# TempTool-R1: Tool-Augmented Reinforcement Learning for Temporal Knowledge Graph Question Answering

**Zicheng Huang**[1,2,3*], **Yajuan Tong**[1,2,3*], **Xinhui Tu**[1,2,3†], **Tingting He**[1,2,3†]
[1]Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, Wuhan, China
[2]School of Computer, Central China Normal University, Wuhan, China
[3]National Language Resources Monitor and Research Center for Network Media, Central China Normal University, Wuhan, China
{zichenghuang,tongauan}@mails.ccnu.edu.cn, tuxinhui@ccnu.edu.cn, tthe@mail.ccnu.edu.cn

## Abstract

Temporal knowledge graph question answering (TKGQA) addresses time-sensitive queries over temporal knowledge graphs, but existing approaches struggle with multi-hop reasoning and implicit temporal constraints. We introduce **TempTool-R1**, a novel tool-integrated reasoning framework that enables large language models to explicitly use temporal tools for precise reasoning. First, we design a unified temporal tool-based API capable of transforming implicit temporal cues into executable operations, establishing the structural foundation for tool interaction. In the second stage, supervised fine-tuning teaches the model to interweave chain-of-thought reasoning with think-then-tool usage, allowing it to call temporal tools during inference. Finally, we apply reinforcement learning with fine-grained, order-sensitive reward functions tailored for temporal tool use, further refining the model's tool-use policy. Experiments on three challenging TKGQA benchmarks demonstrate that TempTool-R1 significantly outperforms existing methods. In particular, our approach excels on complex questions requiring multi-hop temporal reasoning, highlighting the effectiveness of temporal tool integration and reward optimization in improving TKGQA performance.

## 1 Introduction

Temporal knowledge graph question answering (TKGQA) aims to answer time-sensitive questions by reasoning over temporal knowledge graphs (TKGs), where facts are represented as time-stamped quadruples $(s, p, o, t)$. In practice, questions can involve not only explicit temporal constraints (e.g., *"in 2010"*, *"after 2015"*) but also implicit ones (e.g., *"after the Danish Ministry"*). Moreover, TKGQA often requires multi-hop reasoning across facts and reasoning with different temporal operations (Chen et al., 2023b; Sun et al., 2025). These challenges demand models that can understand temporal expressions and perform precise temporal reasoning.


Figure 1: An illustrative example of complex temporal question with Tool-Integrated Reasoning.

Recently, large language models (LLMs) have become the dominant paradigm for TKGQA (Chen et al., 2024b; Hu et al., 2025; Gong et al., 2025). However, existing LLM-based methods typically rely on carefully crafted prompts, and they still struggle with multi-hop temporal queries.

Reinforcement learning (RL) has shown great potential for improving the reasoning capabilities of LLMs. For example, models such as OpenAI-o1 (OpenAI et al., 2024) and DeepSeek-R1 (DeepSeek-AI et al., 2025) demonstrate that RL can significantly boost performance on complex reasoning tasks. Meanwhile, an increasingly important direction is **Tool-Integrated Reasoning** (TIR), where models invoke external tools during the reasoning process. Recent work (Jin et al., 2025; Tan et al., 2025; Qian et al., 2025) shows that, for complex reasoning tasks, LLMs often cannot answer correctly using internal knowledge alone and need to interact with external resources such as search engines, code interpreters, or even TKGs. Building on these insights, we hypothesize that equipping an LLM with a set of temporal tools to interact with

*Equal contribution.
†Corresponding authors.



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 16224–16241

TKGs and training it with RL can enable more effective temporal reasoning. Figure 1 illustrates an example where a temporal question is solved through tool use. Notably, although prior TIR approaches have enhanced tool use and multi-step reasoning, none have directly tackled the unique challenges of temporal reasoning on TKGs.

We identify two key limitations in applying TIR to TKGQA: **(1) Lack of unified and reusable temporal tool interfaces.** Some prior work (Gong et al., 2025; Qian et al., 2024) exposes temporal information through question decomposition or retrieval, but the underlying temporal operations remain implicitly encoded in prompt or LLMs' reasoning. ARI (Chen et al., 2024b) defines fine-grained actions, but these actions are not designed for interacting with TKGs and are partially redundant. As a result, models struggle to generalize to new query types or compositional temporal constraints, and they lack reusable temporal operators. **(2) Lack of fine-grained reward design for temporal tool.** Existing tool-using RL approaches (Li et al., 2025; Tan et al., 2025) typically provide only coarse outcome-based rewards, without evaluating whether intermediate tool usage is correct. Although ToolRL (Qian et al., 2025) highlights the importance of fine-grained rewards for tool use, it mainly matches tool names, arguments, without capturing the order of tool calls or uses coarse matching for parameter values, which are crucial for TKGQA.

To address these limitations, we propose **TempTool-R1**, a Tool-Integrated Reasoning (TIR) training framework for TKGQA that enables LLMs to explicitly use various temporal tools during reasoning. We train LLMs to interact with TKGs via tools, enabling step-by-step reasoning over complex temporal questions. Specifically, we first design a unified temporal tool-based API for TKGQA and construct a tool-augmented reasoning dataset that pairs chain-of-thought reasoning with executable temporal tool calls. Second, we perform supervised fine-tuning (SFT) to equip LLMs with the ability to leverage temporal tools during reasoning and follow a think-then-tool format. Finally, we introduce a fine-grained reward tailored to temporal tool and apply Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to refine the model's tool-use policy on complex temporal questions. Experiments on three TKGQA benchmarks show that our approach outperforms existing methods, especially on questions requiring multi-hop temporal reasoning. The contributions of this work are summarized as follows:

- We design dataset-agnostic temporal tool-based API for TKGQA that turns implicit temporal reasoning into executable tool programs, enabling reusable temporal operators and interpretable reasoning.
- We propose **TempTool-R1**, a novel training framework for TKGQA that equips LLMs with the ability to integrate temporal tools into the reasoning process and strengthens their temporal inference.
- We design fine-grained, order-sensitive reward functions tailored for temporal tool-integrated reasoning and conduct extensive RL experiments to validate their effectiveness. Although task-specific, the underlying principle—encoding tool dependencies in the reward signal rather than in the tool interface or policy architecture—is general and applicable to other domains where correct tool usage depends on logical ordering.

## 2 Related Work

### 2.1 TKGQA

Existing TKGQA approaches fall into three groups: embedding-based, semantic parsing-based, and LLM-based methods.

**Embedding-based Methods** CronKGQA (Saxena et al., 2021) derives time embeddings of entities and relations from pre-trained models. TempoQR (Mavromatis et al., 2022) incorporates contextual and time-aware modules into the question. MultiQA (Chen et al., 2023b) introduces multi-granularity temporal constraints. Some methods (Jia et al., 2021; Sharma et al., 2023; Liu et al., 2023) integrate graph neural networks. These methods offer high execution efficiency but struggle with complex temporal tasks.

**Semantic Parsing-based methods** These approaches translate questions into structured queries (e.g., logical forms, programs) that can be executed over TKGs (Ding et al., 2022; Chen et al., 2024a). Such methods can achieve high precision when the parsing is correct. However, they often suffer from parsing errors and inaccurate entity or relation extraction, which leads to unstable behavior and degraded overall performance.

**LLM-based Methods** With the emergent abilities of LLMs, LLM-based methods have become the dominant paradigm. ARI (Chen et al., 2024b) enhances LLMs through inductive reasoning, GenTKGQA (Liao et al., 2024) adopts a two-stage generation strategy, TimeR$^4$ (Qian et al., 2024) introduces a retrieve-rewrite-rerank strategy, and RTQA (Gong et al., 2025) iteratively decomposes complex temporal queries. Unlike existing methods, our approach introduces a unified API for temporal tools and utilizes SFT and RL to strengthen LLMs' proficiency in tool usage, thereby enabling more effective complex reasoning in TKGQA.

### 2.2 Tool-Integrated Reasoning of LLMs

Tool-integrated reasoning (TIR) has emerged as a promising direction for enhancing the capabilities of LLMs. Early studies (Schick et al., 2023; Qin et al., 2024a) introduced the idea of equipping LLMs with external tools to compensate for limitations inherent to pretraining, such as code executors (Chen et al., 2023a) and search engines (Vu et al., 2024). Subsequent work (Schick et al., 2023; Qin et al., 2024b) relied on SFT with carefully curated datasets to improve tool-use performance. More recently, reinforcement learning (RL) has been recognized as an effective approach for further advancing TIR, demonstrating strong potential in information retrieval (Jin et al., 2025), research (Zheng et al., 2025), mathematics (Li et al., 2025), and code generation (Feng et al., 2025). In parallel, several studies have explored fine-grained rewards for tool use (Qian et al., 2025) and multi-step TIR (Xue et al., 2025). However, none of these approaches have directly addressed temporal reasoning over TKGs. Our work is the first to bridge this gap, by defining unified temporal tools and a dedicated training framework to integrate them into LLM reasoning for TKGQA.

## 3 Method

### 3.1 Task Definition

Given a temporal question $q$ and a Temporal Knowledge Graph (TKG) $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T}, \mathcal{F})$, TKGQA aims to provide the correct answer based on TKG, where the answer may be an entity $e_s/e_o \in \mathcal{E}$, a timestamp $t \in \mathcal{T}$, or a Boolean value.

### 3.2 Overview

TempTool-R1 is a reasoning-driven, tool-integrated framework that trains LLMs to invoke temporal tools appropriately when interacting with the TKG, leading to more accurate answers. Concretely, it consists of three stages: (1) We decompose complex temporal questions into recurring reasoning patterns and design a set of generalizable temporal tools to handle these patterns. (2) We carefully construct training examples that interleave chain-of-thought reasoning with temporal tool calls. Based on these data, we perform supervised fine-tuning (SFT) to teach the LLM to generate responses in a think-then-tool format. (3) Starting from the SFT model, we further apply reinforcement learning with order-sensitive and fine-grained rewards at the entity and relation level to stabilize training. These rewards improve the model's ability to handle complex questions. The overall framework is illustrated in Figure 2.

### 3.3 Definition of Temporal Tools

TempTool-R1 formulates TKGQA as a multi-turn interaction with a set of temporal tools. We prioritize two design principles. First, tool interfaces are decoupled from the underlying TKG schema to ensure portability across different temporal knowledge graphs. Second, the internal implementation of each tool is designed to maximize the recall of correct candidates, so that the result is not constrained by the retrieval accuracy of temporal tools.

**Tool Interface.** Following (Sun et al., 2025), we group temporal reasoning into temporal retrieval capabilities (temporal constrained retrieval (TCR), timeline position retrieval (TPR)), temporal operation capabilities (temporal semantic operation (TSO), and timeline arithmetic operation (TAO)). In line with our goal that temporal tools should directly interact with the TKG, we focus on TCR and TPR, while leaving TSO and TAO to be handled by the model's CoT reasoning. We therefore construct three temporal tools, as shown in Table 1, which is sufficient to realize the retrieval primitives needed by common multi-hop temporal reasoning. Specifically, at step $i$, the model selects an appropriate tool $T_i$ based on the context and receives a tool feedback $o_i$. Then it uses the updated state to choose the next tool $T_{i+1}$, and continues this process until a final answer is produced. We denote the reasoning trajectory up to step $k$ as $s_k = \{(r_i, T_i, o_i) \mid i = 1, \ldots, k\}$.

**Tool Implementation.** The objective of our tool implementation is to improve the accuracy of retrieved results. To this end, we introduce two key

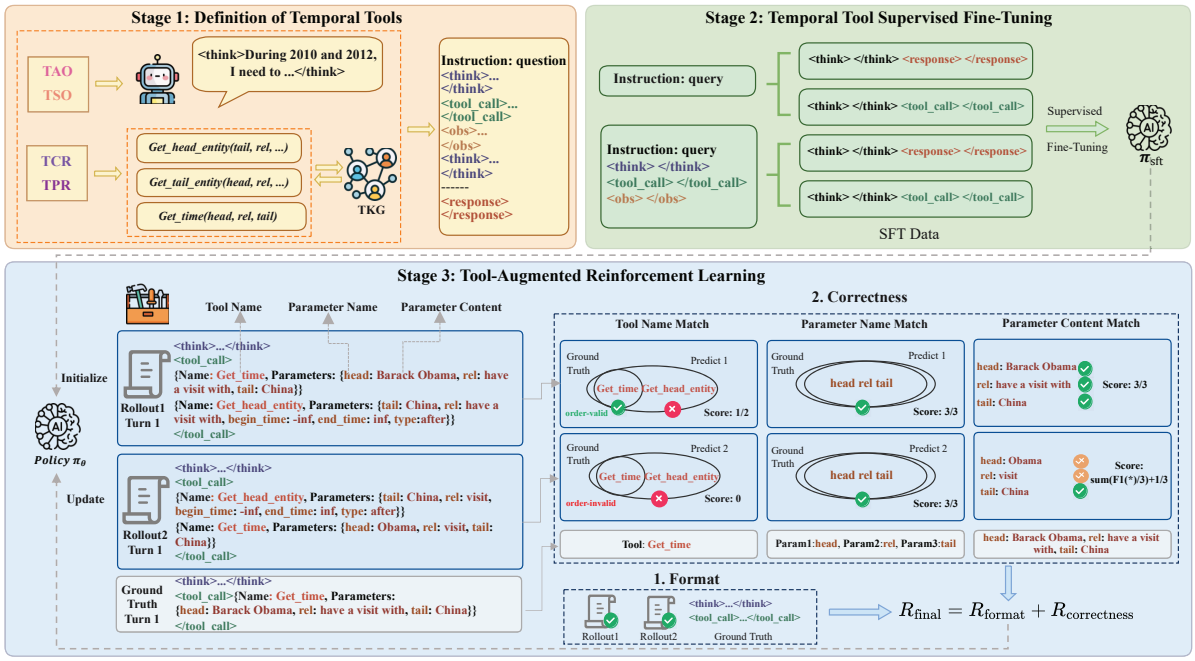


Figure 2: Overview of TempTool-R1. In Stage 1, we define a unified set of temporal tools. In Stage 2, we perform cold-start SFT, and in Stage 3, we apply RL with carefully designed temporal tool rewards to further improve the model's performance on TKGQA.

components: semantic alignment for entities and relations, and temporal filtering. Concretely, once the model outputs a tool call, we parse its entity and relation arguments and compute their semantic similarity to predefined ones in TKG. We select the top-$K$ candidates, rerank them, and take the top 1 as the final aligned entity or relation, which is then used for exact matching in the TKG. The temporal filtering module extracts temporal arguments and constraint types from the tool interface and filters over the TKG, effectively narrowing the search space and improving retrieval precision. We adopt top-1 alignment to ensure deterministic and stable tool execution during RL training. Importantly, the gains of TempTool-R1 cannot be attributed to this retrieval infrastructure alone: as shown in Table 4, equipping the untrained Qwen3-4B with the same tools and identical retrieval pipeline yields only 0.543 Hits@1, compared to 0.851 for TempTool-R1. Since the retrieval module is held fixed across these two settings, the performance gap reflects the reasoning and tool-use policies learned through SFT and RL, rather than retrieval strength.

### 3.4 Temporal Tool Supervised Fine-Tuning

To equip the LLM with the ability to adaptively use temporal tools in a think-then-tool format during reasoning, we first construct tool-augmented reasoning samples. Specifically, the system prompt instructs LLM, upon receiving a new query, to first perform reasoning within <think> and </think> tags. After completing reasoning, the LLM may invoke temporal tools as needed by emitting a tool specification inside <tool_call> and </tool_call>, while the returned results from the tools are inserted into the context within <obs> and </obs>. This process is repeated iteratively: each new observation can trigger further thinking and additional tool calls, until either LLM produces a final answer enclosed in <response> and </response> or a predefined maximum number of tool invocations is reached.

After generation, we filter out samples with correct answers and partition them into four categories (Tan et al., 2025), as illustrated in Figure 2. The first two categories teach the model to answer using its internal knowledge, while the last two encourage it to continue reasoning from temporal tool outputs. We then perform cold-start SFT to teach the model to recognize tools, respond in a think-then-tool format, and further interact with the TKG based on tool feedback, providing a stronger initialization for subsequent RL training.

### 3.5 Reward Design

Rule-based reward mechanisms have demonstrated strong performance and are widely used. Following prior work, we adopt a combination of format

| Tool | Parameter | Description |
|---|---|---|
| *Get_time* | (*head, rel, tail*) | Returns the time interval(s) during which the relation *rel* between *head* and *tail* holds. |
| *Get_head_entity* | (*tail, rel, begin_time, end_time, type*) | Finds the head entity that satisfies the given relation with the known tail within *[begin_time, end_time]*. |
| *Get_tail_entity* | (*head, rel, begin_time, end_time, type*) | Finds the tail entity that satisfies the given relation with the known head within *[begin_time, end_time]*. |

Table 1: Summary of the three temporal tools used for interacting with the temporal knowledge graph. The parameter *type* specifies whether the temporal condition is *"in/on"*, *"before"*, *"after"*, or *"between"*. The *Get_time* tool equips the model with TPR capability, while *Get_head_entity* and *Get_tail_entity* provide TCR capability.

and correctness rewards. Formally, the final reward $R_{\text{final}}$ is composed of $R_{\text{format}}$ and $R_{\text{correct}}$, each described in detail below:

**Format Reward.** The format reward $R_{\text{format}} \in \{0, 1\}$ checks whether the model output contains all required special tokens (thoughts, tool calls, and responses) in the correct order specified by the reference trajectory:

$$R_{\text{format}} = \begin{cases} 1, & \text{all format tags are correct,} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

**Correctness Reward.** Conventional correctness rewards typically focus only on the final answer. However, TIR involves multi-turn interactions where each step may invoke multiple tools with structured arguments, and thus requires finer-grained rewards to effectively support RL training. ToolRL (Qian et al., 2025) takes an important step in this direction, but it does not explicitly model the ordering of tools within the same turn. For example, in answering "*After the Danish Ministry of Defence and Security, who was the first to visit Iraq?*", a gold trajectory first calls *Get_time(head: the Danish Ministry ..., rel: visit, tail: Iraq)* and then calls *Get_head_entity(tail: Iraq, rel: visit, ...)* after obtaining the timestamp. During RL sampling, issuing both tools in a single turn with *Get_time* before *Get_head_entity* still respects the temporal logic and should receive partial credit, whereas placing *Get_head_entity* before *Get_time* violates the temporal dependency. Yet both cases obtain the same score in ToolRL. This motivates an order-sensitive correctness reward to guide more faithful temporal tool use.

In addition, ToolRL uses exact matching for parameter content, which is overly strict for tasks that extract entities and relations from natural language. We instead adopt a word-level unigram F1 reward: we split the predicted and gold parameter strings into tokens and compute F1 over their word overlap, providing a finer signal for parameter-content matching.

Given predicted tool calls $P = \{P_i\}_{i=1}^{m}$ and ground-truth calls $G = \{G_i\}_{i=1}^{n}$, the detailed formulation is given as follows:

- *Tool Name Matching:*

$$r_{\text{name}} = \begin{cases} \dfrac{|N_G \cap N_P|}{|N_G \cup N_P|}, & \text{if order is valid} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where $N_G$ and $N_P$ denote the tool-name sets from the gold and predicted calls, respectively. As noted by (Sun et al., 2025), TPR must precede TCR. Thus, if *Get_time* appears before *Get_head_entity* or *Get_tail_entity*, we compute the Jaccard similarity, otherwise the reward is set to 0.

- *Parameter Name Matching:*

$$r_{\text{param}} = \sum_{G_j \in G} \frac{\left|\text{keys}(P_G) \cap \text{keys}(P_P)\right|}{\left|\text{keys}(P_G) \cup \text{keys}(P_P)\right|}, \quad (3)$$

where $\text{keys}(P_G)$ and $\text{keys}(P_P)$ denote the parameter names of the gold and predicted tool calls, respectively.

- *Parameter Content Matching:*

$$r_{\text{value}} = \sum_{G_j \in G} \sum_{k \in \text{keys}(G_j)} \text{F1}\big(P_G[k], P_P[k]\big), \quad (4)$$

where

$$\text{F1}(x, y) = \frac{2\,|W(x) \cap W(y)|}{|W(x)| + |W(y)|}, \quad (5)$$

and $W(x)$ denotes the multiset of words obtained by splitting string $x$ on whitespace. We apply unigram F1 matching only to the argument values of entities and relations, while keeping other arguments (e.g., time and type) under exact matching.

- *Total match score for each match is:*

$$r_{\text{match}} = r_{\text{name}} + r_{\text{param}} + r_{\text{value}} \in [0, S_{\max}], \quad (6)$$

where $S_{\max} = 1 + |G| + \sum_{G_j \in G} |\text{keys}(G_j)|$ denotes the maximum possible score. All other components are kept the same as in ToolRL. This finer-grained reward design enables the model to better capture order-sensitive features of temporal tool use and more completely extract entities and relations, which is crucial for TKGQA.

### 3.6 Tool-Augmented Reinforcement Learning

After obtaining the cold-start SFT model, we further perform Temporal Tool Reinforcement Learning (TempToolRL) based on this checkpoint. Specifically, we select challenging examples that it answers incorrectly and then filter out cases that are essentially unanswerable due to incomplete tool retrieval or inherent model limitations. The remaining examples are used to construct the RL training set. We then apply the carefully designed reward functions from the previous section and employ GRPO (Shao et al., 2024) as our RL algorithm. The optimization objective of GRPO is given by:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\, \mathbb{E}_{o_i\sim\pi_\theta}\Big[\min\Big(\frac{\pi_\theta(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)} A_i, \text{clip}\Big(\frac{\pi_\theta(o_i \mid q)}{\pi_{\text{old}}(o_i \mid q)}, 1-\varepsilon, 1+\varepsilon\Big) A_i\Big) - \beta D_{\text{KL}}\Big], \quad (7)$$

where $A_i = \frac{R_i - \text{mean}(R)}{\text{std}(R)}$ represents the normalized advantage of the i-th output within the current group, calculated using our temporal tool reward function.

## 4 Experiment

### 4.1 Experimental Setup

**Datasets.** We evaluate TempTool-R1 on three challenging TKGQA benchmarks: **MULTITQ** (Chen et al., 2023b), **TIMELINECRONQUESTIONS**, and **TIMELINEICEWS** (Sun et al., 2025). These three datasets are highly challenging due to their multi-hop reasoning and complex temporal constraints. Further details about the datasets are provided in the Appendix A.

**Baselines.** For MULTITQ, we compare against three types of baselines: (1) **Pre-trained LMs**, including BERT (Devlin et al., 2019) and ALBERT (Lan et al., 2020). (2) **Embedding-based methods**, including EmbedKGQA (Saxena et al., 2020), CronKGQA (Saxena et al., 2021), and MultiQA (Chen et al., 2023b). (3) **LLM-based-methods**, including ChatGPT, ARI (Chen et al., 2024b), TempAgent (Hu et al., 2025), TimeR$^4$ (Qian et al., 2024), RTQA (Gong et al., 2025), and ProgTQA (Chen et al., 2024a). For TIMELINECRONQUESTIONS and TIMELINEICEWS, we adopt RAG and RTQA as baselines, since few methods currently support these datasets due to the difficulty and novelty.

**Evaluation Metric.** We report Hits@1 as the primary evaluation metric, following prior work on these datasets (Chen et al., 2024b; Gong et al., 2025; Qian et al., 2024; Chen et al., 2024a). Each question in these benchmarks is associated with a single canonical answer (an entity, a timestamp, a time interval, or a boolean value), so set-based metrics such as Precision, Recall, and F1 are not applicable. We use Hits@1 rather than Hits@10 because Hits@10 can overestimate performance in tool-based reasoning, where a model may obtain the correct answer even when the underlying temporal reasoning or tool usage is incorrect. Hits@1 more strictly reflects correct tool selection, parameterization, and temporal dependency handling. Since model outputs are in free text and may differ in surface form from gold answers (e.g., aliases or partial time expressions), we employ an LLM-based judge for normalization; exact matching is applied whenever outputs are already in canonical form. Further details on evaluation are provided in Appendix C.

**Implementation Details.** We use Qwen3-4B as the backbone model for TempTool-R1. All experiments are conducted on a server equipped with NVIDIA RTX PRO 6000 Blackwell Workstation Edition GPUs. More implementation details can be found in Appendix C.

### 4.2 Main Results

We compare TempTool-R1 against several strong baselines. Tables 2 and 3 report results on MULTITQ and TimelineKGQA, respectively, where the best scores are highlighted in bold and the second-best are underlined. TempTool-R1 achieves the highest performance across all datasets.

On the MULTITQ dataset, TempTool-R1 achieves the best performance across all question types, with an overall improvement of about 11%. Specifically, LMs (BERT, ALBERT, ChatGPT) perform the worst due to their lack of temporal knowledge. Embedding-based models show clear limitations on complex questions. LLM-based methods (ARI, TempAgent, TimeR$^4$, RTQA) yield improvements overall, yet still struggle with complex rea-

| Model | Overall | Question Type | | Answer Type | |
|---|---|---|---|---|---|
| | | Simple | Complex | Entity | Time |
| BERT | 0.083 | 0.092 | 0.061 | 0.101 | 0.040 |
| ALBERT | 0.108 | 0.116 | 0.086 | 0.139 | 0.032 |
| EmbedKGQA | 0.206 | 0.235 | 0.134 | 0.290 | 0.001 |
| CronKGQA | 0.279 | 0.134 | 0.134 | 0.328 | 0.156 |
| MultiQA | 0.293 | 0.347 | 0.159 | 0.349 | 0.157 |
| ChatGPT | 0.113 | 0.149 | 0.085 | 0.151 | 0.004 |
| ARI | 0.380 | 0.680 | 0.210 | 0.394 | 0.344 |
| TempAgent | 0.702 | 0.857 | 0.316 | 0.624 | 0.870 |
| TimeR$^4$ | 0.728 | 0.887 | 0.335 | 0.639 | 0.945 |
| RTQA | 0.765 | 0.902 | 0.424 | 0.692 | 0.942 |
| ProgTQA | 0.797 | 0.817 | 0.750 | 0.790 | 0.815 |
| **TempTool-R1** | **0.851** | **0.952** | **0.753** | **0.817** | **0.947** |

Table 2: Hits@1 performance comparison on the MULTITQ dataset.

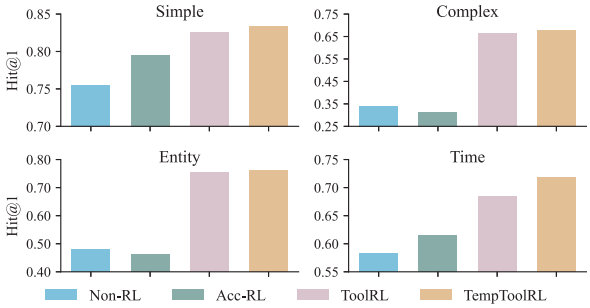


Figure 3: Comparison of different reward strategies on the MULTITQ.

soning. ProgTQA is a semantic parsing method that performs well on complex questions but, because it depends on correct logical query syntax, underperforms RTQA on simple ones. In contrast, TempTool-R1, which employs tool-augmented reinforcement learning for TKGQA, achieves the best performance across all question types. It demonstrates the effectiveness of our strategy.

On the TIMELINECRONQUESTIONS and TIMELINEICEWS, TempTool-R1 also achieves the best performance across all question categories, with overall scores of 0.517 and 0.475, respectively. In particular, for medium and complex questions, our method substantially outperforms RTQA and RAG, which further demonstrates the effectiveness of our strategy on multi-hop temporal reasoning.

### 4.3 Ablation Study

TempTool-R1 is a three-stage framework consisting of temporal tool definition, cold-start SFT, and Temporal Tool Reinforcement Learning (TempToolRL). To verify the contribution of each component, we conduct ablation studies. The results are shown in Table 4.

**Impact of Temporal Tool.** To assess the effectiveness of our temporal tool design for TKGQA, we equip the Qwen3-4B model with the defined temporal tools, denoted as *w/ TempTool*. As shown in Table 4, Qwen3-4B *w/ TempTool* clearly outperforms the base model that relies only on its internal knowledge. This indicates that many temporal facts are missing from the backbone model and must be accessed via interactions with the TKG, which in turn confirms the usefulness of our temporal tool interface.

**Impact of SFT.** To examine the necessity of SFT, we fine-tune Qwen3-4B on the tool-integrated reasoning dataset, denoted as *w/ TempTool-SFT*. Compared to the untuned setting with tools enabled (TempTool), the overall performance improves by 44.8%, with particularly strong gains on simple question types and time-based answers. This demonstrates that SFT is an effective and necessary step. It helps the model quickly familiarize itself with the tool interface and output format, and learn basic strategies for solving simple temporal questions. However, its ability to handle more complex questions still requires further improvement.

**Impact of RL.** To further validate the necessity of Tool-Augmented Reinforcement Learning, we perform RL on both the Qwen3-4B backbone and the TempTool-SFT model, obtaining the TempTool-RL and TempTool-R1 variants, respectively. The results show that TempTool-RL achieves slightly lower *overall* performance than the SFT model; however, most of this gap is concentrated on simple questions, while the RL model outperforms SFT on multi-hop temporal questions (Complex: 0.681 vs. 0.662). This suggests that RL helps the model explore better policies and is particularly beneficial for complex temporal reasoning. In particular, models often reach correct answers while violating temporal constraints, such as performing temporal filtering or entity retrieval before resolving the relevant timestamp. These errors are indistinguishable under answer-level supervision and cannot be reliably eliminated by SFT. Our order-sensitive RL reward explicitly models these temporal dependencies between tool calls, correcting such systematic reasoning errors that persist after SFT. Moreover, TempTool-R1 consistently outperforms both the SFT-only and RL-only settings, indicating that

| | TimelineCronQuestions | | | | TimelineICEWS | | | |
|---|---|---|---|---|---|---|---|---|
| | **Overall** | **Simple** | **Medium** | **Complex** | **Overall** | **Simple** | **Medium** | **Complex** |
| RAG | 0.235 | 0.704 | 0.092 | 0.009 | 0.265 | 0.660 | 0.128 | 0.011 |
| RTQA | 0.298 | 0.608 | 0.218 | 0.135 | – | – | – | – |
| **TempTool-R1** | **0.517** | **0.809** | **0.426** | **0.495** | **0.475** | **0.761** | **0.402** | **0.453** |

Table 3: Hits@1 comparison on TimelineCronQuestions and TimelineICEWS.

| **Model** | **Overall** | **Question Type** | | **Answer Type** | |
|---|---|---|---|---|---|
| | | **Simple** | **Complex** | **Entity** | **Time** |
| Qwen3-4B | 0.076 | 0.102 | 0.003 | 0.115 | 0.002 |
| + w/ TempTool | 0.543 | 0.755 | 0.342 | 0.478 | 0.583 |
| + w/ TempTool-SFT | 0.786 | 0.912 | 0.662 | 0.692 | 0.942 |
| + w/ TempTool-RL | 0.750 | 0.833 | 0.681 | 0.763 | 0.719 |
| **+ TempTool-R1** | **0.851** | **0.952** | **0.753** | **0.817** | **0.947** |

Table 4: Ablation study of TempTool-R1 on MultiTQ.

| **Model** | **Overall** | **Question Type** | | **Answer Type** | |
|---|---|---|---|---|---|
| | | **Simple** | **Complex** | **Entity** | **Time** |
| Llama3.2-3B | 0.054 | 0.080 | 0.003 | 0.855 | 0.005 |
| + TempTool | 0.459 | 0.615 | 0.216 | 0.386 | 0.515 |
| + TempTool-R1 | 0.816 | 0.937 | 0.695 | 0.756 | 0.873 |
| Qwen3-0.6B | 0.005 | 0.010 | 0.000 | 0.008 | 0.001 |
| + TempTool | 0.017 | 0.025 | 0.008 | 0.020 | 0.015 |
| + TempTool-R1 | 0.737 | 0.906 | 0.563 | 0.675 | 0.837 |
| Qwen3-1.7B | 0.032 | 0.050 | 0.001 | 0.056 | 0.002 |
| + TempTool | 0.145 | 0.250 | 0.050 | 0.095 | 0.168 |
| + TempTool-R1 | 0.786 | 0.926 | 0.617 | 0.698 | 0.875 |
| Qwen3-4B | 0.076 | 0.102 | 0.003 | 0.115 | 0.002 |
| + TempTool | 0.543 | 0.755 | 0.342 | 0.478 | 0.583 |
| **+ TempTool-R1** | **0.851** | **0.952** | **0.753** | **0.817** | **0.947** |

Table 5: Hits@1 on the MultiTQ dataset across different model families and parameter scales.

each stage of our three-stage framework contributes meaningfully to the final performance.

### 4.4 Further Experimental Analysis

**Generalization across Model Families and Scales.** To further assess the generality of our approach across model families and parameter scales, we conduct additional experiments on MultiTQ using the Qwen3 and Llama3.2 model series. For the Qwen3 family, we consider models with 4B, 1.7B, and 0.6B parameters, and for Llama3.2 we use the 3B variant, as shown in Table 5. The results show that even when switching the backbone to Llama3.2-3B, applying the same training still yields substantial gains over prior methods. Similarly, across different sizes of Qwen3, both the 0.6B and 1.7B models achieve competitive performance.

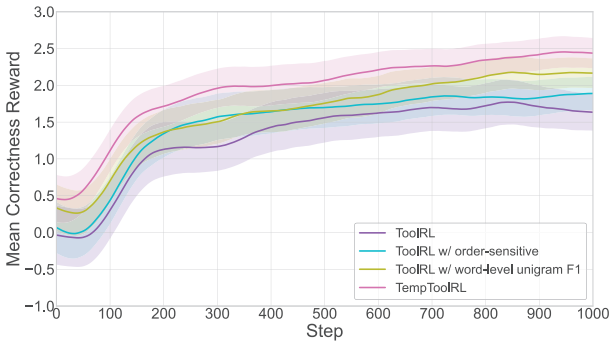


Figure 4: Correctness reward trends across training steps for Qwen3-4B with different reward design.

It shows that TempTool-R1 generalizes well across both model families and parameter scales. Notably, the backbone models alone perform poorly on MultiTQ, suggesting that they lack sufficient temporal knowledge and struggle with complex temporal reasoning. Moreover, larger models interact more effectively with temporal tools, which is consistent with scaling law behavior.

**Analysis of Reward Design.** To investigate the effectiveness of our reward design, we train Qwen3-4B with three reward schemes: the vanilla accuracy reward, the tool correctness reward from ToolRL, and our temporal tool reward (TempToolRL). As illustrated in Figure 3, the vanilla accuracy reward performs the worst. In Tool-Integrated Reasoning, the model interacts with tools over multiple turns, making answer-only rewards sparse and insufficient to capture partially correct intermediate steps. Moreover, errors may arise from tool outputs rather than incorrect tool choices, which further limits the reliability of answer-level supervision. As a result, answer-only rewards are too coarse to effectively guide temporal tool use. In contrast, ToolRL assigns fine-grained rewards to tool calls, improving the correctness of intermediate reasoning. However, it does not account for the ordering of tools within the same turn and relies on exact matching for parameter values, which limits its performance. As illustrated in Figure 4, we compare correctness

reward trends under different designs during training. Two key improvements over ToolRL are the order-sensitive constraint and the word-level unigram F1 metric. Both yield higher reward values, with the latter providing larger gains. Overall, by explicitly modeling ordering constraints and adopting word-level unigram F1 for parameter matching, our temporal tool reward produces more informative supervision and leads to better performance.

### 4.5 Case Study

**Why does TempTool-R1 lead to significant performance improvements?** To answer this question, we analyze cases selected from the MULTITQ test set, as shown in Appendix Table 6. In these cases, we observe that TempTool-R1 adaptively selects appropriate tools and invokes them over multiple steps, progressively refining its reasoning rather than attempting to produce the final answer in a single shot. The multi-step tool use stands in contrast to approaches that rely solely on question decomposition or handcrafted plans. It further exhibits the strong capability of our method in handling complex questions.

We also find that the fine-tuned model's outputs better match the evaluation targets. For instance, when the gold answer is a full timestamp, the base LLM often produces only a partial date, whereas TempTool-R1 generates complete answers. This indicates that our method not only enhances complex temporal reasoning but also provides more comprehensive answers.

## 5 Conclusion

We present TempTool-R1, a novel training framework that introduces tool-integrated reasoning into TKGQA. We design a dataset-agnostic set of unified temporal tools and employ SFT and RL training scheme that enables the model to adaptively invoke appropriate tools for different types of questions. In addition, we design dedicated reward functions tailored to temporal tool-integrated reasoning. Extensive experiments demonstrate that TempTool-R1 consistently improves performance, confirming the effectiveness of our framework. We hope our framework will open up a new direction for TKGQA, and we plan to further explore the role of RL on TKGQA task in future work.

## Limitations

Despite its strong performance, our approach has several limitations. First, the correctness of TempTool-R1 depends not only on selecting appropriate temporal tools but also on the accuracy of their outputs, and ambiguous entity or relation semantics may lead to incorrect tool results that propagate through multi-step reasoning. Second, due to computational constraints, our experiments are mainly conducted on models under 4B parameters, and we do not fully explore very large models (e.g., 32B or 72B), although preliminary scaling results suggest consistent performance gains with increasing model size. Third, while the underlying principle of encoding tool dependencies in the reward signal is general, the specific reward components (e.g., the TPR-before-TCR ordering constraint) are tailored for TKGQA, and their direct applicability to other question answering tasks or non-temporal reasoning settings remains to be explored. Future work will focus on improving tool reliability and verification, systematically studying scaling behavior at larger model sizes, and developing more general and transferable reward designs for broader QA tasks.

## Acknowledgments

This work was supported by National Language and Character Research Base (ZDI145-168).

## References

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023a. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

Zhuo Chen, Zhao Zhang, Zixuan Li, Fei Wang, Yutao Zeng, Xiaolong Jin, and Yongjun Xu. 2024a. 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, Torino, Italia. ELRA and ICCL.

Ziyang Chen, Dongfang Li, Xiang Zhao, Baotian Hu, and Min Zhang. 2024b. 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, Bangkok, Thailand. Association for Computational Linguistics.

Ziyang Chen, Jinzhi Liao, and Xiang Zhao. 2023b. 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, Toronto, Canada. Association for Computational Linguistics.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

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, Minneapolis, Minnesota. Association for Computational Linguistics.

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, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *Preprint*, arXiv:2504.11536.

Alberto García-Durán, Sebastijan Dumančić, and Mathias Niepert. 2018. Learning sequence encoders for temporal knowledge graph completion. *Preprint*, arXiv:1809.03202.

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, Suzhou, China. Association for Computational Linguistics.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

Qianyi Hu, Xinhui Tu, Cong Guo, 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, Albuquerque, New Mexico. Association for Computational Linguistics.

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*, CIKM '21, page 792–802, New York, NY, USA. Association for Computing Machinery.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

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 *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. Torl: Scaling tool-integrated rl. *Preprint*, arXiv:2503.23383.

Ruotong Liao, Xu Jia, Yangzhe Li, Yunpu Ma, and Volker Tresp. 2024. GenTKG: Generative forecasting on temporal knowledge graph with large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4303–4317, Mexico City, Mexico. Association for Computational Linguistics.

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 *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5141–5149. International Joint Conferences on Artificial Intelligence Organization. Main Track.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5825–5833.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex

Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *Preprint*, arXiv:2504.13958.

Xinying Qian, Ying Zhang, Yu Zhao, Baohang Zhou, Xuhui Sui, Li Zhang, and Kehui Song. 2024. TimeR$^4$ : 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, Miami, Florida, USA. Association for Computational Linguistics.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, and 24 others. 2024a. Tool learning with foundation models. *ACM Comput. Surv.*, 57(4).

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024b. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

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, Online. Association for Computational Linguistics.

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, Online. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Preprint*, arXiv:2302.04761.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

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, Dubrovnik, Croatia. Association for Computational Linguistics.

Qiang Sun, Sirui Li, Du Huynh, Mark Reynolds, and Wei Liu. 2025. Timelinekgqa: A comprehensive question-answer pair generator for temporal knowledge graphs. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 797–800, New York, NY, USA. Association for Computing Machinery.

Zhiwen Tan, Jiaming Huang, Qintong Wu, Hongxuan Zhang, Chenyi Zhuang, and Jinjie Gu. 2025. Rag-r1: Incentivizing the search and reasoning capabilities of llms through multi-query parallelism. *Preprint*, arXiv:2507.02962.

Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. 2024. FreshLLMs: Refreshing large language models with search engine augmentation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13697–13720, Bangkok, Thailand. Association for Computational Linguistics.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *Preprint*, arXiv:2509.02479.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *Preprint*, arXiv:2504.03160.

# A Dataset Details

## A.1 MULTITQ

MULTITQ (Chen et al., 2023b) is a complex temporal question-answering dataset that contains multi-granularity temporal information. Compared to existing datasets, MULTITQ offers several advantages, including a larger scale, richer relationships, the need for multi-hop reasoning, and various temporal granularities, making it more reflective of real-world scenarios. MULTITQ includes six types of questions, ranging from simple to complex temporal problems. The complex questions involve multiple temporal constraints, such as "after", "before", "first", and "last". Additionally, Chen et al. use a subset of the Integrated Crisis Early Warning

System (ICEWS) database, specifically ICEWS05-15 (García-Durán et al., 2018), as the knowledge graph for MULTITQ, which poses a greater challenge for LLMs. The question types of MULTITQ are shown in Table 7.

### A.2 TIMELINEKGQA

TIMELINEKGQA (Sun et al., 2025) is an automatic QA-pair generation framework for temporal knowledge graphs, used to construct two challenging benchmark datasets from ICEWS (García-Durán et al., 2018) and CronQuestions (Saxena et al., 2021). Solving its questions requires models to perform Temporal Constrained Retrieval (TCR), Timeline Position Retrieval (TPR), Temporal Semantic Operation (TSO), and Timeline Arithmetic Operation (TAO), and involves multi-hop and complex temporal reasoning, which significantly increases the difficulty of the task. The question types of TIMELINEICEWS and TIMELINECRONQUESTIONS are shown in Table 8.

## B Baseline Details

In this section, we provide a detailed description of the baseline models used in our experiments.

- **Pre-trained LMs**: To evaluate BERT (Devlin et al., 2019) and ALBERT (Lan et al., 2020), we first generate their LM-based question embedding and then concatenate it with the entity and time embeddings. These embeddings are processed through a learnable projection layer. Finally, the resulting embeddings are scored using dot products with all entities and timestamps.
- **EmbedKGQA**: (Saxena et al., 2020) handles multi-hop KGQA reasoning tasks, and for evaluating the TKGQA dataset, timestamps are ignored during training, with random time embeddings used instead.
- **CronKGQA**: (Saxena et al., 2021) is designed to handle single-granularity temporal questions. To address the multi-granularity temporal questions in MULTITQ, temporal embeddings of either yearly or monthly granularity are randomly sampled from the corresponding time embeddings.
- **MultiQA**: (Chen et al., 2023b) is a method specifically designed for handling multi-granularity temporal problems, with a transformer-based time aggregation module.
- **ChatGPT**: We use OPENAI API (gpt-4o-mini[1]) to directly answer the questions.
- **RAG**: To investigate the performance of LLMs integrated with external knowledge in the TKGQA task (Guu et al., 2020), we first embed the TKG and the questions using the Qwen3-Embedding-8B model. Based on the similarity scores, we rank and select the top-K entries, which are then passed to the large model for final answer generation. Throughout this process, no additional handling is applied to the temporal aspects of the questions.
- **ARI**: (Chen et al., 2024b) is one of the advanced methods for handling multi-hop question answering tasks. It advocates for abstracting past actions into generalizable strategies, which then guide the handling of subsequent similar questions. Additionally, this method effectively addresses temporal questions with multiple granularities and temporal constraints, making it well-suited for TKGQA tasks.
- **ProgTQA**: (Chen et al., 2024a) design fundamental temporal operators for time constraints and introduce a novel self-improvement Programming method. This method constructs a dedicated dataset for supervised fine-tuning, enabling the model to transform temporal questions into logical queries and thereby solve TKGQA tasks with higher precision.
- **TempAgent**: (Hu et al., 2025) is a novel LLM-based autonomous agent framework designed to enhance the temporal reasoning and understanding capabilities of LLMs. By explicitly integrating temporal constraints into the information retrieval process, TempAgent can effectively filter out irrelevant content and focus on extracting temporally relevant information and factual knowledge.
- **TimeR**[4]: (Qian et al., 2024) is a state-of-the-art method for handling TKGQA tasks, effectively addressing multi-hop TKGQA problems. It first performs an initial retrieval for complex questions, and after retrieving answers, it rewrites the original question to transform implicit temporal constraints into

[1] https://platform.openai.com/docs/models/gpt-4o-mini

explicit ones. Additionally, this method implements a *retriever-rank* module, designed to retrieve semantically and temporally relevant facts from the TKG and reorder them based on temporal constraints. To achieve this, they fine-tune the retriever using a contrastive time-aware learning framework.

- **RTQA**: (Gong et al., 2025) recursively decomposes the original question into a series of sub-questions, solves them in a bottom-up manner by leveraging both LLMs and TKG knowledge, and employs multi-path answer aggregation to enhance robustness to errors. With this design, RTQA achieves substantially improved performance on complex types of questions.

# C Implementation Details

## C.1 Tool Implementation

For each temporal tool, we first parse the tool name, arguments, and argument values from the tool interface, and then perform semantic alignment for entity and relation arguments. Concretely, we use Qwen3-Embedding-8B to compute similarity between the extracted entities/relations and all entities and relations in the TKG, and retain the top 50 candidates as a preliminary candidate set. We then apply Qwen3-Reranker-8B to rerank these candidates and select the top-1 as the final aligned entity or relation, using the reranking prompt shown in the Figure 6. Finally, we combine the aligned entities and relations with the time arguments and temporal constraint type to filter the TKG and return the corresponding knowledge graph tuples as the output of the tool call.

Furthermore, the datasets differ in how they represent temporal information. MULTITQ adopts timestamps, while TIMELINEKGQA encodes time as intervals. To address this discrepancy, we implement a unified mechanism that enables our tools to correctly parse and process temporal data regardless of whether it takes the form of timestamps or time intervals. In addition, we standardize the time-related parameters in our tool interfaces as *begin_time* and *end_time*, which provides a consistent abstraction for both temporal formats. With this design, our tools can reliably handle diverse temporal representations and effectively prevent inference failures caused by incorrect temporal data parsing.

## C.2 Construction of Training Data

We first use the predefined temporal tools and call the DEEPSEEK-V3.2 API to obtain tool-augmented reasoning trajectories on the MULTITQ and TIMELINEKGQA datasets. The system prompt is shown in the figure 7. We then filter these trajectories based on both answer correctness and the validity of the intermediate reasoning steps, and retain only high-quality trajectories as our usable corpus $D$. For the SFT stage, we randomly sample a subset from $D$ and split it into training, validation with a 9:1 ratio. The SFT training set contains 10k examples from MULTITQ, 7k from TIMELINECRONQUESTIONS, and 7k from TIMELINEICEWS, yielding 24k instances in total. In particular, the proportions of the sampled question types in the train set are kept consistent with their distribution in the original dataset.

For the RL stage, we use the SFT model to generate predictions on the remaining (non-training) portion of the data and select those examples where the model's answers are incorrect as RL training instances. These samples are predominantly multi-hop (2/3/4-hop) or involve complex temporal operations. Additionally, we retain a small portion of simple samples to prevent the training set from becoming overly difficult, which could otherwise slow down model convergence. In total, we select 6k such hard examples from the three datasets, ensuring that they do not overlap with the SFT training data. In the pure-RL setting (i.e., without the cold-start SFT stage), we instead construct the RL training set by sampling a mixture of simple, medium, and complex questions from all three datasets.

## C.3 Training Details

The training process consists of two main stages: SFT and RL. During SFT stage, we perform full-parameter fine-tuning with the following hyperparameters: a batch size of 16, 3 training epochs, and a learning rate of 1e-5. For the RL stage, we use GRPO algorithm (Shao et al., 2024) with the following hyperparameters: a batch size of 128, a maximum prompt length of 4096, a maximum response length of 1024, a learning rate of 5e-7, a clipping ratio of 0.2, a KL loss coefficient of 0.001, a rollout sample size of $n = 4$, a sampling temperature of 0.7 and 10 training epochs.

| Question | Model | Reasoning / Tool Calls | Predicted Answer |
|---|---|---|---|
| After Roh Moo Hyun, who was the first to criticise the South Korean ministry? | TempTool-R1 | **Step 1**: Call *Get_time* with head = "Roh Moo Hyun", rel = "criticise", tail = "Ministry (South Korea)", obtaining "2007-12-19".<br>**Step 2**: Call *Get_head_entity* with tail = "Ministry (South Korea)", rel = "criticise", begin_time = "2007-12-19", type = "after". Candidates: Governor (South Korea) at "2010-08-04" and Japan at "2011-05-25".<br>**Step 3**: TempTool-R1 selects the earliest critic. | Governor (South Korea) (correct) |
| | DeepSeek-V3 | Directly answers based on internal knowledge without temporal tool calls or comparison over the TKG. | Ministry (South Korea) (incorrect) |
| In which month did Luiz Gushiken threaten the leader of Brazil? | TempTool-R1 | **Step 1**: Call *Get_time* with head = "Luiz Gushiken", rel = "threaten", tail = "the leader of Brazil", obtaining "2007-08-24". | 2007-08 (correct) |
| | DeepSeek-V3 | Answers focused only on the month and omits the year, resulting in an answer with an incorrect format. | August (incorrect) |

Table 6: Case studies comparing TempTool-R1 and DeepSeek-V3 on two temporal reasoning questions. TempTool-R1 interacts with the TKG via temporal tools to obtain the information needed to produce correct answers.

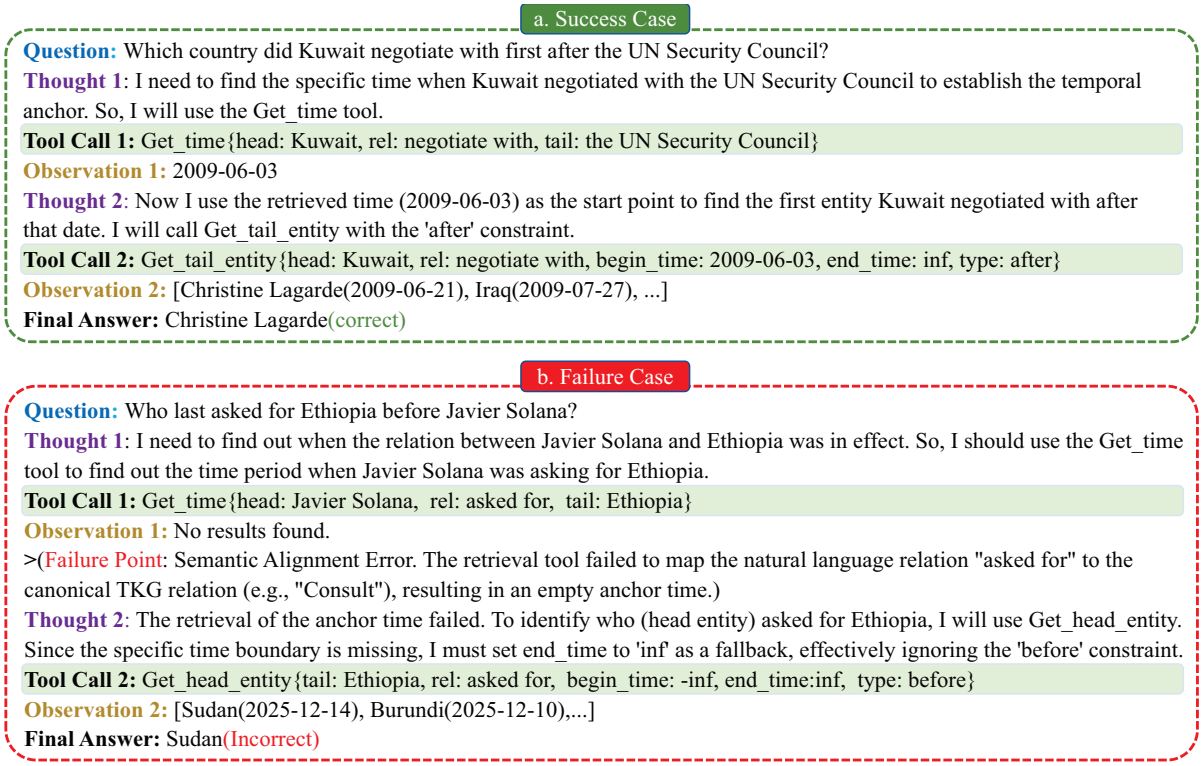


Figure 5: Case study on two examples from MULTITQ using Qwen3-4B as a backbone.

## C.4 Evaluation

After training, we evaluate TempTool-R1 on temporal QA benchmarks. The answers to temporal questions fall into three main types: entity answers, temporal answers, and logical answers (e.g., Boolean). For entity answers, we require strict matching against the gold entity set and report Hits@1, i.e., we check whether the first entity produced by the model appears in the ground truth. For temporal answers, we require the prediction to be semantically identical to the gold timestamp and precise to the specified granularity. For example, if the gold answer is "2010-10", a prediction such as "October 2010" is considered correct, whereas "October" alone is counted as incorrect. For logical answers, we only require that the predicted label be logically consistent with the gold label (such questions primarily come from TIMELINEKGQA). To automate evaluation, we design dedicated judging prompts and use the DEEPSEEK-V3.2 API as

| Category | | Train | Dev | 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** | **587,979** | **54,584** |

Table 7: Statistics of question categories in MULTITQ.

| Source KG | Type | Train | Val | Test |
|---|---|---|---|---|
| *ICEWS Actor* | Simple | 17,982 | 5,994 | 5,994 |
| | Medium | 15,990 | 5,330 | 5,330 |
| | Complex | 19,652 | 6,550 | 6,550 |
| | **Total** | **53,624** | **17,874** | **17,874** |
| *CronQuestion KG* | Simple | 7,200 | 2,400 | 2,400 |
| | Medium | 8,252 | 2,751 | 2,751 |
| | Complex | 9,580 | 3,193 | 3,193 |
| | **Total** | **25,032** | **8,344** | **8,344** |

Table 8: Questions distribution across Train/Val/Test sets for TIMELINEICEWS and TIMELINECRONQUESTIONS.

an LLM-as-a-judge to decide whether a model output should be marked correct. While many recent works have adopted LLM-based judges, we additionally validate their reliability in our setting. We randomly sample 1,000 instances, obtain both human judgments and DEEPSEEK-V3.2 judgments, and measure their agreement. The agreement between human evaluators and the model-based judge reaches 0.92, indicating a high level of consistency in evaluating TKGQA answers. Although minor discrepancies may still exist, their impact can be mitigated by averaging results over multiple evaluation runs, with the remaining variance staying within a controllable range. Therefore, considering both evaluation cost and difficulty, we conclude that using an LLM as a judge is an effective and practical approach. Moreover, a growing body of prior work has demonstrated the effectiveness of LLMs as judges for model evaluation (Gu et al., 2025). Building on the above observations, we employ an LLM for automated evaluation. We employ the pre-defined test sets from both MULTITQ and TIMELINEKGQA. All reported experimental results are obtained by averaging the model's performance over multiple runs on these test sets.

**Reranker Prompt Template**

You are a semantic reranker for entity/relation alignment in a temporal knowledge graph.
Your task is to rank candidates by how semantically equivalent they are to the query relation, focusing on realized actions or factual relations.

Query: {*entity_or_rel*}
Candidates: {*candidates*}

Figure 6: Reranker prompt template.

## D Case Study

In this section, we provide a more detailed case study of TempTool-R1. We select two challenging multi-hop temporal questions, as illustrated in the figure 5. In the success case, the model first reasons about the complex query, then chooses the correct tool name, parameter names, and parameter values, and interacts with the TKG step by step through temporal tools, eventually producing the correct answer. This process clearly demonstrates the effectiveness of our approach. Through SFT and RL training, the model learns when and how to use temporal tools and demonstrates strong generalization across different datasets.

In the failure case, the model still selects an appropriate tool and fills in reasonable parameter names and values, yet the tool ultimately returns the message "No results found". A deeper inspection shows that the core issue lies in the semantic alignment stage, where the relation "ask for" is not correctly aligned with the TKG relation "consult". As a consequence, an incorrect relation is used, no relevant subgraph is retrieved, and the tool returns "No results found". We regard this phenomenon primarily as an implementation and alignment limitation of the tool rather than a fundamental weakness of the model. Fortunately, such corner cases account for only a very small fraction of the entire test set. Nevertheless, their presence indicates that there is still room for further improvement in the overall performance. This issue can be alleviated by refining the prompt used for the reranker or by fine-tuning the reranking model, and we plan to further explore these directions in future work.

## E Latency Analysis

We conducted a latency analysis under the same experimental settings. Specifically, we sampled 2,400 and 2,000 instances from MULTITQ and

| Method | MultiTQ | | | | TimelineCronQuestions | | | | TimelineICEWS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Num | ATPS | Time | TPS | Num | ATPS | Time | TPS | Num | ATPS | Time | TPS |
| Qwen3-4B | 2400 | 4250 | 4176 | 0.575 | 2000 | 4246 | 4285 | 0.467 | 2000 | 4247 | 4187 | 0.478 |
| Llama3.2-3B | 2400 | 4068 | 5328 | 0.450 | 2000 | 4061 | 5405 | 0.370 | 2000 | 4063 | 5349 | 0.374 |
| TempTool-R1 (4B) | 2400 | 5628 | 3642 | 0.659 | 2000 | 5625 | 3795 | 0.527 | 2000 | 5629 | 3763 | 0.531 |

Table 9: Latency analysis between TempTool-R1 and base model on three datasets. Time is recorded in seconds.

| Model | Overall | Question Type | | Answer Type | |
|---|---|---|---|---|---|
| | | Simple | Complex | Entity | Time |
| ARI | 0.380 | 0.680 | 0.210 | 0.394 | 0.344 |
| RTQA | 0.765 | 0.902 | 0.424 | 0.692 | 0.942 |
| TempTool | **0.828** | **0.914** | **0.730** | **0.755** | **0.945** |

Table 10: Hits@1 of our TempTool methods on the MULTITQ dataset.

| Model | MultiTQ | CronQuestionKG | ICEWS Actor |
|---|---|---|---|
| ARI | 0.380 | - | - |
| RTQA | 0.765 | 0.298 | - |
| **TempTool-R1** | **0.785** | **0.427** | **0.416** |

Table 11: Hits@1 Performance on unseen datasets.

TIMELINEKGQA, respectively. Here, ATPS denotes the average number of tokens generated per second, while TPS represents the number of samples processed per second. As shown in Table 9, TempTool-R1 processes tasks significantly faster than the untrained foundation model. This indicates that the trained model can solve challenging problems more efficiently, rather than spending many reasoning steps only to produce incorrect answers. Taken together, these results demonstrate the efficiency of TempTool-R1.

## F Further Discussion

**TempTool vs Others.** To assess the effectiveness of our temporal tool for TKGQA, we invoke the OPENAI API (gpt-4o-mini) and expose our tools as function-style interfaces, allowing the model to adaptively select appropriate temporal tools, interact with the TKG, and iteratively derive the final answer. We refer to this approach as TempTool and compare it with ARI and RTQA, both of which use the same OPENAI API. As shown in Table 10, TempTool achieves the best performance, indicating that explicitly encoding interactions with the TKG as tools can further enhance the reasoning ability of large language models on TKGQA. Interestingly, our TempTool-R1 model slightly outperforms the TempTool with LLM. This observation demonstrates the reliability of our strategy: by first refining and filtering the outputs of the LLM, and then applying knowledge distillation and reinforcement learning, the resulting model can exhibit stronger reasoning capabilities on TKGQA tasks than the general-purpose LLM itself.

**Generalization on Unseen Datasets.** To assess generalization, we train our models on the MULTITQ training set and evaluate them on the TIMELINEKGQA test set, and vice versa. As shown in Table 11, our approach still exhibits strong performance on this unseen dataset. At the same time, we analyze the causes of the performance gap between our model and the fully trained counterpart. Our findings indicate that a major source of degradation lies in the model's failure to accurately extract complete key information when it encounters unfamiliar entities or relations, which in turn introduces errors in subsequent tool retrieval. Nevertheless, the model still demonstrates strong performance in deciding when to invoke tools and in selecting appropriate tools together with their required parameters. We attribute this primarily to the dataset-agnostic design of our unified temporal tool interface and the carefully crafted fine-grained temporal tool rewards, which enable the model to use tools more effectively for TKGQA and thereby improve its ability to generalize across different datasets.

**Further Discussion on Tool Implementation.** In our tool implementation, we adopt a word-wise embedding and reranking strategy to semantically align the entities and relations extracted by the model with the predefined entities and relations stored in the library, and then perform precise retrieval over the TKG. An alternative design is to bypass explicit entity and relation extraction and alignment, and instead apply sentence-wise embedding directly to the relevant utterances in order to select the corresponding knowledge graph. However, this design has notable drawbacks. Natural language queries often contain substantial noise, such as interrogative words, and multi-hop questions must first be decomposed into sub-questions

before embedding. We argue that this increases the difficulty of reliably transforming queries into appropriate tool calls. Therefore, we choose to let the model explicitly extract the relevant entities and relations and feed them directly as tool parameters. This design is more consistent with the logic of Tool-Integrated Reasoning and decouples the parameters within the query, which in turn makes it easier to diagnose and analyze error sources.

Furthermore, when selecting the embedding and reranking models, we follow the principles of being open-source and achieving strong empirical performance. Guided by the MTEB leaderboard[2] , we adopt Qwen3-Embedding-8B and Qwen3-Reranker-8B as our base models. However, during inference, we observe that the reranking model is not sufficiently sensitive to certain entity and relation expressions, which makes it difficult to consistently place the ground-truth entity-relation pair at the top rank. As a result, in the next stage, we plan to fine-tune the reranking model to address this issue. Improving the alignment of entities and relations through such fine-tuning is crucial for obtaining high-precision tool retrieval results.

[2]Available at https://huggingface.co/spaces/mteb/leaderboard.

## System Prompt Template

**You are an intelligent temporal knowledge graph reasoning agent.**
Your goal is to answer temporal questions by **reasoning step-by-step** and by **calling tools when needed**. For every question, you must first reason, then either call a tool or output the final answer.
**1. Available Tools** You have three tools. Choose carefully **which tool to use, when to use it, and how to fill the parameters**.
(1) *Get_time(head: str, rel: str, tail: str)*
Use this when you want to know **when** a given relation between a head and tail holds.

(2) *Get_tail_entity(head: str, rel: str, begin_time: str, end_time: str, type: "in/on" | "before" | "after" | "between")*
Use this when you know the **head entity and relation**, and want to find **which tail entity** satisfies this relation within a time range.

(3) *Get_head_entity(tail: str, rel: str, begin_time: str, end_time: str, type: "in/on" | "before" | "after" | "between")*
Use this when you know the **tail entity and relation**, and want to find **which head entity** satisfies this relation within a time range.

**2. How to Parse the Question** When analyzing the question, you must **extract entities and relations as complete, continuous substrings** from the original question.
**Entities**: named entities or definite noun phrases (e.g., "Obama", "the head of Turkmenistan", "the Cabinet Council of Ministers of Kazakhstan", "Iran", "China").
**Relations**: verbs or verbal phrases describing actions or states (e.g., "visit", "praise", "express the intention to negotiate with").
Do **not** break entity phrases or mix them into the relation.
Do **not** paraphrase or summarize the verb phrase: use the **exact substring** from the question.
**3. Time Parameter Construction** When constructing time parameters *begin_time* and *end_time*:

- If the question says "before X": *begin_time = "-inf"*
  *end_time = "X"*
- If the question says "after X": *begin_time = "X"*
  *end_time = "inf"*
- If the question has no explicit time restriction: *begin_time = "-inf"*
  *end_time = "inf"*
- For *in / on / between*, set both *begin_time* and *end_time* literally following the question.

Always use the literal strings *"-inf"* for infinitely early time and *"inf"* for infinitely late time.
**4. Output Format Rules**
For **every reasoning step**, you must follow this structure exactly:
(1) First, internal reasoning:
<*think*>
Explain your reasoning process here (why you choose this tool or why not).
</*think*>
Here you describe how you parsed entities and relations, how you chose the tool, time range, etc.
(2) If you decide to use tools, output a <*tool_call*> block:
<*tool_call*>
*{"name": "ToolName", "parameters": {"param1": "value1", "param2": "value2"}}*
*{"name": "AnotherToolName", "parameters": {"paramA": "valueA"}}*
</*tool_call*>
You may invoke **one or multiple** tool calls at once. Each tool call is a JSON object with *"name"* (one of *Get_time*, *Get_tail_entity*, *Get_head_entity*) and a *"parameters"* dictionary.
(3) Tool results:
After you output <*tool_call*>, the system will return the tool results inside <*obs*>...</*obs*> tags in the next input. You must **not** output <*obs*> tags yourself. On the next step, read those observations and reason again in a new <*think*>...</*think*> block.
(4) Final answer:
When you have enough information and no more tools are needed, output:
<*response*>
Your final answer should be an **entity**, a **time expression**, or **no answer** only, not a full sentence.
</*response*>

Figure 7: System prompt template for TempTool.