correction. Meanwhile, its FPR was only 0.31%, indicating it rarely misclassified originally correct queries.

This is particularly important for the entire pipeline. If the Gate's recall is too low, the Corrector will not be triggered for many incorrect queries. If the FPR is too high, the Corrector may be frequently triggered, corrupting originally correct queries. Therefore, the Gate's role is not only to reduce inference overhead but also to perform risk screening between correct and incorrect queries. Due to the superior overall performance of the RoBERTa-based gate, we use RoBERTa as the default Gate backbone in subsequent experiments.

Effect of Selective Correction We further compare selective correction with Generator-only and Generator + Corrector, where the Corrector is applied to all generated queries. As shown in Table 2, Generator-only achieves 90.23% query-level accuracy with approximately 30 hours of inference time. Correcting all samples increases the inference time to approximately 60 hours, but only improves query-level accuracy to 92.34%.

In contrast, the complete GGC framework achieves 98.33% query-level accuracy with approximately 33 hours of inference time. This indicates that the Gate is essential not only for reducing computational cost, but also for avoiding unnecessary correction of originally correct queries.

Gate Threshold Sensitivity We analyze whether the Gate depends on careful threshold tuning. As shown in Table 5a, varying the threshold mainly introduces a small precision–recall trade-off: lower thresholds trigger more corrections and slightly improve recall, while higher thresholds reduce false triggers and slightly improve precision. However, the overall performance remains stable.

When the threshold changes from 0.25 to 0.75, the trigger rate only changes from 9.45% to 9.26%, and the F1 score remains around 94%. This suggests that the Gate learns stable question-query mismatch signals rather than relying on a carefully tuned threshold. We therefore use $\tau = 0.5$ as the default setting.

Effect of KG Execution Feedback Intuitively, feedback from KG execution can provide useful information, such as whether the query can be executed successfully or whether the returned result is abnormal. Therefore, we compare the with-feedback and without-feedback settings for both the Gate and the pipeline.

The results in Table 5b demonstrate that KG feedback improves both Gate performance and end-to-end pipeline performance. For example, pipeline query-level accuracy increases from 98.27% to 98.70%. However, the improvement is relatively limited, especially on the main query-level metric. In contrast, the without-feedback setting avoids an additional execution step before correction, making the pipeline simpler and more efficient.

Therefore, KG feedback is useful but not essential. The strong without-feedback results suggest that the NLQ-SPARQL pairs already contain sufficient signals for detecting semantic errors. Considering both performance and efficiency, the without-feedback setting is more practical as the default configuration.

Effect of Corrector Training Data Composition Finally, we analyze how adding correct samples affects Corrector training. We use W to denote initially wrong queries produced by the Generator and C to denote initially correct queries. $W1.0$ denotes training only on the full set of wrong samples, while $W1.0 + C0.2$, $W1.0 + C0.5$, and $W1.0 + C0.8$ additionally add correct samples whose sizes correspond to 20%, 50%, and 80% of all training samples, respectively. Thus, the training set size increases as more correct samples are added.

Gate Model	Acc. (%)	Prec. (%)	Rec. (%)	F1 (%)	FPR (%)	Trigger Rate (%)
RoBERTa	98.95	97.00	92.11	94.49	0.31	9.31
SBERT	98.32	96.45	86.03	90.94	0.34	8.74

Table 4: Comparison of Gate models.

Threshold	Acc. (%)	Prec. (%)	Rec. (%)	F1 (%)	FPR (%)	Trigger Rate (%)
0.25	98.89	95.98	92.57	94.24	0.42	9.45
0.50	98.95	97.00	92.11	94.49	0.31	9.31
0.75	98.93	97.13	91.79	94.38	0.29	9.26

(a) Effect of Gate threshold.

Setting	Gate Acc. (%)	Gate F1 (%)	Pipeline Query Acc. (%)	Pipeline Item F1 (%)
With feedback	99.36	96.66	98.70	99.09
Without feedback	98.95	94.49	98.27	97.72

(b) Effect of KG feedback.

Composition	Query Acc. (%)	Item Prec. (%)	Item Rec. (%)	Item F1 (%)
W1.0	98.27	97.09	98.37	97.72
W1.0 + C0.2	98.33	99.62	98.70	99.16
W1.0 + C0.5	98.63	87.66	98.85	92.92
W1.0 + C0.8	98.80	99.66	99.00	99.33

(c) Effect of Corrector training data composition.

Table 5: Ablation results of the framework.

As shown in Table 5c, adding correct samples does not weaken the Corrector. Query-level accuracy increases from 98.27% under $W1.0$ to 98.80% under $W1.0 + C0.8$, suggesting that correct samples help the model preserve original correct queries and reduce unnecessary rewriting. Since larger C ratios also increase training overhead, we use $W1.0 + C0.2$ as the default setting in the main experiments to balance performance and training efficiency.

Item-level metrics show some fluctuation. For example, $W1.0+C0.5$ achieves higher query-level accuracy than $W1.0 + C0.2$, but its item-level precision and F1 are lower, possibly due to outlier queries with large answer sets. Therefore, we treat query-level accuracy as the primary metric and item-level metrics as supplementary evidence.

Overall, this ablation suggests that Corrector training benefits from both repair and preservation signals: wrong samples teach correction, while correct samples discourage unnecessary rewriting.

Additional case studies, subset-level Gate-Corrector coordination analysis and additional dataset preliminary results are provided in Appendices C, D.1, and D.2.

5 Conclusion

In this paper, we presented a GGC framework for reliable LLM-based Text-to-SPARQL generation. The framework first generates an initial SPARQL query, uses a Gate to decide whether correction is needed, and triggers the Corrector only on high-risk queries, improving accuracy and reliability while reducing inference overhead.

Experiments on the MCQA dataset show that the main errors of the Generator are executable but semantically inconsistent queries. This suggests that execution failure alone is insufficient for triggering correction. The Gate identifies queries requiring correction with a low FPR, and the full framework substantially improves query-level accuracy over both Generator-only and correct-all settings. Compared with correcting all generated queries, selective correction achieves higher accuracy while reducing inference time by about 45%.

Overall, reliable Text-to-SPARQL requires both strong generation capability and effective detection of when to correct. The proposed GGC framework demonstrates the potential of addressing this issue through a post-hoc and selective correction strategy, and we hope it will encourage further research in this direction.

Limitations

While this paper proposes a Generator–Gate–Corrector framework for reliable LLM-based Text-to-SPARQL generation and demonstrates its effectiveness on MCQA, we acknowledge several limitations that warrant further exploration in future work:

- The main experiments are conducted on MCQA, a movie-domain dataset with label-based SPARQL queries. Although the results demonstrate the effectiveness of selective correction in this setting, evaluation beyond MCQA remains limited in scope. We provide an additional preliminary evaluation on another Text-to-SPARQL dataset in Appendix D.2, but more comprehensive experiments across diverse domains, knowledge graphs, SPARQL formats, and model backbones are needed to further assess the generalizability of the GGC framework. In particular, extending the framework to ID-based SPARQL settings may require additional entity linking, schema alignment, and relation grounding mechanisms.

- Dependence on the Generator's error distribution. The Gate and Corrector are trained using the outputs of a specific Generator. Therefore, their performance may depend on the error patterns of that Generator. When a different or stronger base model is used, the Gate and Corrector may need to be retrained or adapted.

- Additional offline computation. Although the proposed framework reduces unnecessary correction during inference, constructing training data for the Gate and Corrector still requires generating initial SPARQL queries on a large number of training samples. This introduces additional offline computational cost, especially when scaling to larger datasets or stronger base models.

Ethical Considerations

Our proposed framework focuses on improving the reliability of LLM-based Text-to-SPARQL generation over knowledge graphs. The experiments are conducted on an existing KGQA dataset and do not involve collecting private user data or annotating sensitive personal information. However, if such systems are deployed in real-world applications, incorrectly generated SPARQL queries may still lead to misleading or incomplete answers, especially when the underlying knowledge graph contains outdated, biased, or incomplete facts. Therefore, practical deployment should include appropriate validation, uncertainty indication, and human oversight in high-stakes scenarios. In addition, extending the framework to open-domain or ID-based knowledge graphs may introduce risks from entity linking errors and biases encoded in the knowledge graph, which should be carefully examined in future work.

References

S. Auer, Dante Augusto Couto Barone, Cassiano Bartz, E. Cortes, Mohamad Yaser Jaradeh, Oliver Karras, Manolis Koubarakis, Dmitry I. Mouromtsev, Dmitrii Pliukhin, Daniil Radyush, Ivan Shilin, Markus Stocker, and Eleni Tsalapati. 2023. The sciqa scientific question answering benchmark for scholarly knowledge. *Scientific Reports*, 13.

Debayan Banerjee, Pranav Ajit Nair, Jivat Neet Kaur, Ricardo Usbeck, and Chris Biemann. 2022. Modern baselines for sparql semantic parsing. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2260–2265, New York, NY, USA. Association for Computing Machinery.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Ziru Chen, Shijie Chen, Michael White, Raymond Mooney, Ali Payani, Jayanth Srinivasa, Yu Su, and Huan Sun. 2023. Text-to-SQL error correction with language models of code. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1359–1372, Toronto, Canada. Association for Computational Linguistics.

Jacopo D'Abramo, Andrea Zugarini, and Paolo Torroni. 2025. Investigating large language models for text-to-SPARQL generation. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pages 66–80, Albuquerque, New Mexico, USA. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

FacebookAI. 2024. FacebookAI/roberta-base model card. Hugging Face model card.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. W3C Recommendation. Accessed: 2026-05-23.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 553–561, New York, NY, USA. Association for Computing Machinery.

Lily Hoang, Fiona Liausvia, Yan Liu, and Thanh-Son Nguyen. 2024. Semi-automated construction of complex knowledge base question answering dataset using large language model. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 230–248. Springer.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Gregg Kellogg, Olaf Hartig, Pierre-Antoine Champin, and Andy Seaborne. 2026. RDF 1.2 Concepts and Abstract Data Model. W3C Candidate Recommendation Snapshot. Accessed: 2026-05-23.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Complex knowledge base question answering: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11196–11215.

Yunshi Lan and Jing Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 969–974, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Xueli Pan, Victor de Boer, and Jacco van Ossenbruggen. 2025. Firesparql: A llm-based framework for sparql query generation over scholarly knowledge graphs.

In *Proceedings of the 17th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR*, pages 123–134. INSTICC, SciTePress.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sentence Transformers. 2025. sentence-transformers/all-MiniLM-L6-v2 model card. Hugging Face model card.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2).

Unsloth AI. 2024. unsloth/Llama-3.2-3B-Instruct. Hugging Face model card.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Silei Xu, Shicheng Liu, Theo Culhane, Elizaveta Pertseva, Meng-Hsi Wu, Sina Semnani, and Monica Lam. 2023. Fine-tuned LLMs know more, hallucinate less with few-shot sequence-to-sequence semantic parsing over Wikidata. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5778–5791, Singapore. Association for Computational Linguistics.

A Detailed Experimental Setup

A.1 Models

The Generator and the Corrector are fine-tuned based on “unsloth/Llama-3.2-3B-Instruct" (Unsloth AI, 2024). The model has the following parameter counts:

- Total Parameters: 3,237,063,680
- Trainable Parameters: 24,313,586

The backbones of the Gate are implemented based on "FacebookAI/roberta-base" (FacebookAI, 2024) and "sentence-transformers/all-MiniLM-L6-v2" (Sentence Transformers, 2025). These models have the following parameter counts:

- RoBERTa-base: $\sim$ 125,000,000
- all-MiniLM-L6-v2: 22,713,216

A.2 Hyper-parameters

Generator and Corrector The hyper-parameters of the Generator and the Corrector are as follows:

- Maximum sequence length: 2048
- Batch size: 2
- Gradient accumulation steps: 4
- Number of epochs: 1
- Learning rate: 2×10^{-4}
- LoRA rank r: 16
- LoRA alpha: 16
- LoRA dropout: 0
- Optimizer: AdamW_8bit (Loshchilov and Hutter, 2019)
- Weight Decay: 0.01
- Random seed: 3407

Gate The hyper-parameters of the Gate are as follows:

- Maximum sequence length: 256
- Batch size: 8
- Number of epochs: 3
- Learning rate: 2×10^{-5}
- Random seed: 42

A.3 Hardware and Software Environment

All experiments were conducted on a server with the following configurations:

- GPU: NVIDIA GeForce RTX 3090 with 24 GB VRAM.
- CPU: Intel Xeon W-2295
- Memory: 256 GB
- Operating System: Linux Ubuntu 22.04
- Deep learning framework: PyTorch 2.8.0 with CUDA 12.8.
- LLM fine-tuning libraries: Transformers 4.55.4, Datasets 3.6.0, Unsloth 2025.9.7
- Machine learning library: scikit-learn 1.8.0.
- Knowledge graph and SPARQL tools: RDFLib 7.2.1 and LangChain 0.3.27.
- Knowledge graph engine: GraphDB with a local SPARQL endpoint.

A.4 Training Time

Table 6 summarizes the training time of the Generator, the Gates with two backbones, and the Correctors with different training data compositions.

Setting	Training Samples	Epochs	Time
Generator	12,450	1	$\sim$1h
Gate-RoBERTa	88,506	3	$\sim$1h
Gate-SBERT	88,506	3	$\sim$1h
Corr. W1.0	11,534	1	$\sim$1h
Corr. W1.0 + C0.2	14,418	1	$\sim$1.5h
Corr. W1.0 + C0.5	23,068	1	$\sim$2h
Corr. W1.0 + C0.8	57,670	1	$\sim$5h

Table 6: Training time of different modules and Corrector data compositions. W denotes initially wrong queries generated by the Generator, and C denotes initially correct queries.

B Prompt Design

In preliminary prompting trials, the instruction model often produced outputs mixed with natural-language explanations or incomplete SPARQL fragments. Therefore, we use a fine-tuned model as the default Generator and Corrector setting in all main experiments.

B.1 Prompt for the Generator

Table 7 summarizes the prompt for the Generator.

Training

---

system

You are a useful SPARQL assistant. You are tasked to review a question and generate a SPARQL query to answer the question.

SPARQL Database used is WikiData. [<Entity>] is the topic entity in the question. Only use these two prefixes if needed:

```
PREFIX wd: <https://www.wikidata.org/entity/>
PREFIX rdfs: <http://www.w3.org/2000/01/
   rdf-schema#>
```

Do not use `wdt` syntax to query WikiData.

---

user

[Question]

---

assistant

[Ground-truth SPARQL]

Inference

---

system

You are a useful SPARQL assistant. You are tasked to review a question and generate a SPARQL query to answer the question.

SPARQL Database used is WikiData. [<Entity>] is the topic entity in the question. Only use these two prefixes if needed:

```
PREFIX wd: <https://www.wikidata.org/entity/>
PREFIX rdfs: <http://www.w3.org/2000/01/
   rdf-schema#>
```

Do not use `wdt` syntax to query WikiData.

---

user

[Question]

---

assistant

[<Generated by the model>]

Table 7: Prompt for Generator training and inference.

B.2 Prompt for the Corrector

Table 8 summarizes the prompt for the Corrector.

Training

---

system

You are a SPARQL corrector. You are only called when a Gate has decided that the generated SPARQL query needs fixing.

Fix the generated SPARQL query based on the question. If you are not confident about a change, return the original generated SPARQL query unchanged. Do not change the question intent or invent facts.

Only output the final SPARQL query, nothing else.

SPARQL Database used is WikiData. Only use these two prefixes if needed:

```
PREFIX wd: <https://www.wikidata.org/entity/>
PREFIX rdfs: <http://www.w3.org/2000/01/
   rdf-schema#>
```

Do not use `wdt` syntax to query WikiData.

---

user

Question: [Question]
SPARQL: [Initial SPARQL]

---

assistant

[Ground-truth SPARQL]

Inference

---

system

You are a SPARQL corrector. You are only called when a Gate has decided that the generated SPARQL query needs fixing.

Fix the generated SPARQL query based on the question. If you are not confident about a change, return the original generated SPARQL query unchanged. Do not change the question intent or invent facts.

Only output the final SPARQL query, nothing else.

SPARQL Database used is WikiData. Only use these two prefixes if needed:

```
PREFIX wd: <https://www.wikidata.org/entity/>
PREFIX rdfs: <http://www.w3.org/2000/01/
   rdf-schema#>
```

Do not use `wdt` syntax to query WikiData.

---

user

Question: [Question]
SPARQL: [Initial SPARQL]

---

assistant

[<Generated by the model>]

Table 8: Prompt for Corrector training and inference.

C Case Study

C.1 Initial SPARQL is incorrect, and Corrector successfully corrects

In Case 1 shown in Table 9, the question is: "what are the movies that have the same actor as [Wind] and [Maxwell Caulfield] was the lead actor?".

This question requires retrieving movies that satisfy two conditions: they share an actor with movie *Wind*, and Maxwell Caulfield is one of the lead actors in these movies.

In this case, the Generator failed to correctly represent the semantic structure. The main problem is entity-relation role confusion: the model did not correctly distinguish which components should be modeled as entity labels and which should be modeled as relation labels. For instance, the generated SPARQL incorrectly represented "Maxwell Caulfield" as a relation-related component rather than as an entity constraint. Simultaneously, it also failed to correctly use "starred_actors" to represent the starring relationship. In other words, while the Generator captured the keywords in the question, it failed to correctly project the semantic roles of these keywords in the KG query, leading to an error in the query logic.

In contrast, the Corrector successfully fixed this problem. The corrected SPARQL correctly restored the role of "starred_actors" as a relation label and represented "Maxwell Caulfield" as an explicit entity constraint, matching the ground-truth SPARQL. This case demonstrates that the role of the Corrector is not limited to fixing syntax errors but can effectively correct entity-relation role confusion, which is commonly seen in the generation stage. Especially in questions involving multiple constraints, the Corrector can help the system generate a more accurate semantic structure.

Question

What are the movies that have the same actor as [Wind] and [Maxwell Caulfield] was the lead actor?

Generator SPARQL Correctness

False

Corrector SPARQL Correctness

True

Generator SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?rel1 rdfs:label "Maxwell Caulfield" .
  ?e1 ?rel1 ?e2 ;
      rdfs:label "Wind" .
  ?uri ?rel1 ?e2 ;
       ?rel4 ?e4 ;
       rdfs:label ?label .
  FILTER (?label != "Wind")
  ?rel4 rdfs:label "starred_actors" .
  ?e4 rdfs:label "starred_actors" .
}
```

Corrector SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?rel1 rdfs:label "starred_actors" .
  ?e1 ?rel1 ?e2 ;
      rdfs:label "Wind" .
  ?uri ?rel1 ?e2 ;
       ?rel4 ?e4 ;
       rdfs:label ?label .
  FILTER (?label != "Wind")
  ?rel4 rdfs:label "starred_actors" .
  ?e4 rdfs:label "Maxwell Caulfield" .
}
```

Ground-Truth SPARQL

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?rel1 rdfs:label "starred_actors" .
  ?e1 ?rel1 ?e2 ;
      rdfs:label "Wind" .
  ?uri ?rel1 ?e2 ;
       ?rel4 ?e4 ;
       rdfs:label ?label .
  FILTER (?label != "Wind")
  ?rel4 rdfs:label "starred_actors" .
  ?e4 rdfs:label "Maxwell Caulfield" .
}
```

Table 9: Case study of a semantic error corrected by the Corrector.

C.2 Initial SPARQL is correct, but Corrector over-corrects

In Case 2 shown in Table 10, the question is "What does [Matthias Schoenaerts] star in and is classified as [drama]?".

This question requires retrieving movies that satisfy two conditions: Matthias Schoenaerts acted in them, and they are tagged as "drama".

In this case, the Generator has correctly captured the question semantics. The initial SPARQL is consistent with the ground-truth SPARQL in structure and constraints, correctly using "starred_actors" to represent actor relationships and "has_tags" to represent movie tag constraints. Therefore, the Generator was able to accurately express the two core conditions of the original question.

However, the Corrector made unnecessary modifications in this case and corrupted the originally correct SPARQL. For instance, the Corrector incorrectly rewrote "starred_actors" representing the actor relationship to "directed_by". This incorrectly converts the question semantics from "Which movies did Matthias Schoenaerts act in?" to "Which movies did Matthias Schoenaerts direct?". Meanwhile, the Corrector rewrote "has_tags" to "has_genre". Although these two relations appear semantically similar, they are not equivalent in the current KG. Therefore, this substitution changed the original constraint structure, causing the generated SPARQL to no longer correspond to the ground-truth SPARQL.

This case illustrates a typical risk of the Corrector, which is over-correction. When the Generator has already produced a correct SPARQL, the Corrector still attempts to rewrite it, introducing new semantic errors. This demonstrates that although the Corrector can benefit from revising many incorrect samples, without an accurate triggering mechanism, unnecessary correction can also disrupt the originally correct query. This case also illustrates the importance of the Gate: triggering correction only when the Generator output truly has a problem can effectively reduce the negative impact of over-correction.

Question

What does [Matthias Schoenaerts] star in and is classified as [drama]?

Generator SPARQL Correctness

True

Corrector SPARQL Correctness

False

Generator SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?uri ?rel1 ?e1 ;
       ?rel2 ?e2 ;
       rdfs:label ?label .
  ?rel1 rdfs:label "starred_actors" .
  ?e1 rdfs:label "Matthias Schoenaerts" .
  ?rel2 rdfs:label "has_tags" .
  ?e2 rdfs:label "drama" .
}
```

Corrector SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?uri ?rel1 ?e1 ;
       ?rel2 ?e2 ;
       rdfs:label ?label .
  ?rel1 rdfs:label "directed_by" .
  ?e1 rdfs:label "Matthias Schoenaerts" .
  ?rel2 rdfs:label "has_genre" .
  ?e2 rdfs:label "drama" .
}
```

Ground-Truth SPARQL

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?uri ?rel1 ?e1 ;
       ?rel2 ?e2 ;
       rdfs:label ?label .
  ?rel1 rdfs:label "starred_actors" .
  ?e1 rdfs:label "Matthias Schoenaerts" .
  ?rel2 rdfs:label "has_tags" .
  ?e2 rdfs:label "drama" .
}
```

Table 10: Case study of an initially correct SPARQL query corrupted by the Corrector.

C.3 Neither Generator nor Corrector obtains the correct SPARQL

In Case 3 shown in Table 11, the question is "Who wrote the movie [Toy Story 3] and also [Toy Story 2]?".

This question requires finding the author who wrote both *Toy Story 3* and *Toy Story 2*. The key in the SPARQL query is to model a "shared author" structure, i.e., the same author must be related to both movies through the "written_by" relationship.

However, the Generator did not correctly model this logical structure. Firstly, it did not generate a complete SPARQL query. Second, it introduced a specific person who was not provided in the question. More importantly, it incorrectly compressed two different movies into one variable, making the same entity represent both *Toy Story 3* and *Toy Story 2*, which obviously violates the "shared author" structure in the original question. This shows the Generator did not capture the core logic in the question, and instead generated an incorrect structure for the overall semantic.

In this case, the Corrector did not improve the Generator's result either. Instead, it kept the original error. This indicates that, when the error is not a local label mismatch or relation constraint error, but involves logical composition, the capability of the Corrector is limited.

Question

Who wrote the movie [Toy Story 3] and also [Toy Story 2]?

Generator SPARQL Correctness

False

Corrector SPARQL Correctness

False

Generator SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

{
  ?movie rdfs:label "Toy Story 3" .
  ?author rdfs:label "John Lasseter" .
  ?movie rdfs:label "Toy Story 2" .
}
```

Corrector SPARQL Response

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

{
  ?movie rdfs:label "Toy Story 3" .
  ?author rdfs:label "John Lasseter" .
  ?movie rdfs:label "Toy Story 2" .
}
```

Ground-Truth SPARQL

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
    schema#>

SELECT DISTINCT ?label ?uri WHERE {
  ?e1 ?rel ?uri ;
      rdfs:label "Toy Story 3" .
  ?e3 ?rel ?uri ;
      rdfs:label "Toy Story 2" .
  ?rel rdfs:label "written_by" .
  ?uri rdfs:label ?label .
}
```

Table 11: Case study of an uncorrected SPARQL error.

D Additional Ablation Study

Subset	W1.0	W1.0 + C0.2	W1.0 + C0.5	W1.0 + C0.8
Correction Required	89.58	92.38	93.46	95.30
Originally Wrong	82.55	85.20	86.10	87.98
Originally Correct	99.98	99.99	99.99	99.98

Table 12: Query-level accuracy (%) on three subsets for Gate–Corrector coordination analysis. "Correction Required" denotes samples triggered by the Gate. "Originally Wrong" denotes samples incorrectly generated by the Generator. "Originally Correct" denotes samples correctly generated by the Generator.

D.1 Subset-level Gate-Corrector Coordination

Since the Corrector is only applied to samples that the Gate predicts as requiring correction, overall pipeline performance alone cannot fully explain how the two modules interact. Therefore, we analyze query-level accuracy on three subsets: samples triggered by the Gate, samples originally generated incorrectly by the Generator, and samples originally generated correctly by the Generator. This allows us to examine whether the Gate routes high-risk queries to the Corrector, whether the Corrector can repair truly wrong queries, and whether the correction process damages originally correct queries.

As shown in Table 12, the performance on the correction-required subset improves from 89.58% under $W1.0$ to 95.30% under $W1.0 + C0.8$, indicating that adding correct samples to Corrector training improves its stability on the actual samples routed by the Gate. A similar trend is observed on the originally wrong subset, where query-level accuracy increases from 82.55% to 87.98%, suggesting that adding correct samples does not weaken the Corrector's repair ability. Meanwhile, performance on the originally correct subset remains nearly perfect across all settings. This indicates that the Corrector introduces very few additional errors when the input query is already correct. Overall, the results show that selective correction works through the coordination of both modules: the Gate routes high-risk queries to the Corrector, while the Corrector repairs wrong queries without substantially damaging correct ones.

D.2 Preliminary Results on SciQA

We further evaluate the proposed framework on SciQA (Auer et al., 2023) as an additional dataset experiment. SciQA is a scientific question answering benchmark over the Open Research Knowledge Graph (ORKG), containing NLQ-SPARQL pairs with answers retrieved from the ORKG. It contains 2,565 examples in total, with 1,795 training, 257 validation, and 513 test examples. Since SciQA differs from MCQA in both domain and knowledge graph, we use it as a preliminary setting to examine whether the proposed correction mechanism can improve the same Generator backbone beyond the main MCQA setting.

Error Type	Count	Percentage (%)
Syntax Error	14	25.00
Semantic Error	42	75.00
Total	56	–

(a) Generator error distribution.

Setting	Query Acc. (%)	Item F1 (%)
Generator-only	80.70	93.94
Gen. + Gate + Corr.	91.62	98.31
Gain	+10.92	+4.37

(b) Results on SciQA.

Table 13: Analysis on SciQA. *Gen. + Gate + Corr.* denotes the Generator-Gate-Corrector framework with a RoBERTa-based Gate.

The error distribution in Table 13a shows that most Generator-side failures are semantic errors, accounting for 42 out of 56 errors, or 75.00%. In contrast, syntax errors account for 14 cases, or 25.00%. This pattern is consistent with the MCQA error analysis, providing preliminary evidence that semantic errors are a common source of failure across different Text-to-SPARQL settings.

Rather than comparing systems with different model scales or training budgets, we focus on the relative gain obtained by applying the GGC framework to the same Generator backbone. As shown in Table 13b, the framework improves query accuracy from 80.70% to 91.62%, yielding a 10.92 percentage-point gain over the Generator-only setting. It also improves item micro F1 from

93.94% to 98.31%. These results provide supplementary evidence that selective correction can improve SPARQL generation outside the main MCQA setting, although more comprehensive additional dataset evaluation remains an important direction for future work.