From Decomposition to Validation: A Multi-stage Framework for Temporal Knowledge Graph Question Answering with LLMs

Anonymous ACL submission

Abstract

Temporal Knowledge Graph Question Answering (TKGQA) requires reasoning over time-dependent facts. However, existing approaches suffer from two fundamental limitations: (1) general-purpose text retrievers often fail to properly align entities, relations, and temporal constraints in the question with relevant TKG facts; and (2) large language models (LLMs) remain unreliable in temporally constrained reasoning, often making incorrect temporal comparisons or boundary-sensitive judgments. To address these challenges, we propose **TempReasoner**, a multi-stage framework for TKGQA. TempReasoner decomposes complex temporal questions into structured sub-questions, retrieves temporally relevant facts using a trained TKG-Retriever, filters noisy facts through LLM-generated alignment signals, and generates answers along with a supporting evidence set. A dedicated validation module further refines uncertain predictions by issuing verification queries, improving both answer accuracy and reliability. Experiments on two widely used benchmarks, MultiTQ and CronQuestions, show that TempReasoner consistently delivers strong performance. In particular, TempReasoner achieves substantial improvements on multi-step temporal questions, demonstrating its effectiveness in temporal reasoning and evidence grounding. Additional analyses further confirm the effectiveness of the proposed TKG-Retriever and show that TempReasoner remains effective across several LLM backbones under the same pipeline configuration.

1 Introduction

Many real-world questions involve explicit or implicit temporal constraints, requiring accurate reasoning over time. However, current large language models (LLMs) remain unreliable when answering temporal questions. As illustrated in Figure 1(a), directly querying LLMs with temporal questions often leads to incorrect answers without background knowledge. To better support temporally grounded reasoning, the Temporal Knowledge Graph Question Answering (TKGQA) task has been introduced, which leverages temporal knowledge graphs (TKGs) as structured background knowledge to represent time-dependent facts and facilitate temporal reasoning (Chen et al., 2024c; Gao et al., 2024; Qian et al., 2024; Gong et al., 2025; Hu et al., 2025).

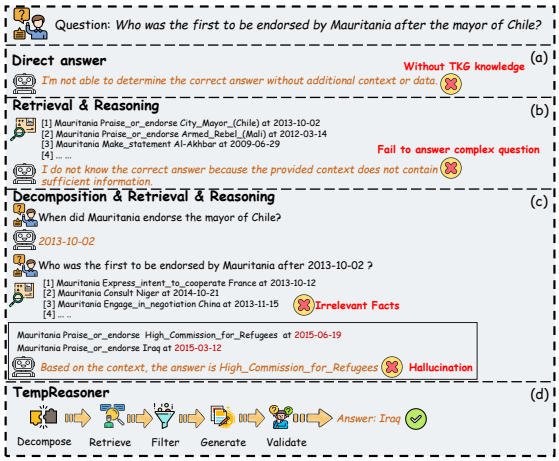

Figure 1: Illustration of the paradigms in the current LLM-based TKGQA task. (a) Direct LLM answering. (b) Retrieval & Reasoning. (c) Decomposition & Retrieval & Reasoning. (d) Our proposed TempReasoner

In TKGQA, a straightforward **Retrieval & Reasoning** paradigm, as shown in Figure 1(b), is often insufficient for complex multi-step temporal questions. To mitigate this limitation, several methods (Chen et al., 2024c; Gong et al., 2025) adopt a **Decomposition & Retrieval & Reasoning** strategy that decomposes complex questions into sub-questions, performs retrieval and reasoning for each sub-question, and aggregates the intermediate results to produce the final answer. Although this strategy alleviates the difficulty of complex temporal reasoning to some extent, it still faces several fundamental challenges:

Limitations of retrievers for TKGs. A central

difficulty in TKGQA is that the correct supporting fact must compete against many lexically related but only partially aligned candidates. As illustrated in Figure 1(c), for the question "*Who was the first to be endorsed by Mauritania after 2013-10-02?*", the target relation should be *Praise_or_endorse*, yet a general-purpose retriever may return semantically proximate but irrelevant facts such as "*Mauritania Engage_in_negotiation China at 2013-11-15*". Our controlled diagnostic further confirms that retrieval misses account for a substantial portion of TKGQA failures, suggesting that retrieval for TKGQA must explicitly model entity, relation, and temporal alignment rather than relying only on generic semantic similarity (see Appendix A.1).

Limitations of LLMs in generating answers for temporal reasoning. Retrieval failure alone cannot fully explain the errors observed in TKGQA. As also shown in Figure 1(c), an LLM may still produce an incorrect answer even when a relevant fact such as "*Mauritania Praise_or_endorse Iraq at 2015-03-12*" is already present in the retrieved context. Our controlled diagnostic further shows that such errors persist even when gold-relevant evidence has been retrieved (see Appendix A.1), indicating that robust TKGQA requires not only better retrieval but also explicit verification of whether the generated answer is truly supported by temporally consistent evidence.

To overcome these challenges, we propose **TempReasoner**, a multi-stage framework for Temporal Knowledge Graph Question Answering. TempReasoner follows a staged processing pipeline in which each step is handled by a specialized module. The framework consists of five core modules: *Decompose*, *Retrieve*, *Filter*, *Generate*, and *Validate*, each responsible for a distinct stage in the reasoning workflow.

We summarize our contributions as follows:

- We propose **TempReasoner**, a TKGQA framework for multi-step temporal inference that combines decomposition, structure-aligned retrieval, alignment-based evidence filtering, and evidence-grounded validation.
- We develop a temporal-sensitive and structure-aware **TKG-Retriever** trained with temporally informed hard negatives. This enables more accurate alignment between question semantics, temporal constraints, and TKG facts in the TKGQA task.
- Experiments on MultiTQ (Chen et al., 2023) and CronQuestions (Saxena et al., 2021) demonstrate that TempReasoner consistently outperforms representative baselines across multiple evaluation dimensions, with particularly notable gains on complex temporal questions.

2 Related Work

Existing approaches to temporal knowledge graph question answering (TKGQA) can be broadly grouped into **semantic similarity–based**, **semantic parsing–based**, and **retrieval-augmented generation** methods.

Semantic similarity–based methods embed questions and temporal facts into a shared representation space and perform reasoning via similarity scoring. Representative works include ExaQT (Jia et al., 2021), LGQA (Liu et al., 2023), TwiRGCN (Sharma et al., 2023), as well as language-model-based approaches such as TempoQR (Mavromatis et al., 2022) and MultiQA (Chen et al., 2023). While effective for simple queries, these methods often struggle with complex multi-step temporal reasoning due to the lack of explicit stepwise inference.

Semantic parsing–based methods convert questions into executable logical forms defined over temporal operators, exemplified by TEQUILA (Jia et al., 2018), SF-TQA (Ding et al., 2022), and Prog-TQA (Chen et al., 2024b). Although interpretable, these approaches are sensitive to parsing errors and prone to cascading failures in complex scenarios.

Retrieval-augmented generation methods retrieve relevant temporal facts or subgraphs and rely on LLMs for reasoning and answer generation, including GenTKGQA (Gao et al., 2024), TimeR4 (Qian et al., 2024), ARI (Chen et al., 2024c), TempAgent (Hu et al., 2025), RTQA (Gong et al., 2025), and PoK (Qian et al., 2025). Most existing methods couple retrieval and generation within a single reasoning loop, with limited explicit support for error validation or correction. By contrast, our framework improves reasoning reliability through the integration of decomposition, structure-aligned rewriting, alignment-signal-based filtering, and evidence-grounded validation. A capability-level comparison with the representative TKGQA frameworks is provided in Appendix A.2.

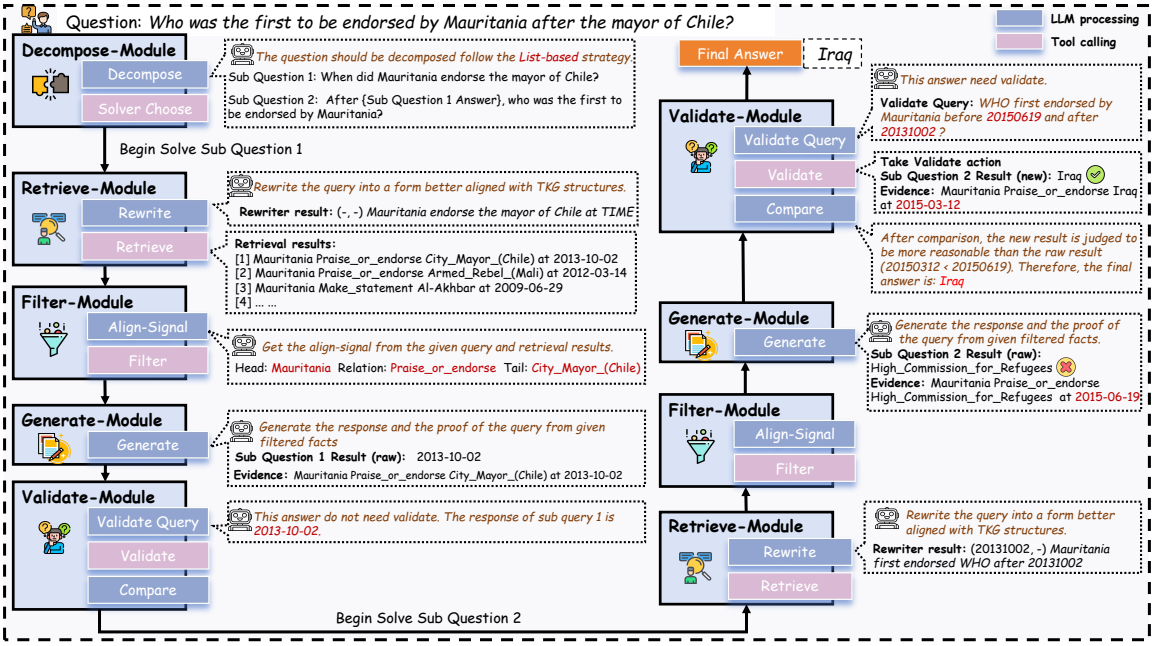

Figure 2: Overview of the *TempReasoner* framework. The figure illustrates this complete process on the example question "*Who was the first to be endorsed by Mauritania after the mayor of Chile?*".

3 Method

3.1 Overview

We propose TempReasoner, a multi-stage framework that performs TKGQA through structured decomposition, retrieval, filtering, generation, and validation. As illustrated in Figure 2, TempReasoner treats the QA task as a multi-stage reasoning process to produce a reliable answer.

The *Decompose-Module* (Section 3.2) selects a List- or Tree-based strategy based on the question type and decomposes the question into sub-questions, whose results are later combined according to the selected strategy. The *Retrieve-Module* (Section 3.3) rewrites the question into a TKG-compatible query format and retrieves the top-K temporal facts using a trained retriever. The *Filter-Module* (Section 3.4) constructs an alignment signal to filter noisy or irrelevant evidence. The *Generate-Module* (Section 3.5) produces a candidate answer based on the filtered facts. Finally, the *Validate-Module* (Section 3.6) generates a validation query to assess the correctness of the candidate answer and selects the final answer based on validation consistency, thereby improving answer reliability. A detailed illustration of the full algorithmic pipeline can be found in Appendix A.3 and we provide task preliminaries in Appendix A.4.

Each module in TempReasoner follows a unified design pattern based on LLM-driven reasoning and tool-calling. Details of the module architectures are given in Appendix A.5.

3.2 Question-structure–based Decomposition

The *Decompose-Module* is designed to decompose temporal questions into structured sub-questions. We adopt two representative decomposition strategies according to question structure: a List-based strategy and a Tree-based strategy. The List-based strategy handles sequential queries that require stepwise inference. For example, in the query *"Who was the first to be endorsed by Mauritania after the mayor of Chile?"*, one must first determine *when* Mauritania endorsed the mayor of Chile, and only then answer *who was the first* after that timestamp. The Tree-based strategy targets queries that require retrieving multiple partial results in parallel before combining them. For instance, the query *"Which country did China wish to meet with in the same month as Japan?"* requires retrieving whom China wished to meet and whom Japan wished to meet, and then identifying the entity that satisfies the same-month temporal constraint across both results. We formalize the decomposition process as:

$$(q_1, q_2, \dots) = \text{DecomposeModule}(Q, \mathcal{P}, S), \quad S \in \{\text{List-based}, \text{Tree-based}\}, \tag{1}$$

where Q denotes the original input question, $\mathcal{P}$ represents the instruction prompt used to guide the

Table 1: Example of sample construction in *TKG-Retriever*. The corresponding TKG fact is (*Agence_France-Presse*, *Praise_or_endorse*, *Iraq*, 2014-10-01). Three query templates are used: the raw-query template, the last-query template, and the first-query template. Five negative-sample templates are constructed: *inv_neg*, *sim_ent_neg*, *sim_rel_neg*, *temp_last_neg*, and *temp_first_neg*. All examples in the table place `WHO` in the subject position, and analogous templates can be constructed by placing `WHO` in the object position.

query-template	query example	neg-template	neg example
raw-query		***inv_neg***	
		$\{o\}$ $\{r\}$ $\{s\}$ at $\{t\}$	Iraq Praise_or_endorse Agence_France-Presse at 2014-10-01
		sim_ent_neg	
WHO $\{r\}$ $\{o\}$ at $\{t\}$?	WHO Praise or endorse Iraq at 2014-10-01 ?	$\{s_{sim}\}$ $\{r\}$ $\{o\}$ at $\{t\}$	Government_(France) Praise_or_endorse Iraq at 2014-10-01
		sim_rel_neg	
		$\{s\}$ $\{r_{sim}\}$ $\{o\}$ at $\{t\}$	Agence_France-Presse Make_optimistic_comment Iraq at 2014-10-01
last-query		***temp_last_neg***	
WHO last $\{r\}$ $\{o\}$ before $\{t+1\}$?	WHO last Praise or endorse Iraq before 2014-10-02 ?	$\{s_{sim}\}$ $\{r\}$ $\{o\}$ at $\{< t\}$	Government_(France) Praise_or_endorse Iraq at 2014-09-20
first-query		***temp_first_neg***	
WHO first $\{r\}$ $\{o\}$ after $\{t-1\}$?	WHO first Praise or endorse Iraq after 2014-9-30 ?	$\{s_{sim}\}$ $\{r\}$ $\{o\}$ at $\{> t\}$	Civil_Service_(France) Praise_or_endorse Iraq 2014-10-07

LLM-based decomposition process, S indicates the decomposition strategy, and q_i is the i-th sub-question produced by the *Decompose-Module*.

3.3 TKG-aware Retrieval

The *Retrieve-Module* treats the question as a query and retrieves relevant facts from the TKG. General-purpose retrievers such as BGE-M3 (Chen et al., 2024a) often misalign entities or relations and fail to recognize temporal constraints embedded in queries. To mitigate these limitations, we design a retrieval pipeline that explicitly accounts for the structural and temporal characteristics of TKGs.

Structure-aligned rewrite. Each question is rewritten into a format that is aligned with the canonical quadruple structure used in temporal knowledge graphs. The rewritten query consists of (1) a temporal-constraint component that extracts explicit time bounds and restricts retrieval accordingly, and (2) a textual component where `TIME` and `WHO` denote unspecified timestamps and entities. Temporal operators such as *first/last* and *before/after* are preserved to guide later retrieval. For example, the question "*Who was the first that China wanted to negotiate with after 20141010?*" becomes "$(20141010, -)$ *China first want_to_negotiate_with WHO after 20141010*," where $(20141010, -)$ restricts retrieval to facts occurring strictly after the specified date. This step facilitates better alignment between the question and the TKG storage format, thereby reducing semantic drift.

Notably, this rewriting step is not merely temporal grounding or time completion. Instead, it performs structure-level alignment by making entities, relations, and temporal constraints explicit in a retrieval-oriented format.

TKG-Retriever. To further enhance retrieval accuracy, we train a *TKG-Retriever* that incorporates semantic similarity and temporal order. We construct harder negative pairs that enforce fine-grained semantic and temporal discrimination instead of random negatives such as (Qian et al., 2024). For each entity e and relation r, we gather sets of semantically similar entities e_{sim} and relations r_{sim}. We introduce a residual quantization method for efficiently identifying semantically similar entities; details are provided in Appendix A.6. Given a fact (s, r, o, t), its query is treated as a positive instance, and multiple hard negatives are constructed, including: (*i*) *inv_neg*: swapping subject and object, (*ii*) *sim_ent_neg*: replacing the subject with an entity from e_{sim}, (*iii*) *sim_rel_neg*: replacing the relation with one from r_{sim}.

To encode temporal sensitivity, we extend the raw-query template to *last-query* and *first-query* variants by injecting operators (*last/before* or *first/after*) and shifting timestamps to $t{+}1$ or $t{-}1$ (e.g., 2014-10-01 + 1 $\rightarrow$ 2014-10-02). Corresponding temporal negatives (*temp_last_neg*, *temp_first_neg*) substitute the subject with a similar entity and assign timestamps $< t$ or $> t$. These negatives encourage the retriever to jointly model entity identity, relation semantics, and chronological ordering.

For each fact, we construct N_{neg} hard negatives, and treat other samples within the batch as in-batch negatives. The *TKG-Retriever* is trained using an InfoNCE loss $\mathcal{L}$:

$$Z = \exp(\mathrm{sim}(q,p)/\tau) + \exp(\mathrm{sim}(q,n_{\text{hard}})/\tau) + \sum_{b \in \mathcal{N}_{\text{batch}}} \exp(\mathrm{sim}(q,b)/\tau),$$
$$\mathcal{L} = -\log \frac{\exp(\mathrm{sim}(q,p)/\tau)}{Z}, \quad (2)$$

where q denotes the embedding of the rewritten query, p is the embedding of its corresponding positive TKG fact, n_{hard} denotes one of the constructed hard negative samples, and $\mathcal{N}_{\text{batch}}$ denotes the set of in-batch negative samples within the same training batch. The function $\text{sim}(\cdot, \cdot)$ measures the similarity between two embeddings (e.g., cosine similarity), and τ is the temperature coefficient.

For each rewritten query, we apply the trained *TKG-Retriever* to obtain facts aligned with both its semantic content and temporal constraints. The retrieved evidence is then passed to the *Filter-Module* for further noise reduction.

3.4 Alignment-based Filtering

The *Filter-Module* is designed to mitigate noise in the facts retrieved by the retriever. Prior approaches either directly feed raw retrieved facts into the LLM (Gong et al., 2025) or apply a reranking step (Qian et al., 2024). However, these methods often lack explicit mechanisms to reliably filter irrelevant evidence, which may still mislead the LLM during reasoning.

To address this issue, the *Filter-Module* constructs an *alignment signal* from the retrieved evidence and applies it to filter noisy facts, producing a denoised evidence set. As illustrated in Figure 2, consider the sub-question #1: *"When did Mauritania endorse the mayor of Chile?"*. In this case, the question implies a target relation corresponding to *Praise_or_endorse*, with *Mauritania* as the subject. However, the raw retrieved set contains semantically related but irrelevant facts, such as *(Mauritania, Make_statement, Al-Akhbar, 2009-06-29)*.

The *Filter-Module* generates an alignment signal based on the top-n retrieved facts and uses this signal to exclude such noisy evidence. One example of a generated alignment signal for this sub-question is shown below:

Alignment.Head: Mauritania
Alignment.Relation: Praise_or_endorse
Alignment.Tail: ?

After filtering, only the most relevant fact *(Mauritania, Praise_or_endorse, City_Mayor_(Chile), 2013-10-02)* is retained, providing cleaner and more reliable input for subsequent answer generation. This filtering step reduces the impact of irrelevant evidence and helps alleviate error propagation and hallucinations in later stages. Additional implementation details of the *Filter-Module* are provided in Appendix A.7.

3.5 Controlled Answer Generation

The *Generate-Module* is responsible for producing a candidate answer for each sub-question after retrieval and filtering. Given a sub-question q_i produced by the *Decompose-Module* and a filtered fact set $\mathcal{F}_i$ returned by the *Filter-Module*, the *Generate-Module* generates an answer together with a supporting *evidence set*, which consists of a subset of facts from $\mathcal{F}_i$ that the model identifies as evidence for the generated answer.

We formalize the behavior of the *Generate-Module* as the following mapping:

$$(a_i, \hat{f}_i) = \text{GenerateModule}(q_i, \mathcal{F}_i, \Theta_{\text{LLM}}), \quad (3)$$

where q_i denotes the i-th sub-question, $\mathcal{F}_i = \{f_1, f_2, \ldots, f_m\}$ is the filtered fact set aligned with q_i, and Θ_{LLM} represents the parameters of the LLM used for answer generation. The output a_i is a raw candidate answer for the sub-question, and $\hat{f}_i \subseteq \mathcal{F}_i$ denotes the set of supporting facts selected by the model as evidence for a_i.

Although the *Generate-Module* produces an answer candidate, it does not guarantee correctness or optimality. Instead, answer reliability is achieved through the upstream constraints imposed by retrieval and filtering, and is further refined by the downstream *Validate-Module*, which explicitly verifies and revises uncertain predictions.

3.6 Answer Validation and Self-Correction

The *Validate-Module* assesses whether a candidate answer a_i generated for a sub-question q_i requires further verification. Given q_i, its generated answer a_i, and the associated supporting facts $\hat{f}_i$, the module first examines whether a_i is sufficiently supported by $\hat{f}_i$ and satisfies the temporal and relational constraints specified in q_i. If the retrieved evidence is consistent with the query constraints, no additional validation is performed.

When the support for a_i is insufficient or ambiguous, the *Validate-Module* constructs a validation query that explicitly re-examines the temporal and relational conditions implied by the candidate answer. Importantly, this validation query is designed to *challenge* the current answer by imposing stricter or counterfactual temporal constraints, rather than reinforcing the original prediction.

As illustrated in Figure 2, for the sub-question "*Mauritania first endorsed WHO after 20131002*",

the module formulates a validation query based on the raw answer: *"WHO was first endorsed by Mauritania after 20131002 and before 20150619?"*. This query is processed through the same Retrieve–Filter–Generate pipeline to obtain an alternative candidate answer $\hat{a}_i$.

The final answer is selected as:

$$a_i^* = \text{Select}(a_i, \hat{a}_i), \tag{4}$$

where the selection function prefers the answer that is more strongly supported by retrieved evidence and better satisfies the imposed temporal constraints.

4 Experiments

4.1 Datasets and Baselines

We evaluate TempReasoner on two widely used TKGQA benchmarks: MultiTQ (Chen et al., 2023) and CronQuestions (Saxena et al., 2021). Dataset statistics are provided in Appendix A.8.

We select a set of representative baselines for comparison, which can be broadly categorized into three groups: (1) **PLM-based methods**, including BERT (Devlin et al., 2019), ALBERT (Lan et al., 2020), and DistilBERT (Sanh, 2019). (2) **Embedding-based methods**, such as EmbedKGQA (Saxena et al., 2020), CronKGQA (Saxena et al., 2021), and MultiQA (Chen et al., 2023). (3) **LLM-based methods**, which can be further divided into two subcategories: models that directly rely on LLMs to produce answers, including LLaMA2 and models that incorporate knowledge graph information into LLM reasoning, including KG-RAG, CoT (Wei et al., 2022), ReAct (Yao et al., 2022), ARI (Chen et al., 2024c), RTQA (Gong et al., 2025), TempAgent (Hu et al., 2025), and PoK (Qian et al., 2025). For several baselines, we report results following (Chen et al., 2024c) and (Gong et al., 2025).

4.2 Implementation Details

For training the TKG-Retriever, we fine-tune BGE-M3 (Chen et al., 2024a) as the base encoder. The model is trained on two NVIDIA A800 GPUs (80GB memory each), after which we build the retrieval index using FAISS Library (Johnson et al., 2019) for efficient similarity search. For each quadruple, we construct $N_{\text{neg}} = 10$ negatives: five with `WHO` in the subject position and five in the object position.

We employ `DeepSeek-V3-0324` model via the DeepSeek API as the backbone LLM for all agents during evaluation. The retrieval context size is set to 100 facts, while the *Filter-Module* uses the top 20 retrieved facts to construct the filter alignment signals. We follow the data preprocessing as (Chen et al., 2024c). Model performance is evaluated using Hits@1, which measures the percentage of questions whose top-ranked prediction matches the ground-truth answer.

4.3 Main Results

Model	Overall	Question Type		Answer Type	
		Multiple	Single	Entity	Time
BERT	0.083	0.061	0.092	0.101	0.040
DistilBERT	0.083	0.074	0.087	0.102	0.037
ALBERT	0.108	0.086	0.116	0.139	0.032
EmbedKGQA	0.206	0.134	0.235	0.290	0.001
CronKGQA	0.279	0.134	0.337	0.328	0.156
MultiQA	0.293	0.159	0.347	0.349	0.157
LLaMA2	0.185	0.101	0.220	0.239	0.055
ARI	0.380	0.210	0.680	0.394	0.344
TimeR4	0.728	0.335	0.887	0.639	0.945
TempAgent	0.702	0.316	0.857	0.624	0.870
RTQA	0.765	0.424	0.902	0.692	0.942
PoK	0.779	0.409	**0.929**	0.696	**0.962**
TempReasoner	**0.805**	**0.631**	0.875	**0.765**	0.899

Table 2: Performance of Hits@1 on the MultiTQ dataset. **Bold** numbers denote the best performance, and underlined numbers denote the second-best performance in each column.

Model	Overall	Question Type		Answer Type	
		Simple	Complex	Entity	Time
BERT	0.243	0.249	0.239	0.277	0.179
ALBERT	0.248	0.255	0.235	0.279	0.177
EmbedKGQA	0.288	0.290	0.286	0.411	0.056
CronKGQA	0.647	0.987	0.392	0.699	0.549
KG-RAG	0.490	0.460	0.518	0.470	0.520
CoT	0.640	0.690	0.610	0.620	0.660
ReAct	0.685	0.835	0.525	0.650	0.755
ARI	0.707	0.860	0.570	0.660	0.800
TempAgent	0.842	0.895	0.640	0.805	**0.921**
TempReasoner	**0.850**	**0.919**	**0.799**	**0.846**	0.857

Table 3: Performance of Hits@1 on the CronQuestions. **Bold** numbers denote the best performance, and underlined numbers denote the second-best performance in each column.

We present the comparative results of TempReasoner against baseline models on the MultiTQ and

CronQuestions datasets in Table 2 and Table 3 respectively. Specifically, TempReasoner achieves consistently strong results across both MultiTQ and CronQuestions.

On **MultiTQ**, TempReasoner obtains an Overall Hits@1 of 0.805, outperforming the strongest prior baseline PoK. When examining question types, TempReasoner achieves 0.631 on Multiple questions, with the latter representing a large improvement of 20.7% over the previous best model. Multi-step temporal queries often require chaining multiple temporal conditions or selecting entities under strict ordering constraints, and TempReasoner's improvements in this category highlight its enhanced reasoning capability. In terms of answer types, TempReasoner reaches 0.765 on Entity questions and 0.899 on Time questions. The improvement of 6.9% on Entity answers is notable, as most multi-step temporal questions require identifying the correct entity.

On **CronQuestions**, TempReasoner also achieves strong overall performance, with an Overall Hits@1 of 0.850, outperforming all baseline models. TempReasoner performs consistently well across these settings, reaching 0.919 on Simple questions and 0.799 on Complex questions. In particular, its performance on Complex questions shows a clear advantage over prior methods, exceeding the next-best model (TempAgent) by nearly 16%. This result indicates that TempReasoner is more effective at handling multi-step reasoning under temporal constraints. For answer types, TempReasoner achieves 0.846 on Entity questions and 0.857 on Time questions, showing strong overall performance across both answer types and clear gains on Entity questions.

In addition, TempReasoner performs slightly worse than some baselines, particularly for questions with time as the final answer. This performance gap is mainly attributed to occasional errors in temporal expression generation by the LLM. Even when the retrieved evidence is correct, the model may produce natural language expressions (e.g., *January*) instead of the required normalized time format (e.g., *2014-01*), which leads to mismatches under strict evaluation.

4.4 Ablation Study

We further conduct ablation studies to evaluate the contribution of each module in our framework. The results are summarized in Table 4.

Effect of the Decompose-Module. The *w/o Dec.* variant processes questions in their original form without decomposition. Its performance drops by 15.7% overall and by 40.7% on complex *Multiple* questions, showing that the *Decompose-Module* is crucial for multi-step temporal reasoning. It performs slightly better on *equal* questions, likely because a small number of simple questions are unnecessarily decomposed, which may introduce errors during sub-question solving. Appendix A.9 further shows that the remaining failures are not dominated by decomposition alone, but are distributed across decomposition, alignment-signal induction, retrieval, and answer generation stages.

Effect of the Retrieve-Module and Filter-Module. The *w/o Ret.* variant retrieves context using the BGE-M3 encoder with FAISS instead of the proposed *Retrieve-Module* + *Filter-Module*. This variant shows consistent degradation across all metrics, with an overall drop of 29.5%, highlighting that our structure-aligned retrieval combined with temporal filtering is significantly more effective at extracting relevant temporal facts from the TKG.

Effect of the Validate-Module. The *w/o Val.* variant represents a configuration in which the LLM-generated answers are used directly without verification. This leads to a 7.1% decrease on the overall score. Notably, compared with the moderate drop on *equal* questions (-0.9%), performance decreases much more sharply on stricter temporal constraints such as *after_first* (-21.2%) and *before_last* (-11.2%). These results indicate that the *Validate-Module* effectively identifies and corrects erroneous LLM outputs, especially under fine-grained temporal reasoning conditions. We further compare the type-wise API usage of TempReasoner with and without validation in Appendix A.10, which shows that the additional overhead is moderate on average and mainly concentrated on harder temporal operators.

5 Further Analysis

To better understand the behavior and generalization properties of TempReasoner, we conduct a series of further analyses from both the model and retrieval perspectives. Additional error analysis (Appendix A.9), API-overhead statistics (Appendix A.10), and case studies (Appendix A.11) are provided in the appendix.

	Overall	answer type		qtype						qlabel	
		entity	time	after_first	before/after	before_last	first/last	equal	eq_multi	multiple	single
TempReasoner	**0.805**	**0.765**	**0.899**	**0.571**	**0.820**	**0.655**	**0.891**	0.906	**0.700**	**0.631**	**0.875**
w/o Dec.	0.648	0.551	0.868	0.283	0.663	0.109	0.855	**0.910**	0.341	0.224	0.816
w/o Ret.	0.510	0.439	0.677	0.283	0.438	0.359	0.680	0.615	0.205	0.298	0.625
w/o Val.	0.734	0.671	0.876	0.359	0.713	0.543	0.859	0.897	0.682	0.496	0.829

Table 4: Ablation study on the MultiTQ dataset. **Bold** numbers denote the best performance; underlined numbers denote the second-best performance in each column.

5.1 Performance of different LLMs

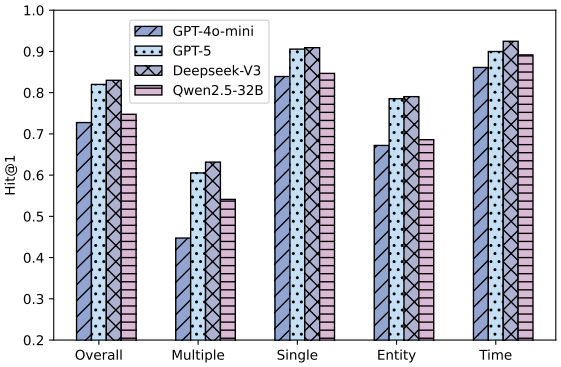

Figure 3: Performance comparison across different LLMs on four question types.

To further examine the adaptability of TempReasoner, we evaluated our framework using four LLMs ranging across different model scales: `DeepSeek-V3`, `GPT-4o-mini`, `GPT-5`, and `Qwen2.5-32B`. The results shown in Figure 3 indicate that models with stronger reasoning capabilities generally achieve higher performance. Nevertheless, TempReasoner still maintains strong performance even when paired with comparatively weaker models such as `GPT-4o-mini` and `Qwen2.5-32B`. This suggests that TempReasoner remains effective across several LLM backbones under the same pipeline configuration.

5.2 Performance of different Retrievers

To evaluate the effectiveness of the proposed TKG-Retriever, we compare it with two widely used text retrievers, BGE-M3 and Sentence-BERT (Reimers and Gurevych, 2019). All retrievers encode TKG facts and build FAISS indexes over their vector representations, and candidate facts are retrieved from the corresponding index for each query. Retrieval effectiveness is measured by the average rank of the ground-truth fact in the retrieved list, where a lower rank indicates better retrieval quality.

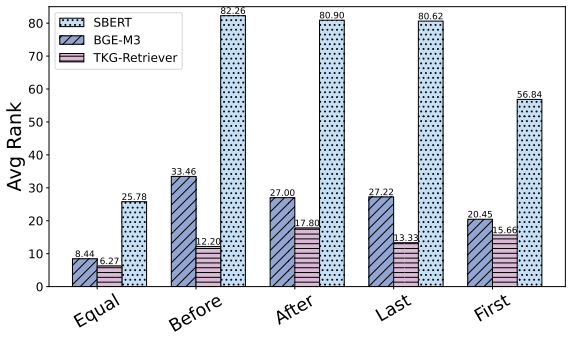

Figure 4: Comparison of retrieval performance between different retrievers.

As shown in Figure 4, the TKG-Retriever consistently outperforms both BGE-M3 and Sentence-BERT across all evaluation settings. The performance gap is particularly pronounced for queries involving temporal constraints such as *before/after* and *last/first*, where sensitivity to temporal ordering is crucial. These results demonstrate that the proposed TKG-Retriever achieves more reliable temporal fact retrieval on TKGs.

6 Conclusion

We presented **TempReasoner**, a multi-stage framework for TKGQA that integrates question decomposition, TKG-aware retrieval, alignment-based filtering, controlled answer generation, and verification-based validation. By explicitly structuring temporal reasoning, TempReasoner addresses the limitations of general-purpose retrievers and improves answer reliability under temporal constraints. Experiments on MultiTQ and CronQuestions show consistent improvements, especially on complex multi-step temporal queries. Additional analyses further confirm the effectiveness of the proposed TKG-Retriever and show that TempReasoner remains effective across several LLM backbones. Overall, TempReasoner provides a principled framework for combining structured temporal reasoning with LLMs for TKGQA.

Limitations

While TempReasoner demonstrates strong performance on complex temporal question answering, it still has several limitations.

First, the proposed framework primarily operates at the level of individual temporal facts (quadruples) during retrieval and reasoning. Although the multi-stage design effectively mitigates noise and temporal misalignment, it does not explicitly exploit the global topological structure of temporal knowledge graphs, such as multi-step connectivity patterns or path-level dependencies. As a result, certain forms of structural reasoning that rely on explicit graph traversal or subgraph-level aggregation are not directly modeled.

Second, TempReasoner relies on LLMs to perform key intermediate steps, including question decomposition and alignment signal induction. While the downstream filtering and validation modules substantially reduce error propagation, imperfect decomposition or signal generation may still affect performance in some cases, particularly for questions with highly implicit temporal constraints.

In many real-world applications, temporally grounded knowledge is primarily stored in unstructured documents rather than explicit temporal knowledge graph form, and such data often contains substantial noise. An important direction for future work is to efficiently construct temporal knowledge graphs from noisy temporal documents and perform retrieval and reasoning over the resulting structured graphs.

Ethics Statement

All steps and data described in our paper follow the ACL Ethics Policy[1].

References

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Zhuo Chen, Zhao Zhang, Zixuan Li, Fei Wang, Yutao Zeng, Xiaolong Jin, and Yongjun Xu. 2024b. Self-improvement programming for temporal knowledge graph question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 14579–14594.

Ziyang Chen, Dongfang Li, Xiang Zhao, Baotian Hu, and Min Zhang. 2024c. Temporal knowledge question answering via abstract reasoning induction. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4872–4889.

Ziyang Chen, Jinzhi Liao, and Xiang Zhao. 2023. Multi-granularity temporal question answering over knowledge graphs. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11378–11392.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, pages 4171–4186.

Wentao Ding, Hao Chen, Huayu Li, and Yuzhong Qu. 2022. Semantic framework based query generation for temporal question answering over knowledge graphs. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1867–1877.

Yifu Gao, Linbo Qiao, Zhigang Kan, Zhihua Wen, Yongquan He, and Dongsheng Li. 2024. Two-stage generative question answering on temporal knowledge graph using large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6719–6734.

Zhaoyan Gong, Juan Li, Zhiqiang Liu, Lei Liang, Huajun Chen, and Wen Zhang. 2025. Rtqa: Recursive thinking for complex temporal knowledge graph question answering with large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 9864–9881.

Qianyi Hu, Xinhui Tu, Guo Cong, and Shunping Zhang. 2025. Time-aware react agent for temporal knowledge graph question answering. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6013–6024.

Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. 2018. Tequila: Temporal question answering over knowledge bases. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 1807–1810.

Zhen Jia, Soumajit Pramanik, Rishiraj Saha Roy, and Gerhard Weikum. 2021. Complex temporal question answering on knowledge graphs. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 792–802.

[1] https://www.aclweb.org/portal/content/acl-code-ethics

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Yonghao Liu, Di Liang, Mengyu Li, Fausto Giunchiglia, Ximing Li, Sirui Wang, Wei Wu, Lan Huang, Xiaoyue Feng, and Renchu Guan. 2023. Local and global: Temporal question answering via information fusion. In *IJCAI*, pages 5141–5149.

Costas Mavromatis, Prasanna Lakkur Subramanyam, Vassilis N Ioannidis, Adesoji Adeshina, Phillip R Howard, Tetiana Grinberg, Nagib Hakim, and George Karypis. 2022. Tempoqr: temporal question reasoning over knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 5825–5833.

Xinying Qian, Ying Zhang, Yu Zhao, Baohang Zhou, Xuhui Sui, and Xiaojie Yuan. 2025. Plan of knowledge: Retrieval-augmented large language models for temporal knowledge graph question answering. *arXiv preprint arXiv:2511.04072*.

Xinying Qian, Ying Zhang, Yu Zhao, Baohang Zhou, Xuhui Sui, Li Zhang, and Kehui Song. 2024. Timer4: Time-aware retrieval-augmented large language models for temporal knowledge graph question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 6942–6952.

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.

V Sanh. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Proceedings of Thirty-third Conference on Neural Information Processing Systems (NIPS2019)*.

Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. 2021. Question answering over temporal knowledge graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6663–6676.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4498–4507.

Aditya Sharma, Apoorv Saxena, Chitrank Gupta, Mehran Kazemi, Partha Talukdar, and Soumen Chakrabarti. 2023. Twirgcn: Temporally weighted graph convolution for question answering over temporal knowledge graphs. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2049–2060.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

A Appendix

A.1 Controlled Diagnostic of Retrieval and Reasoning Failures

To better understand where TKGQA errors come from, we conduct a controlled diagnostic on **MultiTQ** that separates *retrieval failure* from *reasoning failure under retrieved evidence*. Specifically, we randomly sample questions and prompt **DeepSeek-V3-0324** with *query* + *retrieved facts* + *evidence* to answer temporal operators such as *before*, *after*, *first*, and *last*. We vary the retrieval budget as $K \in \{10, 20, 50, 100\}$, and for each query we check whether the retrieved set contains at least one gold-standard relevant fact.

As K increases, p_{miss} decreases substantially from 0.3271 to 0.1235, showing that a considerable portion of TKGQA errors is caused by retrieval ranking and recall limitations. At the same time, a_{hit} remains clearly below perfection even when gold-relevant evidence is already present in the retrieved set, which indicates that retrieval alone does not determine the final outcome and that temporal reasoning errors under available evidence remain a distinct source of failure. These results motivate TempReasoner from two aspects: the retrieval and filtering stages aim to surface temporally aligned evidence more reliably, while the validation stage is introduced because correct evidence does not always guarantee correct temporal judgment by the LLM.

Table 5: Controlled diagnostic on MultiTQ. p_{miss} denotes the probability that the top-K retrieved set contains no gold-standard relevant fact. a_{hit} denotes the answer accuracy conditioned on the retrieved set containing at least one gold-standard relevant fact.

	10	**20**	**50**	**100**
p_{miss}	0.3271	0.2530	0.1420	0.1235
a_{hit}	0.8532	0.8182	0.7770	0.7817

A.2 Comparison with Representative TKGQA Frameworks

To further clarify the positioning of TempReasoner with respect to the representative TKGQA frameworks, we provide a capability-level comparison in Table 6.

As shown in Table 6, the distinction of TempReasoner does not lie in staged design alone, but in the joint integration of question decomposition, structure-aligned rewriting, TKG-aware retrieval optimization, and evidence-grounded verification/correction within a single framework.

Table 6: Capability-level comparison with the representative TKGQA frameworks. Dcmp.: question decomposition; Rew.: query rewriting / optimization; TKG-Ret.: TKG-aware retrieval optimization; Ver.: verification / correction.

Capability	**TempReasoner**	MultiQA	TimeR4	RTQA
Dcmp.	✓	✗	✓	✓
Rew.	✓	✗	✓	✗
TKG-Ret.	✓	✗	✓	✗
Ver.	✓	✗	✗	✗

A.3 Algorithm of TempReasoner

The overall procedure of TempReasoner is summarized in Algorithm 1.

A.4 Preliminary

A Temporal Knowledge Graph (TKG) is defined as $\mathcal{G} = \{(s, r, o, t)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E} \times \mathcal{T}$, where $\mathcal{E}$ denotes the set of entities, $\mathcal{R}$ the set of relations, and $\mathcal{T}$ the set of timestamps. Each quadruple (s, r, o, t) represents a temporal fact, indicating that entity s is connected to entity o by relation r at time t.

Temporal Knowledge Graph Question Answering (TKGQA) aims to answer a temporal query Q by retrieving relevant facts from the TKG. Queries may range from simple one-hop questions, such as *"When did Indonesia want to make a visit to China?"*, to complex multi-step questions, such as *"Which country was the first that China wanted to cooperate with after Indonesia?"*. The answers may involve either a timestamp or an entity.

A.5 Module Architecture

We construct all modules following a general pattern. Each module consists of two components: an LLM processing component and a tool-calling component. The LLM processing component includes the task instruction, few-shot exemplars, input formatting, and output specification. The tool-calling component allows the LLM to invoke external tools depending on the functional requirement of the module. For example, the *Retrieve-Module* calls the retriever to obtain relevant temporal facts, while the *Filter-Module* invokes the filtering tool to refine the retrieved facts. Figure 5 presents representative prompt snippets from the main modules of TempReasoner. To keep the figure compact, repeated

Algorithm 1: TempReasoner: A Multi-Stage Framework for TKGQA

Input: Temporal Knowledge Graph $\mathcal{G}$, query Q
Output: Final answer A

Step 1: Query Decomposition;
Obtain instruction prompt $\mathcal{P}$;
$(q_1, q_2, \ldots, q_n)$, S = Decompose-Module$(Q, \mathcal{P})$;
// $S \in \{$`List-based`, `Tree-based`$\}$

Step 2: Sub-query Solving;
for $i = 1$ **to** n **do**
  // `(a) Temporal-aware retrieval`
  $\mathcal{R}_i$ = Retrieve-Module$(q_i, \mathcal{G})$;
  // `(b) Structure-guided filtering`
  $\mathcal{F}_i$ = Filter-Module$(q_i, \mathcal{R}_i)$;
  // `(c) LLM-based answer generation`
  $(a_i, \hat{f}_i)$ = Generate-Module$(q_i, \mathcal{F}_i)$;
  // a_i `= answer,` $\hat{f}_i$ `= supporting evidence set`
  // `(d) Optional validation`
  if *IsReliable*$(a_i, \hat{f}_i)$ **then**
    $\hat{a}_i \leftarrow a_i$;
  else
    $\hat{q}_i \leftarrow$ ConstructValidateQuery$(q_i, a_i, \hat{f}_i)$;
    $\hat{a}_i$ = Validate-Module$(\hat{q}_i)$;

Step 3: Answer Aggregation;
if S *is List-based* **then**
  A = SequentialCombine$(\hat{a}_1, \ldots, \hat{a}_n)$;
else
  A = ParallelMerge$(\{\hat{a}_i\}_{i=1}^{n})$;
return A;

examples, long schemas, and low-level formatting details are omitted with ellipses.

A.6 Semantic Similarity Dictionary via RQ-KMeans

To efficiently obtain semantically similar entities and relations in the TKG, we construct a similarity dictionary using a Residual Quantization K-Means (RQ-KMeans) framework. The procedure consists of three steps.

(1) Encoder-based representation. We first encode every entity and relation using a pretrained encoder $\mathcal{E}(\cdot)$. For an entity or relation x, its embedding is:

$$\mathbf{h}_x = \mathcal{E}(x) \in \mathbb{R}^d. \tag{5}$$

(2) Multi-level residual quantization. We apply RQ-KMeans to build a hierarchical codebook of size:

$$\ell_1 \times \ell_2 \times \ell_3,$$

where ℓ_i is the number of cluster centers at level i. For each level, residual quantization is performed as:

$$\begin{aligned}
\mathbf{r}_x^{(1)} &= \mathbf{h}_x, \\
c_x^{(1)} &= \arg\min_{c \in \mathcal{C}_1} \|\mathbf{r}_x^{(1)} - \mathbf{c}\|_2, \\
\mathbf{r}_x^{(2)} &= \mathbf{r}_x^{(1)} - \mathbf{c}_x^{(1)}, \\
c_x^{(2)} &= \arg\min_{c \in \mathcal{C}_2} \|\mathbf{r}_x^{(2)} - \mathbf{c}\|_2, \\
\mathbf{r}_x^{(3)} &= \mathbf{r}_x^{(2)} - \mathbf{c}_x^{(2)}, \\
c_x^{(3)} &= \arg\min_{c \in \mathcal{C}_3} \|\mathbf{r}_x^{(3)} - \mathbf{c}\|_2,
\end{aligned} \tag{6}$$

where $\mathcal{C}_i$ denotes the i-th level K-Means codebook. Thus each entity or relation x is assigned a *semantic code*:

$$\mathrm{sid}(x) = (c_x^{(1)}, c_x^{(2)}, c_x^{(3)}). \tag{7}$$

(3) Prefix-based semantic similarity. Two entities or relations x and y are considered semantically similar if their semantic codes share a prefix:

$$\mathrm{prefix}_k(\mathrm{sid}(x)) = \mathrm{prefix}_k(\mathrm{sid}(y)), \quad k \in \{1, 2, 3\}. \tag{8}$$

A larger shared prefix (e.g., first two or all three levels) implies stronger semantic closeness.

This RQ-KMeans–based construction enables efficient lookup of semantically similar entities and relations, providing a scalable and accurate mechanism for supporting the hard-negative construction used by our TKG-Retriever.

A.7 Details of Filter-Module

Figure 6 provides a concrete example of how the *Filter-Module* operates for the question *"Who visited Malaysia after 2007-11-15?"*. The retriever first returns the top 100 facts whose textual surface matches portions of the query. However, many of these results correspond to semantically unrelated relations such as *Make_statement* or entities that only contain the token "Malaysia" in their names (e.g., *Court_Judge_(Malaysia)*, *Police_(Malaysia)*), rather than actual "visiting" events. This illustrates a common failure mode of TKG retrieval: high lexical similarity but low semantic alignment.

To refine this noisy candidate set, the *Filter-Module* first extracts the top-k retrieved facts and presents them to the LLM, asking it to infer an alignment signal. This signal captures the essential structural constraints that any valid supporting fact should satisfy. In this example, the LLM determines that the question requires facts matching the pattern: *(head, Make_a_visit, Malaysia, time)*, where the head entity is unknown (`null`),

Decompose-Module

TASK: decide whether to decompose the input temporal-KG query.
If needed, choose a solver type.
SolverType: “List Solver” or “Tree Solver”.
Use #{i} and #{i}-day/month/year for dependencies.
Preserve entity and relation surface forms; ...
Output: solver type + sub-question list.

Retrieve-Module (Rewriter)

You are a Query Rewriter for Temporal-KG retrieval.
Produce a STRICT JSON with fields: head, relation, tail, begin_time, end_time, last_or_first.
Preserve named entities exactly as written.
Keep relation phrasing close to the question.
Canonicalize temporal expressions such as on / before / after / in; ...
Return ONLY the JSON object.

Filter-Module

ROLE: You are an Entity & Relation Selector for Temporal-KG alignment.
Get the alignment-signal from the given query and retrieval results.
Focus on semantic consistency with the query.
If no candidate fits confidently, return null for that slot.
Output: head + relation + tail.

Generate-Module

TASK: generate the final answer given a ranked CONTEXT of candidate facts.
Return the canonical answer and decisive proof lines.
Use CONTEXT only.
Apply first/last/before/after constraints strictly; ...
If no fact satisfies all constraints, return None.
Output: result + proof.

Validate-Module

Rethink step: verify whether the current answer truly satisfies the temporal intent.
Generate one verification query if needed; ...
Compare step: compare OLD and NEW answers with proofs,
then choose the better-supported result.
Output: type + reason + result.

Answer Aggregation

TASK: combine sub-question answers into the final answer.
Use only the given sub-answers.
Resolve dependencies such as #{i} and #{i}-day/month/year; ...
If required sub-answers are missing or inconsistent, return None.
Output: {"result": "..." }.

Figure 5: Representative prompt snippets used by the main modules of TempReasoner.

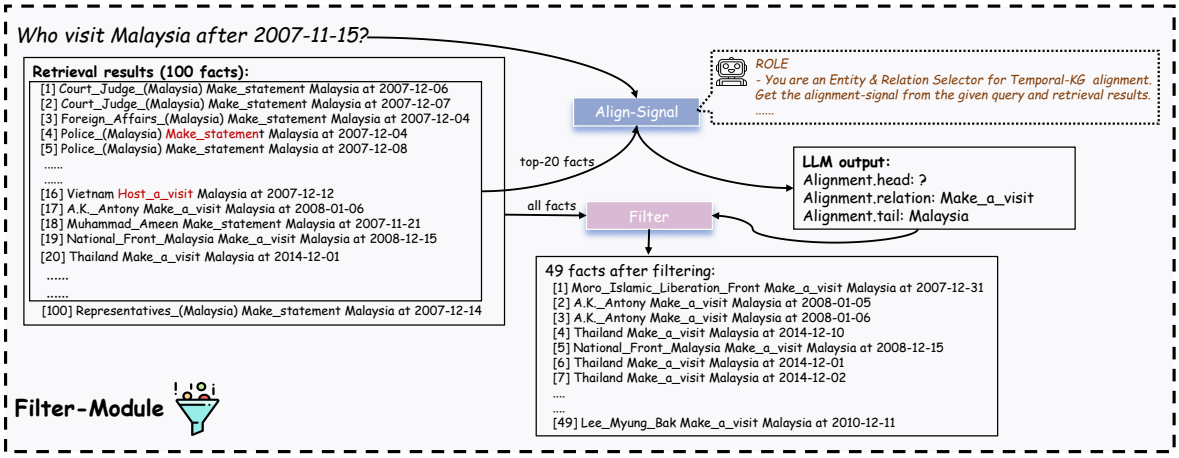

Figure 6: Illustration of *Filter-Module*.

the relation must correspond to the action "visit" (identified as *Make_a_visit*), and the tail must be *Malaysia*. This signal represents the LLM's distilled understanding of which entities and relations are semantically compatible with the question intent.

Once the alignment signal is obtained, the module applies it to *all* retrieved facts—including those beyond the top-k—to perform structured filtering. Any fact whose entity or relation does not match the inferred signal is removed. As shown in the figure, facts involving relations such as *Make_statement* or tail entities unrelated to Malaysia are eliminated.

After filtering, the candidate set is reduced from 100 facts to 49 high-quality facts, all of which correspond to genuine "visit Malaysia" events occurring after the specified date. This denoised evidence set preserves only semantically aligned temporal facts, enabling subsequent modules to reason over precise and relevant information rather than being distracted by irrelevant or misleading evidence.

This structured filtering process is crucial for mitigating semantic drift in retrieval and significantly improves the quality of evidence provided to downstream reasoning modules.

A.8 Details of Dataset

MultiTQ. MultiTQ is a large-scale temporal question answering dataset that incorporates multi-granularity temporal information. It provides a comprehensive evaluation protocol across several dimensions: Question Type (Multiple vs. Single), Answer Type (Entity vs. Time), and Time Granularity (year, month, and day). The detailed statistics of MultiTQ are presented in Table 7.

Type		Train #	Valid #	Test #
Single	Equal	135,890	18,983	17,311
	Before/After	75,340	11,655	11,073
	First/Last	72,252	11,097	10,480
Multiple	Equal Multi	16,893	3,213	3,207
	After First	43,305	6,499	6,266
	Before Last	43,107	6,532	6,247
Total		**386,787**	**57,979**	**54,584**

Table 7: Dataset Statistics of MultiTQ.

CronQuestions. CronQuestions is a large-scale temporal QA benchmark consisting of 410K unique question–answer pairs with annotated entities and timestamps. Questions in CronQuestions can be grouped into two major categories: simple temporal reasoning (e.g., Simple Entity and Simple Time) and complex temporal reasoning (e.g., Before/After, First/Last, and Time-Join queries), depending on the temporal constraints involved. The detailed statistics of CronQuestions are presented in Table 8.

Category	Train #	Valid #	Test #
Simple Entity	90,651	7,745	7,812
Simple Time	61,471	5,197	5,046
Before/After	23,869	1,982	2,151
First/Last	118,556	11,198	11,159
Time Join	55,453	3,878	3,832
Simple Reasoning	152,122	12,942	12,858
Complex Reasoning	197,878	17,058	17,142
Entity Answer	225,672	19,362	19,524
Time Answer	124,328	10,638	10,476
Total	**350,000**	**30,000**	**30,000**

Table 8: Dataset Statistics of CronQuestions.

A.9 Error Analysis

To better understand the remaining failure modes of TempReasoner, we further conduct an error analysis on failed cases and categorize them into four types: *decomposition error*, *alignment-signal error*, *retrieval error*, and *LLM generation error*. The results are summarized in Table 9. We annotate failed cases into four mutually exclusive categories according to the earliest stage that directly causes the final error. Decomposition error refers to cases where the sub-question structure is incorrect; alignment-signal error refers to cases where the induced structural constraints are incorrect; retrieval error refers to cases where relevant evidence is missed despite correct decomposition/alignment; and LLM generation error refers to cases where the final answer is wrong despite adequate upstream evidence.

The results show that the remaining failures are not dominated by decomposition alone. While decomposition errors account for 26.81% of failed cases, the largest category is alignment-signal error (32.33%), followed by LLM generation error (24.32%) and retrieval error (16.54%). This suggests that the main challenge is not only whether a question is decomposed correctly, but also whether the induced alignment constraints are sufficiently accurate to support downstream filtering and answer verification. More broadly, the distribution indicates that failures arise from multiple stages of the pipeline rather than from a single bottleneck.

Table 9: Error-type distribution on failed cases.

Error type	Proportion
Decomposition error	0.2681
Alignment-signal error	0.3233
Retrieval error	0.1654
LLM generation error	0.2432

Table 10: Type-wise API overhead on MultiTQ with and without validation.

Type	With val.	W/o val.
Avg.	7.42	6.09
equal	5.00	5.00
before_after	6.00	5.93
first_last	6.95	5.00
before_last	11.59	8.00
after_first	12.22	8.00
multi_equal	7.62	7.46

A.10 API Overhead Analysis

We further analyze the practical overhead on MultiTQ. Specifically, we report type-wise API-call statistics for TempReasoner with and without validation. The results are summarized in Table 10. We additionally find that the validation loop is triggered for 12.25% of queries on MultiTQ. These results show that validation acts as a selective second-pass control rather than an unconditional extra loop for every query. Overall, enabling validation increases the average number of API calls from 6.09 to 7.42, and the additional cost is mainly concentrated on harder temporal operators such as *before_last* and *after_first*. Notably, even without validation, harder operators such as *before_last* and *after_first* already require more API calls than simpler cases such as *equal*, indicating that a substantial part of the cost comes from the base multi-step inference pipeline itself rather than from validation alone.

A.11 Case Study

Step-by-step Visualization As illustrated in Figure 7, we take the temporal question *"Who was the first to visit Iraq after the Belgian Ministry?"* as an example and present the key intermediate outputs produced by each module. This step-by-step visualization clearly demonstrates how TempReasoner, through the coordinated interaction of its multi-stage modules, progressively identifies the relevant evidence and ultimately arrives at the correct answer.

Specifically, the first sub-question aims to identify the time at which the Belgian Ministry visited Iraq. The Retrieve-Module rewrites the query into a TKG-compatible form and retrieves multiple candidate facts, many of which involve semantically related but irrelevant entities. The Filter-Module then performs entity–relation alignment, retaining only facts consistent with the inferred alignment constraints (i.e., *Ministry_(Belgium)*, *Make_a_visit*, and *Iraq*). Based on the filtered evidence, the Generate-Module produces the answer *2009-06-04* together with its supporting fact. Since the retrieved evidence fully satisfies the temporal and relational constraints of the sub-question, the Validate-Module determines that no further verification is required and accepts the result as a reliable temporal anchor.

Using this inferred timestamp, the second sub-question is reformulated to identify entities that visited Iraq after *2009-06-04*. The retriever again collects candidate facts that satisfy the temporal constraint, including multiple visits occurring at different times. Through alignment-based filtering, the framework removes facts with incompatible relations or roles, leaving only valid visit events. The Generate-Module then selects *Chevron* as the earliest entity that visited Iraq after the anchor date, supported by the corresponding TKG fact.

At this stage, the Validate-Module constructs a stricter validation query by bounding the temporal interval between the inferred anchor date and the timestamp of the candidate answer. This query is re-evaluated through the same Retrieve–Filter–Generate pipeline to check whether an alternative entity satisfies the constraints more optimally. If a better-supported or earlier valid result were found during this verification step, the framework would replace the original answer accordingly. In this example, no such alternative is identified, and the initial answer *Chevron* is therefore selected as the final result. This case study demonstrates how TempReasoner incrementally refines evidence, enforces temporal consistency, and applies self-correction to ensure reliable temporal reasoning.

Question: Who was the first to visit Iraq after the Belgian Ministry?

==========

Decompose-Module

Decompose.Strategy: List Strategy

Decompose.Result:

SubQuestion #1: When did the Belgian Ministry visit Iraq?

SubQuestion #2: After #1-day, who was the first to visit Iraq?

==========Begin Solve SubQuesiton #1 ! ==========

Retrieve-Module

Rewrite.Result: the Belgian Ministry visit Iraq at TIME

TKG-Retriever.Result:

[1] Al_Bayan Make_a_visit Iraq at 2005-03-20

[2] Student_(Belgium) Make_a_visit Iraq at 2007-03-29

[3] Oman Make_a_visit Iraq at 2012-03-28

...

==========

Filter-Module

Alignment-Signal: Head: Ministry_(Belgium) Relation: Make_a_visit Tail: Iraq

Filter.Result:

[1] Ministry_(Belgium) Make_a_visit Iraq at 2009-06-04

==========

Generate-Module

Question: When did the Belgian Ministry visit Iraq?

Context:

[1] Ministry_(Belgium) Make_a_visit Iraq at 2009-06-04

Generate.Result: 20090604 Generate.Evidence: [1] Ministry_(Belgium) Make_a_visit Iraq at 2009-06-04

==========

Validate-Module

Validate-Query: No need to rethink or validate.

==========Begin Solve SubQuesiton #2 ! ==========

Retrieve-Module

Rewrite.Result: [20090605, -] WHO first visit Iraq after 20090604

TKG-Retriever.Result:

[1] Chevron Make_a_visit Iraq at 2009-06-17

[2] Barack_Obama Make_a_visit Iraq at 2009-07-03

[3] Nuri_al-Maliki Make_statement Iraq at 2009-06-16

...

==========

Filter-Module

Alignment-Signal: Head: null Relation: Make_a_visit Tail: Iraq

Filter.Result:

[1] Chevron Make_a_visit Iraq at 2009-06-17

[2] Barack_Obama Make_a_visit Iraq at 2009-07-03

...

==========

Generate-Module

Question: After 20090604, who was the first to visit Iraq?

Context:

[1] Chevron Make_a_visit Iraq at 2009-06-17

[2] Barack_Obama Make_a_visit Iraq at 2009-07-03

...

Generate.Result: Chevron Generate.Evidence: [1] Chevron Make_a_visit Iraq at 2009-06-17

==========

Validate-Module

Validate-Query: [20090605, 20090616] WHO first visited Iraq after 20090604

Validate-Result: No better answer, choose the raw answer.

==========

Final Answer: Chevron

==========

Figure 7: An example of TempReasoner to answer the temporal knowledge graph question.