Benchmarking Multi-Hop Reasoning with Controllable Synthetic Datasets

Anonymous ACL submission

Abstract

Large Language Model (LLM)-based methods for Temporal Knowledge Graph (TKG) reasoning tasks have found success by relying on LLMs' impressive pattern recognition abilities. But the extent of this capability on complex multi-hop patterns is understudied. In order to understand the limits of LLM-based methods' multi-hop reasoning abilities, we create a novel synthetic TKG generator and a suite of realistic TKG datasets with varied complexity along several important dimensions. In particular, we study multi-hop patterns that have been complicated by number of hops, time dispersion, and imbalanced relation and entity distributions. We benchmark the abilities of LLM- and Graph Neural Network (GNN)-based methods on these synthetic TKGs, finding that LLMs can far outperform GNN-based methods when provided with ideal contexts. However, their performance degrades sharply as contextual noise increases, indicating that retrieval, not multi-hop composition itself, is the primary bottleneck.

1 Introduction

TKG reasoning has traditionally been approached by extending methods for static KG (Knowledge Graph) reasoning. LLMs are now proving useful at TKG reasoning tasks on the basis of their generalizable pattern recognition abilities. Specifically, via in-context learning in which an LLM is provided with a historical context of relevant facts from a TKG and prompted to predict the completion of a test fact. While LLMs have shown promising overall results on benchmark tasks, recent work has found that LLMs struggle with multi-hop reasoning tasks (Wu et al., 2024b; Khodadad et al., 2025; Yang et al., 2025). These complex, multi-hop tasks are the ones that are most of interest in real-world applications and for extending the capabilities of LLM-based TKG reasoning.

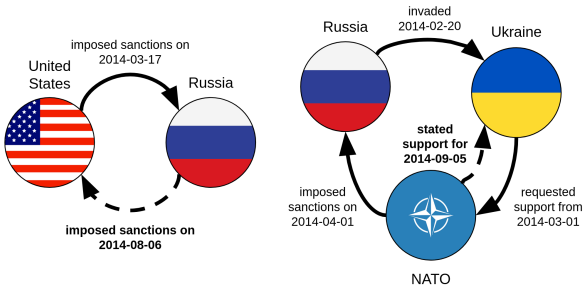


Figure 1: Example of two sequences of events of varying hop length: **(left)** 1-hop pattern where the U.S. imposed sanctions on Russia, which imposed reciprocal sanctions in return; **(right)** 3-hop pattern where Russia invaded Ukraine, which requested support from NATO, which imposed sanctions on Russia, and later stated support for Ukraine.

In order to understand exactly where current LLMs fall short on multi-hop reasoning, we require a controllable, but realistic dataset. Current real-world TKG benchmarks are created from large, noisy event data without certainty of the precise generating patterns. Synthetic TKG datasets are often built around relatively simple schema at the cost of realism. A controllable, realistic dataset would allow the study of multi-hop patterns on specific dimensions of their complexity.

By assembling realistic temporal networks from these motifs, we can probe the multi-hop reasoning ability of LLMs while directly controlling task difficulty along interpretable axes. Our experiments reveal that the main bottleneck in LLM-based TKG reasoning is not pattern recognition itself, but reliably retrieving and composing the correct contextual evidence as structural complexity increases.

To summarize, our contributions are:

- We introduce a synthetic TKG generator that produces controllable and realistic datasets, and we release a suite of such benchmarks.
- We systematically benchmark LLM- and GNN-based methods on these datasets under controlled

variations in pattern complexity, temporal dispersion, and entity/relation distributions.

2 Related Work

2.1 TKG Reasoning Methods

TKG reasoning methods must capture both local graph structure and temporal dynamics of data.

2.1.1 Pattern Mining Methods

Rule-mining approaches extract interpretable temporal rules that are also useful for forecasting. TLogic (Liu et al., 2022) mines rules from cyclic temporal random walks. TR-Rules (Li et al., 2023) allow acyclic rules and reduces temporal redundancy using a windowed confidence metric.

2.1.2 GNN-Based Methods

History-aware neural models leverage TKG structure with recurrent or temporal modules to model evolution over time (Jin et al., 2020; Li et al., 2022). TeMP (Wu et al., 2020) performs temporal message passing closely related to the patterns we study. CyGNet (Zhu et al., 2021) learns separate distributions for high-repetition entities and global entities, exploiting repeated simple patterns in benchmarks. CENET (Xu et al., 2023b) uses historical contrastive learning to separate historical from non-historical dependencies. These works implicitly distinguish between *simple* and *complex* temporal patterns, mirroring our axes of investigation.

2.1.3 LLM-Based Methods

Because LLMs operate on text, TKGs must be textualized before inference (Xu et al., 2023a). Once in text form, standard LLM techniques such as in-context learning (Lee et al., 2023), chain-of-thought prompting (Xia et al., 2024), and retrieval-augmented generation (Liao et al., 2023) can be applied. Lee et al. (2023) find that LLMs can match task-specific GNNs on TKG forecasting, even when semantic information is obfuscated. Chain-of-History (Xia et al., 2024) uses higher-order histories as intermediate reasoning steps, while GenTKG (Liao et al., 2023) mines temporal rules and uses them in a RAG framework. We similarly focus on how the provided context interacts with task complexity.

2.2 TKG and Reasoning Datasets

TKG reasoning is typically evaluated on event datasets such as GDELT (Leetaru and Schrodt, 2013), Wikidata (Vrandečić and Krötzsch, 2014), and ICEWS (Boschee et al., 2015). These corpora are realistic but noisy and do not provide guarantees about the necessity or sufficiency of mined patterns, hence motivating synthetic datasets with known generative structure.

For static KGs, controllable benchmarks such as FRUNI and FTREE (Martin et al., 2023) vary size and noise under fixed schemas, and synthetic KG QA testbeds have been built to probe robustness of supervised fine-tuning and in-context learning for multi-hop QA (Shah et al.). In contrast, we focus on temporal KGs and directly control the underlying temporal patterns rather than only the surface statistics.

Multi-hop reasoning has also been studied on non-KG datasets, particularly multi-hop QA over documents (Yang et al., 2018; Geva et al., 2021). 2WikiMultiHopQA (Ho et al., 2020) defines explicit evidence chains, while MuSiQue (Trivedi et al., 2022) composes single-hop questions to reduce shortcut opportunities. Our setting abstracts away from natural language and evaluates multi-hop reasoning directly on the underlying TKG.

2.3 LLM Reasoning

Recent work shows that LLM performance degrades with hop count and often relies on shortcuts or flawed reasoning chains. Biran et al. (2024) find that 2-hop queries are solved via early identification of *bridge* entities followed by later resolution, leaving relevant information vulnerable to being overwritten. Counterfactual and unprecedented variants of QA benchmarks reveal a strong dependence on spurious correlations (Wu et al., 2024a,b). Although LLMs exhibit some compositional reasoning ability (Yang et al., 2025; Khodadad et al., 2025), it is unreliable. Grounding LLM chain-of-thought rationales in KGs also shows that generated paths are frequently invalid (Nguyen et al., 2024). Our synthetic TKGs isolate the multi-hop reasoning component while controlling for other sources of ambiguity.

3 Preliminaries

3.1 Temporal Knowledge Graphs

Let $\mathcal{E} = \{1, ..., |\mathcal{E}|\}$ be an entity set, $\mathcal{R} = \{1, ..., |\mathcal{R}|\}$ a relation set, and $\mathcal{T} = \{1, ..., |\mathcal{T}|\}$ a set of discrete timestamps. A temporal knowledge graph (TKG) $\mathcal{G}$ is defined as a set of quadruples $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E} \times \mathcal{T}$. In other words, a TKG $\mathcal{G}$ is composed of events of the form $(h, r, t, \tau) \in \mathcal{G}$,

where h and t are called head and tail entities, respectively.

In TKG reasoning, we are presented with a query event of the form $q = (h, r, t, \tau')$ with one of h, r, or t masked. The task is to predict the value of the masked field. In the *interpolation* case, we have access to events before and after query event time τ'. In the *extrapolation* or *forecasting* case, we only have access to events before time τ'. In our work, we focus on the task of tail prediction in the forecasting setting.

3.2 Temporal Reasoning Patterns

Intuitively, each pattern defines a minimal multi-hop reasoning task: a small set of temporally ordered facts whose joint occurrence implies a future event. By explicitly controlling the number, structure, timing, and frequency of such patterns, we can generate TKGs where the difficulty of reasoning is known by construction rather than inferred post hoc.

A **temporal pattern** $p \in \mathcal{P}$ of hop-length k is defined as a tuple $p = (A_p, c_p, \Delta_p)$, with antecedent sequence $A_p = [a_1, ..., a_k]$, consequence triple c_p, and lag constraints $\Delta_p = [\delta_1, ..., \delta_k]$, where each $\delta_i = (\delta_i^-, \delta_i^+)$ is an integer interval. Each antecedent a_i and the consequence c_p are triples of the form:

$$a_i = (x_i, r_i, y_i), c_p = (x_c, r_c, y_c) \tag{1}$$

where x_*, y_* are placeholders from a finite placeholder set $\mathcal{V}$. Relations $r_* \in \mathcal{R}$ are always concrete. We also introduce label set $\ell(h, r, t, \tau)$ to indicate which pattern generated a given quadruple.

A **pattern instantiation** for pattern p in graph $\mathcal{G}$ is a sequence of quadruples $\{(x_i, r_i, y_i)\}_{1 \le i \le k} \cup \{(x_c, r_c, y_c)\}$ at times $\tau_1, ..., \tau_{k+1}$ such that:

$$\begin{aligned} \forall a_i = (x_i, r_i, y_i) \in A_p : (x_i, r_i, y_i, \tau_i) \in \mathcal{G} \\ c_p \oplus \tau_{k+1} = (x_c, r_c, y_c, \tau_{k+1}) \in \mathcal{G} \\ \forall i = 1, ..., k : \tau_{i+1} - \tau_i \in [\delta_i^-, \delta_i^+] \end{aligned} \tag{2}$$

In other words, a sequence of quadruples satisfying the entity, relation, and temporal constraints of the pattern. We formally define an instantiation as a tuple $(\pi, \{\tau\}_{1:k+1})$, where $\pi : \mathcal{V} \to \mathcal{E}$ maps placeholders to concrete entities, and $\{\tau\}_{1:k+1}$ is a time sequence satisfying temporal lag constraints.

To construct the finite pattern set $\mathcal{P}$, we exhaustively enumerate all 1-, 2-, and 3-hop candidate templates over a small placeholder vocabulary and filter them using a collection of structural validity checks. These checks enforce (i) constraints on entity reuse, (ii) optional bans on self-loops, and (iii) a connectivity requirement that ensures the antecedents and consequence form a single cycle in the induced entity graph. The corresponding procedures are listed in Algorithms 8 to 11 in Appendix D.

3.3 Generator Notation

We define sampling distributions over entities and relations, represented by non-negative weight functions $w_E : \mathcal{E} \to \mathbb{R}_{\ge 0}$ and $w_R : \mathcal{R} \to \mathbb{R}_{\ge 0}$. We write $x \sim W_E$ and $r \sim W_R$ for draws from the corresponding normalized distributions.

Finite pattern set $\mathcal{P}$ contains patterns $p \in \mathcal{P}$, each of which is a tuple $p = (A_p, c_p, \Delta_p)$ of hop-length k, as defined in Section 3.2. We associate each pattern p with two control parameters: forcing probability $\rho_{\text{force}}(p) \in [0, 1]$ and forcing frequency $n_{\text{force}}(p) \in \mathbb{N}$. At each time step, we perform $n_{\text{force}}(p)$ independent Bernoulli trials with success probability $\rho_{\text{force}}(p)$ to decide how many explicit instantiations of p to inject.

We also maintain a label map $\ell : E \times R \times E \times T \to 2^{\mathcal{P}}$, where $\ell(h, r, t, \tau)$ is the set of patterns that generated the quadruple (h, r, t, τ); by convention $\ell(h, r, t, \tau) = \emptyset$ if no pattern is responsible. Finally, for a quadruple $q = (h, r, t, \tau)$ we write $\tau(q)$ for its time coordinate.

4 Synthetic TKG Generator

For each time step $\tau \in \mathcal{T}$, the generator iterates over patterns $p \in \mathcal{P}$: (i) injecting a number of forced instantiations of p according to $\rho_{\text{force}}(p)$ and $n_{\text{force}}(p)$ and (ii) instantiating consequences whenever all antecedents of p are already satisfied in the history. Together, these two mechanisms construct a synthetic TKG whose edges are entirely determined by the chosen pattern set. Forced instantiations inject complete pattern instances regardless of history, while spontaneous instantiations emit only consequences when all antecedents are already present in the graph

Algorithm 1 summarizes the full generation loop. It first samples concrete 1-, 2-, and 3-hop patterns from placeholder templates and then simulates the temporal KG via forced instantiations and implication-based consequences. Appendix C (Alg. 4) provides a full implementation, including bookkeeping utilities such as pattern instantiation and label updates.

Algorithm 1 Synthetic multi-hop TKG generation (detailed)

Require: Entity set E, relation set R, time horizon T, maximum hop length $H \in \{1, 2, 3\}$, template libraries $\{\tilde{\mathcal{P}}^{(k)}\}_{k=1}^{H}$, entity and relation weight functions w_E, w_R, time-lag profiles Lag, desired number of instantiated patterns $\{M_k\}_{k=1}^{H}$, forcing parameters for each pattern (probability ρ_{force} and frequency n_{force})

Ensure: Synthetic temporal knowledge graph G and instantiated pattern set $\mathcal{P}$

$\mathcal{P} \leftarrow \emptyset$; $G \leftarrow \emptyset$; $\ell \leftarrow$ empty label map ▷ initialize pattern set, graph, and pattern labels
for $k = 1$ to H **do** ▷ sample M_k concrete k-hop patterns
  for $m = 1$ to M_k **do**
    $\tilde{p} \leftarrow \textsc{SampleTemplate}(\tilde{\mathcal{P}}^{(k)})$ ▷ choose a k-hop template over placeholders
    $\pi \leftarrow \textsc{SampleEntityBinding}(w_E)$ ▷ map placeholders to concrete entities
    $\{r_i\} \leftarrow \textsc{SampleRelations}(\tilde{p}, w_R)$ ▷ instantiate relations in antecedents and consequence
    $\Delta_p \leftarrow \textsc{SampleLags}(\text{Lag}, k)$ ▷ choose a lag profile for this pattern
    $(\rho_{\text{force}}(p), n_{\text{force}}(p)) \leftarrow \textsc{SampleForcingParams}(k)$
    $p \leftarrow \textsc{InstantiatePattern}(\tilde{p}, \pi, \{r_i\}, \Delta_p)$ ▷ construct concrete pattern $p = (A_p, c_p, \Delta_p)$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$
  end for
end for
for time $\tau = 1$ to $|T|$ **do** ▷ simulate the temporal KG over time
  for pattern $p \in \mathcal{P}$ **do**
    $N \sim \text{Binomial}\big(n_{\text{force}}(p), \rho_{\text{force}}(p)\big)$ ▷ number of forced instantiations at time τ
    for $j = 1$ to N **do** ▷ inject explicit pattern instances
      $(G, \ell) \leftarrow \textsc{InjectForcedInstance}(G, \ell, p, \tau)$
    end for ▷ emit consequences implied by existing antecedents
    $(G, \ell) \leftarrow \textsc{CloseImpliedConsequences}(G, \ell, p, \tau)$
  end for
end for
return $G, \mathcal{P}$

We are thus able to control the following qualities of the resulting synthetic TKG. **Complexity** via total number of the provided patterns and their hop lengths in $\mathcal{P}$. **Temporal dispersion** via time lags constraints. **Entity and relation distributions** via weighting functions w_E and w_R. **Size** via number of entities $|\mathcal{E}|$, relations $|\mathcal{R}|$, and time windows $|\mathcal{T}|$. **Density** via instantiation probability ρ_{force} and number of trials n_{force}. In this work, we mainly focus on complexity, temporal dispersion, and entity/relation distributions, motivated by their relevance in event data (see Figures 1, 2, 3).

Realism arises not from semantic grounding (we use numeric identifiers for all entities and relations), but from reproducing structural motifs, frequency imbalance, and temporal dispersion patterns consistently observed in mined event graphs.

We include a characterization of the general shapes of our patterns in Figure 4. Other relation orientations and orderings of these antecedents are included in our generator, too. See Appendix B for a more detailed characterization of allowed shapes.

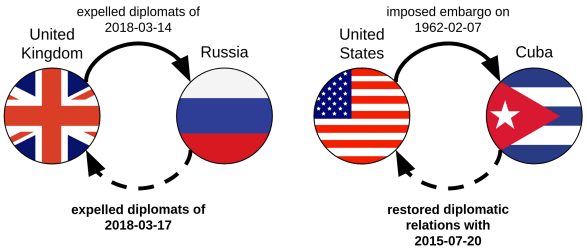


Figure 2: Example of two sequences of events of varying time dispersion: **(left)** A quick pattern in which the U.K. expelled Russian diplomats and 3 days later Russia did the same; **(right)** A slow pattern in which the U.S. imposed an embargo on Cuba in 1962 and it took 53 years for diplomatic relations to be restored.

5 Experiments

5.1 Datasets

We construct an ICEWS14-like synthetic dataset, ICEWS14-Synth, and ten additional synthetic datasets that systematically vary pattern complexity, temporal dispersion, and entity/relation distributions. Descriptive statistics for all datasets and

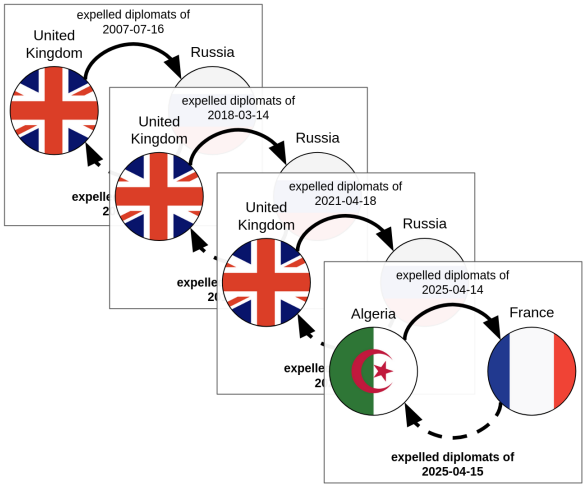


Figure 3: Example of multiple sequences of events from the same pattern - Country A expels Country B's diplomats, followed by a reciprocal expulsion of diplomats - with different entity instantiations. The first three sequences have the same entity instantiations, the U.K. and Russia. The final sequence involves a Algeria and France.

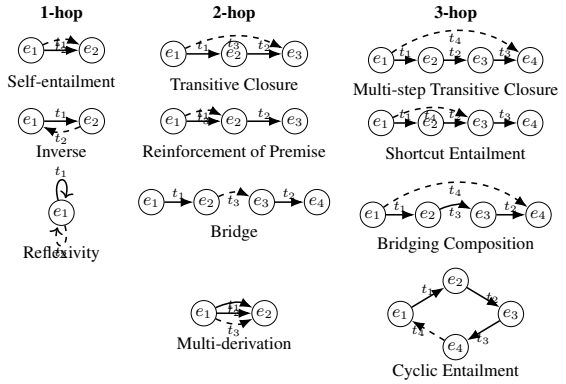


Figure 4: Examples of the general *shapes* of our patterns in our synthetic data generator. Each panel shows only entity connectivity (relation labels are suppressed) with event times τ_i marking the step order. **Legend**: Antecedents have solid lines; the consequence is dashed.

the original ICEWS14 corpus are shown in Table 1.

To obtain ICEWS14-Synth, we calibrate the generator hyperparameters so that the synthetic TKG matches ICEWS14 along simple descriptive statistics (number of entities, relations, timestamps, density, and degree quantiles) and recency/frequency baseline performance. Concretely, we minimize a weighted combination of relative errors in these statistics and absolute differences in Hits@K of the two heuristic baselines using Bayesian optimization (Algorithm 12). In practice we use a TPE-based sampler (Akiba et al., 2019) and a budget of 800 trials.

Our other synthetic datasets fall into three families:

		$\lvert\mathcal{G}\rvert$	$\lvert E\rvert$	$\lvert R\rvert$	$\lvert T\rvert$	Avg. Deg	Avg. Events/T	Gini(E)	Gini(R)
ICEWS	ICEWS14	90,730	7,128	230	365	25.46	248.58	0.85	0.88
	ICEWS14-Synth	83,106	3,282	207	365	50.64	227.69	0.92	0.46
Hops	Only 1-hop	21,322	1,460	54	180	29.21	118.46	0.83	0.39
	Only 2-hop	21,474	1,459	71	180	29.44	119.30	0.83	0.49
	Only 3-hop	20,972	1,419	72	180	29.56	116.51	0.82	0.52
Time lag	$\mathcal{U}(1,3)$	21,438	1,464	67	180	29.29	119.10	0.827	0.458
	$\mathcal{U}(1,5)$	21,402	1,449	71	180	29.54	118.90	0.83	0.49
	$\mathcal{U}(1,8)$	21,360	1,446	66	180	29.54	118.67	0.83	0.49
E/R dist	Ent-Γ Rel-Γ	21,246	1,662	62	180	25.57	118.03	0.495	0.484
	Ent-Γ Rel-$\mathcal{U}$	14,112	1,583	76	180	17.83	78.40	0.49	0.32
	Ent-$\mathcal{U}$ Rel-Γ	21,384	1,800	65	180	23.76	118.80	0.18	0.44
	Ent-$\mathcal{U}$ Rel-$\mathcal{U}$	21,396	1,800	80	180	23.77	118.87	0.18	0.30

Table 1: Descriptive statistics for datasets (and reference ICEWS14) across our synthetic generator runs.

lies:

- **Hops** (3 datasets): We only instantiate 1-, 2-, or 3-hop patterns.
- **Time lag** (3 datasets): We vary the time lag parameter of our generator with a uniform distribution on $[1, 3]$, $[1, 5]$, or $[1, 8]$.
- **Entity/Relation Distribution** (3 datasets): We vary the entity and relation weight functions, w_E and w_R, between a uniform and gamma distribution (shape set to 1, scale to 2).

Within each family, we use calibrate the generator hyperparameters for similar descriptive statistics, with the same method used for our synthetic ICEWS calibration.

5.2 Evaluation

We evaluate models on well-known metrics for link prediction: Hits@k, with k = 1, 3, and 10. We report our results according to the sorted scores (or logprobs for LLMs) of candidate entities for a given query quadruple and calculate the rank of the correct ground-truth entity. Candidate entities are restricted to head and tail entities that appear at least once in the retrieved historical context.

5.3 Models

We conduct experiments using GPT-J-6B (Wang and Komatsuzaki, 2021) and GPT-4.1. GPT-J-6B serves as an open-source, mid-scale baseline model. GPT-4.1 represents a state-of-the-art large language model with substantially stronger reasoning capabilities.

As graph-based baselines, we use RE-Net (Jin et al., 2020), CyGNet (Zhu et al., 2021), and TLogic (Liu et al., 2022), along with two heuristic baselines: FREQUENCY, which ranks entities by historical occurrence counts in the context, and RECENCY, which ranks by most recent appearance.

We restrict historical context for our LLMs and heuristic baselines to the 25 most recent retrieved quadruples, which we found sufficient to contain full pattern instances. To avoid tokenization artifacts where LLMs truncate or conflate entity identifiers, we adopt index prompting. Each entity is prefixed with a compact index (e.g., quadruple $(1071, 20, 1493, 21)$ is presented as `21 : [0. 1071, 20, 1. 1493]` to the LLMs), which empirically improves entity disambiguation.

5.4 Prompting Methods

Here we describe our prompting methods; representative examples are given in the Appendix in Table 6.

5.4.1 Naive Prompting

For a query quadruple $q = (h, r, ?, \tau)$ we retrieve historical context using either entity-based retrieval (all past quadruples sharing head h) or pair-based retrieval (all past quadruples with the same head–relation pair (h, r)).

5.4.2 Oracle In-Context Learning (OICL)

In the oracle setting we provide each consequence with exact antecedents generated by the same underlying pattern as generated the query quadruple. Antecedents and consequence share entities and relations, so all information required to infer the answer is explicitly present in the prompt with as little noise as possible. This serves as an approximate upper bound on LLM performance on our datasets.

5.4.3 Alternative Prompting Method

Our synthetic datasets record pattern identities for every quadruple. Alternative prompting methods exploit this structure by providing the model with solved instances of the same pattern as generated the query quadruple, enabling us to test whether LLMs can generalize across surface-level entity and relation changes. But with more realistic noise in the context than in the OICL setting.

For a query q generated by pattern p we retrieve k earlier consequences of p and use them as analogies. In practice, we set k to 1. We construct a prompt consisting of the retrieved historical context for the analogies and the query event. Specifically, each analogy is paired with a retrieved history to form an in-context example, while the retrieved history of q is used to form the final query. The historical context for both analogies and the query are retrieved using one of the following strategies:

Analogy-head retrieval (AH) Historical context is retrieved based on the head entity. For each analogy and the query quadruple, we collect historical quadruples that share the same head entity, which are then used to construct the in-context example and the query, respectively.

Analogy-relation retrieval (AR) Historical context is retrieved based on relations appearing in the antecedent quadruples of the analogy. We extract all relations from the analogies' antecedents and retrieve historical quadruples involving these relations for both the analogy and the query. The retrieved quadruples are sorted by time before being used as context.

Balanced analogy-relation retrieval (BAR) To mitigate bias toward frequently occurring relations, we further introduce a balanced variant of analogy-relation retrieval. In this setting, each relation is constrained to appear with approximately equal frequency in the retrieved context. For example, if the antecedents involve relations $[r_1, r_2, r_3]$, we retrieve an equal number of instances for each relation and allocate any remaining context slot to the relation with the most recent timestamp.

5.5 Results

We focus on Hits@3 in this section, as it best reflects near-miss reasoning performance while remaining sensitive to ranking errors that are obscured by Hits@10. Our full results are available in Table 7.

Q1: Oracle in-context learning. Under OICL prompting, GPT-4.1 attains Hits@3 above 0.9 on all but one of our datasets (see Table 2), substantially outperforming all our baselines and other prompting strategies for most dataset settings. GPT-J-6B also benefits from OICL prompting, but achieves higher performance with naive entity-based retrieval for some datasets. This confirms that, when provided with perfectly aligned context, LLMs have sufficient capacity to solve our multi-hop temporal reasoning tasks; the main difficulty lies in retrieving and composing the right context rather than in performing the reasoning step itself.

Q2: Alternative prompting via analogies. Compared with OICL, the alternative prompting methods still provide partial information about a query event's underlying pattern, but are designed to evaluate LLMs' ability to identify the pattern from noisier contexts. LLMs are still able to outperform our baselines with these alternative methods, but

do not consistently outperform the naive prompting methods. This points to a limitation of LLMs, namely the ability to handle these noisier contexts as an input for pattern understanding tasks.

Q3: Effect of hop length. Figure 5 shows Hits@3 on datasets containing only 1-, 2-, or 3-hop patterns. Under naive prompting, performance decreases slightly with hop length for GPT-4.1. But this decrease is more prominent with our alternative prompting methods and OICL. In fact, OICL performance drops below naive pair-based retrieval in the only 3-hop pattern dataset, indicating some limitations in the LLM's ability to understand complex patterns. This result is consistent with the intuition that deeper reasoning chains require composing more intermediate facts and are therefore harder.

Q4: Effect of temporal dispersion. To study temporal dispersion, we vary the time-lag constraints used when generating patterns (see Figure 5). Larger lags increase the temporal distance between antecedents and consequences, making relevant events sparser in absolute time. We do not observe a strong monotonic effect of temporal lag dispersion on performance. Because the context window size is fixed, larger lags may reduce redundancy among retrieved events and increase the chance that the remaining context corresponds to the active pattern instance.

Q5: Effect of entity and relation distributions. We next vary entity and relation distributions, holding pattern structure fixed. Uniform distributions yield balanced but sparse histories, whereas Gamma distributions produce long-tailed, highly-imbalanced histories. As we can see in Figure 6, Gamma-distributed entities tend to give the best performance, with history-rich head entities presumably create dense, coherent contexts. But we observe a better performance with uniformly distributed relations, particularly for our alternative prompting methods.

6 Conclusion

We introduced a controllable synthetic benchmark for multi-hop reasoning over TKGs. By generating temporal graphs from an explicit generative process over 1-, 2-, and 3-hop patterns, we can precisely control the structure, frequency, and temporal organization of reasoning tasks. This allows us to probe how well language models and other predictors handle general multi-hop reasoning tasks in TKGs.

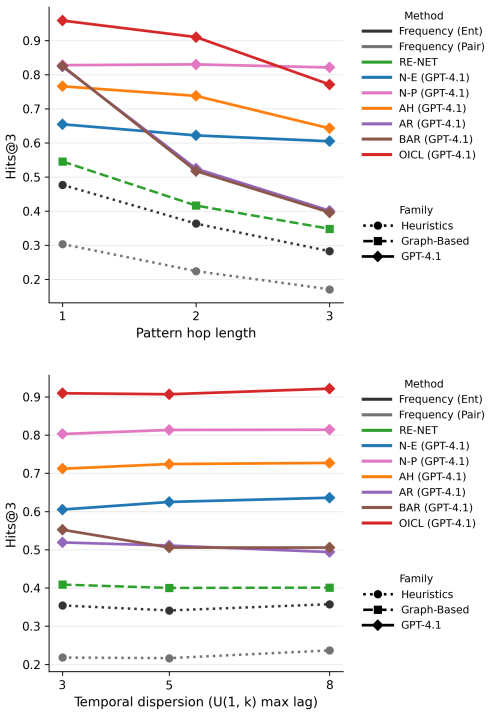


Figure 5: Performance of LLM methods under structural variation. **Top:** Pattern hop length (datasets composed entirely of 1-, 2-, or 3-hop patterns). **Bottom:** Temporal dispersion, where the time between successive antecedents or the final antecedent and consequence is drawn from increasingly large uniform distributions.

Our experiments show that contemporary language models can solve a non-trivial fraction of pattern-based forecasting queries (outperforming heuristic and graph-based baselines), but their performance is highly sensitive to the prompting and retrieval strategies used to construct their inputs. Oracle in-context learning, where the model is given exactly the ground-truth antecedents, reveals a substantial gap between pattern recognition and pattern retrieval. Analogy-based prompting strategies further highlight that models often struggle to transfer knowledge across instances that share a common underlying pattern but differ in surface entities or relation distributions. These findings suggest that improved retrieval schemes and better inductive biases for compositional reasoning are needed to fully exploit the structural regularities encoded in temporal KGs. For researchers developing retrieval-augmented or hybrid neuro-symbolic TKG reasoning systems, our results highlight retrieval as the primary bottleneck under realistic

Variation	Dataset	Heuristics				Graph-Based			GPT-J-6B						GPT-4.1					
		F-E	F-P	R-E	R-P	Cy	Re	Tl	N-E	N-P	AH	AR	BAR	OICL	N-E	N-P	AH	AR	BAR	OICL
Hops	Only 1-hop	**0.448**	0.249	0.379	0.207	0.420	**0.545**	0.071	**0.806**	0.767	0.691	0.735	0.735	0.651	0.654	0.828	0.765	0.824	0.826	**0.958**
	Only 2-hop	**0.360**	0.196	0.273	0.153	0.377	**0.416**	0.090	0.741	0.668	0.574	0.454	0.457	**0.754**	0.622	0.830	0.738	0.524	0.517	**0.910**
	Only 3-hop	**0.298**	0.148	0.213	0.108	0.291	**0.348**	0.097	0.686	0.636	0.490	0.402	0.383	**0.753**	0.604	**0.821**	0.642	0.401	0.396	0.771
Time lag	$\mathcal{U}(1,3)$	**0.358**	0.191	0.255	0.144	0.327	**0.409**	0.096	0.686	0.683	0.531	0.485	0.495	**0.757**	0.605	0.803	0.712	0.519	0.552	**0.909**
	$\mathcal{U}(1,5)$	**0.338**	0.182	0.240	0.145	0.334	**0.400**	0.086	0.726	0.682	0.559	0.471	0.471	**0.763**	0.625	0.813	0.724	0.511	0.506	**0.907**
	$\mathcal{U}(1,8)$	**0.363**	0.207	0.259	0.160	0.365	**0.401**	0.093	0.747	0.695	0.551	0.489	0.484	**0.761**	0.636	0.814	0.727	0.494	0.505	**0.921**
E/R dist	Ent-Γ Rel-Γ	**0.245**	0.034	0.174	0.034	0.213	**0.338**	0.012	**0.899**	0.604	0.812	0.497	0.505	0.767	0.781	0.880	**0.940**	0.583	0.594	0.903
	Ent-Γ Rel-$\mathcal{U}$	**0.273**	0.024	0.224	0.024	0.210	**0.267**	0.011	**0.875**	0.549	0.835	0.588	0.597	0.809	0.781	0.916	**0.951**	0.693	0.682	0.907
	Ent-$\mathcal{U}$ Rel-Γ	**0.203**	0.019	0.164	0.019	0.146	**0.219**	0.011	**0.886**	0.496	0.825	0.504	0.498	0.816	0.777	0.906	**0.945**	0.597	0.617	0.925
	Ent-$\mathcal{U}$ Rel-$\mathcal{U}$	**0.218**	0.008	0.189	0.008	0.127	**0.262**	0.010	**0.897**	0.489	0.848	0.533	0.522	0.803	0.759	0.917	**0.954**	0.641	0.628	0.933

Table 2: Performance (hit@3) across synthetic datasets and methods. **Legend**: (F-E) Frequency entity-based retrieval, (F-P) Frequency pair-based, (R-E) Recency entity-based, (R-P) Recency pair-based, (Cy) CyGNet, (Re) RE-Net, (Tl) Tlogic, (N-E) Naive prompting entity-based, (N-P) Naive prompting pair-based, (AH) Analogy-head retrieval, (AR) Analogy-relation retrieval, (BAR) Balanced analogy-relation retrieval, (OICL) Oracle In-Context Learning. Values are **bolded** where they are the highest for a given model group and underlined for the second-highest.

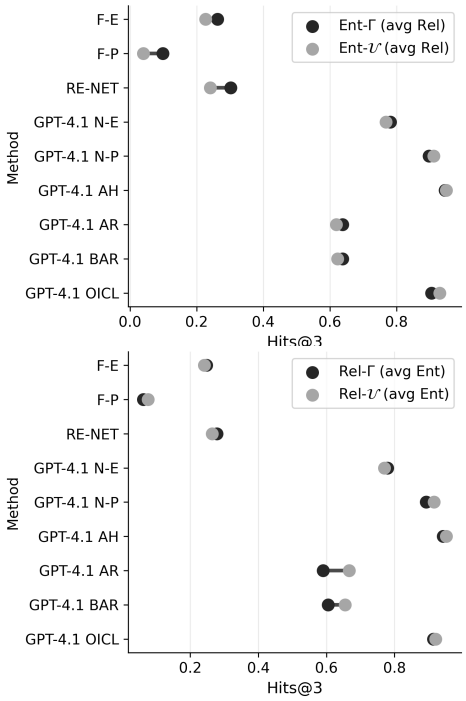


Figure 6: Effect of entity and relation distribution shifts on performance (Hits@3). **Top:** Entity distribution effect, averaging over relation distributions. **Bottom:** Relation distribution effect, averaging over entity distributions. Entities and relations are drawn from either Gamma or Uniform distributions during instantiation.

conditions, rather than reasoning.

Our benchmark is intentionally minimal and synthetic, which makes it easy to analyze but leaves many avenues for future work. Extending the pattern family to more complex graph motifs, incorporating semantically meaningful entities and relations, and aligning the synthetic tasks more closely with real-world temporal reasoning challenges are promising directions. We also expect that jointly learning retrieval and prediction, as well as integrating explicit pattern-aware modules, could narrow the gap between oracle and realistic in-context performance. We hope that the proposed benchmark will serve as a test bed for such methods and help clarify the capabilities and limitations of current models for multi-hop temporal reasoning.

Limitations

Our evaluation is focused on synthetic TKGs from controllable pattern distributions. This allows us to isolate multi-hop reasoning behavior, but does not capture all the semantic and structural properties of real-world TKGs. In particular, while previous work (Lee et al., 2023) has found that LLMs maintain predictive performance without semantics, intuitively it makes sense to examine the relationship between semantics and pattern understanding.

We focus on a finite set of 1-, 2-, and 3-hop patterns with relatively simple connectivity. Real-world reasoning may involve longer chains and richer compositional structure, i.e., conjunctions of patterns. Our conclusions about multi-hop generalization therefore apply to this controlled family and may fall apart on more complex reasoning patterns.

We only consider two LLM models: open-source GPT-J-6B and closed-source GPT-4.1. This does not account for the full variety of LLMs available with varying capabilities, but does represent a starting point.

AI Usage Disclosure

We used AI-assisted tools during the preparation of this manuscript for purposes that do not affect the scientific content or conclusions. Specifically, AI tools were used to assist with data visualization and plotting, LaTeX formatting, and proofreading for clarity and grammar. All methodological decisions, experimental design, analyses, and interpretations were conceived and verified by the authors, who take full responsibility for the content of this paper.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. 2024. Hopping too late: Exploring the limitations of large language models on multi-hop queries. *arXiv preprint arXiv:2406.12775*.

Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, James Starz, and Michael Ward. 2015. Icews coded event data. *Harvard Dataverse*, 12(2).

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. 2020. Recurrent event network: Autoregressive structure inferenceover temporal knowledge graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6669–6683, Online. Association for Computational Linguistics.

Mohammad Khodadad, Ali Shiraee Kasmaee, Mahdi Astaraki, Nicholas Sherck, Hamidreza Mahyar, and Soheila Samiee. 2025. Evaluating multi-hop reasoning in large language models: A chemistry-centric case study. *arXiv preprint arXiv:2504.16414*.

Dong-Ho Lee, Kian Ahrabian, Woojeong Jin, Fred Morstatter, and Jay Pujara. 2023. Temporal knowledge graph forecasting without knowledge using in-context learning. *arXiv preprint arXiv:2305.10613*.

Kalev Leetaru and Philip A Schrodt. 2013. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA annual convention*, volume 2, pages 1–49. Citeseer.

Ningyuan Li, E Haihong, Shi Li, Mingzhi Sun, Tianyu Yao, Meina Song, Yong Wang, and Haoran Luo. 2023. Tr-rules: Rule-based model for link forecasting on temporal knowledge graph considering temporal redundancy. In *Findings of the association for computational linguistics: EMNLP 2023*, pages 7885–7894.

Yujia Li, Shiliang Sun, and Jing Zhao. 2022. Tirgn: Time-guided recurrent graph network with local-global historical patterns for temporal knowledge graph reasoning. In *IJCAI*, pages 2152–2158.

Ruotong Liao, Xu Jia, Yunpu Ma, and Volker Tresp. 2023. Gentkg: Generative forecasting on temporal knowledge graph. *arXiv preprint arXiv:2310.07793*.

Yushan Liu, Yunpu Ma, Marcel Hildebrandt, Mitchell Joblin, and Volker Tresp. 2022. Tlogic: Temporal logical rules for explainable link forecasting on temporal knowledge graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 4120–4127.

Pablo Sanchez Martin, Tarek Besold, and Priyadarshini Kumari. 2023. Fruni and ftree synthetic knowledge graphs for evaluating explainability. In *XAI in Action: Past, Present, and Future Applications*.

Christian Meilicke, Melisachew Wudage Chekol, Manuel Fink, and Heiner Stuckenschmidt. 2020. Reinforced anytime bottom up rule learning for knowledge graph completion. *arXiv preprint arXiv:2004.04412*.

Minh-Vuong Nguyen, Linhao Luo, Fatemeh Shiri, Dinh Phung, Yuan-Fang Li, Thuy Vu, and Gholamreza Haffari. 2024. Direct evaluation of chain-of-thought in multi-hop reasoning with knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2862–2883.

Harshay Shah, Badih Ghazi, Yangsibo Huang, Ravi Kumar, Da Yu, and Chiyuan Zhang. Do language models robustly acquire new knowledge? In *AI That Keeps Up: NeurIPS 2025 Workshop on Continual and Compatible Foundation Model Updates*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

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

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Jian Wu, Linyi Yang, Manabu Okumura, and Yue Zhang. 2024a. Mrke: The multi-hop reasoning evaluation of llms by knowledge edition. *CoRR*.

Jian Wu, Linyi Yang, Zhen Wang, Manabu Okumura, and Yue Zhang. 2024b. Cofca: A step-wise counterfactual multi-hop qa benchmark. *arXiv preprint arXiv:2402.11924*.

Jiapeng Wu, Meng Cao, Jackie Chi Kit Cheung, and William L Hamilton. 2020. Temp: Temporal message passing for temporal knowledge graph completion. *arXiv preprint arXiv:2010.03526*.

Yuwei Xia, Ding Wang, Qiang Liu, Liang Wang, Shu Wu, and Xiao-Yu Zhang. 2024. Chain-of-history reasoning for temporal knowledge graph forecasting. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16144–16159.

Wenjie Xu, Ben Liu, Miao Peng, Xu Jia, and Min Peng. 2023a. Pre-trained language model with prompts for temporal knowledge graph completion. *arXiv preprint arXiv:2305.07912*.

Yi Xu, Junjie Ou, Hui Xu, and Luoyi Fu. 2023b. Temporal knowledge graph reasoning with historical contrastive learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 4765–4773.

Sohee Yang, Nora Kassner, Elena Gribovskaya, Sebastian Riedel, and Mor Geva. 2025. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3971–3992.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Cunchao Zhu, Muhao Chen, Changjun Fan, Guangquan Cheng, and Yan Zhang. 2021. Learning from history: Modeling temporal knowledge graphs with sequential copy-generation networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4732–4740.

A Validation of Synthetic TKG Generator

As an external validity check on our generator, we compare model behavior on real temporal event data versus its synthetic analogue. To construct the synthetic analogue (ICEWS14-Synth), we calibrate generator hyperparameters so that the resulting synthetic TKG matches ICEWS14 along descriptive statistics and heuristic baseline behavior (see Section D.1, Algorithms 12 to 13). We then evaluate whether relative method ordering is preserved when moving from ICEWS14 to ICEWS14-Synth. Figure 7 shows that while absolute scores shift, the relative gaps between entity-based and pair-based retrieval are largely preserved, suggesting that our controllable generator produces a realistic context distribution for this slice of TKG reasoning.

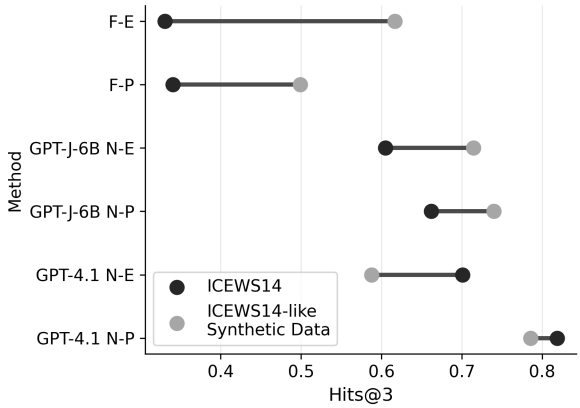


Figure 7: ICEWS14 → ICEWS14-Synth shift in Hits@3 by method. Comparison of Frequency baseline, RE-NET, and GPT-4.1 performance on the original ICEWS14 benchmark and our synthetic ICEWS14-like data.

B Justification for Pattern Choices

Pattern Control Parameters. Each pattern $p = (A_p, c_p, \Delta_p)$ in our generator is associated with three families of control parameters: (i) *structural parameters*, including hop length and entity overlap regime; (ii) *temporal parameters*, defined by per-step lag intervals Δ_p; and (iii) *instantiation parameters*, including forcing probability $\rho_{\text{force}}(p)$ and forcing frequency $n_{\text{force}}(p)$.

These parameters allow us to independently control the depth of relational composition (via hop length), the temporal dispersion of evidence (via lag intervals), and the density and salience of historical context (via instantiation controls). This disentanglement enables targeted evaluation of multi-hop reasoning behavior along a single axis at a time, as reflected in the experimental sweeps in Section 5.5 and Figure 5 and Figure 6.

Relation to Mined KG Patterns. Although our generator does not enumerate all possible logical rules that may arise in real-world KGs, prior work on static and temporal pattern mining (Meilicke et al., 2020; Liu et al., 2022; Li et al., 2023) consistently recovers a small set of recurring structural motifs: transitive closures, inverses, reinforcements, bridges, and cyclic patterns. These motifs are directly represented in our pattern taxonomy

(Table 4), up to temporal ordering and relation orientation.

Our goal is not exhaustive rule coverage, but controlled evaluation. By restricting to a finite yet expressive family of commonly observed motifs, we trade completeness for interpretability and experimental control, which is essential for isolating failure modes in multi-hop reasoning.

Pattern roadmap. Table 4 enumerates the structural *shape templates* used by our generator (1-, 2-, and 3-hops), which define the variable-sharing structure between antecedents and consequence. Tables 3a to 5b then describe the orthogonal control regimes we apply on top of these shapes: entity reuse (Table 3a), relation reuse (Table 3b), temporal lag constraints (Table 5a), and global validity constraints (Table 5b). Together, these knobs allow us to vary one factor at a time while holding the others fixed.

Category	Description	Logical form
Fully reused	Consequence uses exactly the same two entities as (at least) one antecedent edge.	$(e_a, r_1, e_b, t_1) \rightarrow (e_a, r_c, e_b, t_2)$ or (e_b, r_c, e_a, t_2)
Bridge reuse	Consequence uses one entity from each antecedent component; enables patterns where antecedents are disjoint.	$(e_1, r_1, e_2, t_1) \wedge (e_3, r_2, e_4, t_2) \rightarrow (e_2, r_c, e_3, t_3)$

(a) Entity overlap regimes: how the consequence reuses variables from the antecedent. We do not allow the consequence to introduce entities that are not found in at least one antecedent of the pattern.

Category	Description	Logical form
Copy consequence relation	Consequence relation is reused from the antecedent set ($r_c \in \{r_i\}$).	$(e_1, r_1, e_2, t_1) \wedge (e_2, r_2, e_3, t_2) \rightarrow (e_1, r_2, e_3, t_3)$
Novel consequence relation	Consequence relation is not in antecedent relations ($r_c \notin \{r_i\}$), allowed when that flag is False.	$(e_1, r_1, e_2, t_1) \wedge (e_2, r_2, e_3, t_2) \rightarrow (e_1, r_3, e_3, t_3)$
Homogeneous antecedents	Antecedent relations are all the same or dominated by one type (tests "same-relation" compositionality).	$(e_1, r_1, e_2, t_1) \wedge (e_2, r_1, e_3, t_2) \rightarrow (\cdot, r_c, \cdot, t_3)$
Heterogeneous antecedents	Antecedent relations differ (tests multi-relation composition).	$(e_1, r_1, e_2, t_1) \wedge (e_2, r_2, e_3, t_2) \wedge (e_3, r_3, e_4, t_3) \rightarrow (\cdot, r_c, \cdot, t_4)$

(b) Relation reuse regimes: whether the consequence predicate and antecedent predicates are reused or novel.

Table 3: Control regimes for pattern construction (entity reuse and relation reuse).

Algorithm 2 Pattern Instantiation

procedure INSTANTIATEPAT($p, \pi, \tau_{\text{start}}$)
Require: Pattern $p = (A_p, c_p, \Delta_p)$ of length k
Require: Entity assignment $\pi : V \to E$
Require: Anchor time $\tau_{\text{start}} \in T$
  Let $A_p = \big((x_1, r_1, y_1), \ldots, (x_k, r_k, y_k)\big)$
  Let $\Delta_p = \big([\delta_1^-, \delta_1^+], \ldots, [\delta_k^-, \delta_k^+]\big)$
  $\tau_1 \leftarrow \tau_{\text{start}}$ ▷ First antecedent time
  for $i \leftarrow 1, \ldots, k$ **do**
    Sample temporal gap g_i uniformly from $[\delta_i^-, \delta_i^+] \subset \mathbb{N}$
    $\tau_{i+1} \leftarrow \tau_i + g_i$
  end for
  for $i \leftarrow 1, \ldots, k$ **do**
    $(x_i, r_i, y_i) \leftarrow A_p[i]$
    $q_i \leftarrow (\pi(x_i), r_i, \pi(y_i), \tau_i)$
  end for
  Let $c_p = (x_c, r_c, y_c)$
  $q_{k+1} \leftarrow (\pi(x_c), r_c, \pi(y_c), \tau_{k+1})$
  $Q_p \leftarrow \{q_1, \ldots, q_{k+1}\}$
  return $Q_p, \{\tau_i\}_{i=1}^{k+1}$
end procedure

C Synthetic TKG Generator Utilities

We implement generation in two stages. First, we sample and insert pattern instances over time to form the temporal graph (Algorithm 4). Second, we support two ways of producing consequences: *forced instantiation* (directly instantiating a sampled pattern via Algorithm 2) and *spontaneous instantiation* (detecting satisfied antecedents in the existing graph via Algorithms 5–7, then emitting the implied consequence). These utilities are written to make the generator modular: alternative pattern families or lag distributions can plug into the same instantiation and matching routines.

Algorithm 4 is the main driver of the synthetic temporal knowledge graph generator. It iterates over time and over a predefined pattern set, injecting forced pattern instances while also emitting spontaneous consequences when antecedent configurations are detected in the evolving graph. This design allows explicit control over pattern frequency while preserving emergent temporal structure.

Algorithm 2 instantiates a pattern template into concrete temporal quadruples. Given an entity assignment and a starting time, it samples temporal gaps from the pattern's lag intervals and produces a consistent sequence of antecedent and consequence events. This routine is shared by both forced and spontaneous instantiation paths.

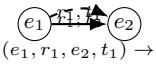
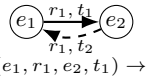
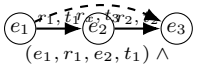
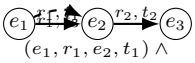
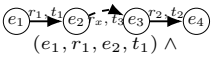
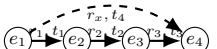
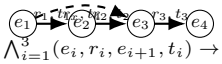
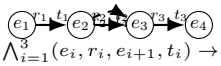
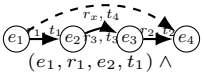
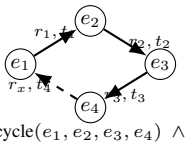
Hops: Name	Description	Illustration	Hops: Name	Description	Illustration
1-hop: Self-entailment	Antecedent and consequence share the same endpoints (e.g., reinforcement, repetition).	 $(e_1, r_1, e_2, t_1) \rightarrow (e_1, r_1, e_2, t_2)$	**1-hop: Inverse**	Same endpoints, reversed direction (e.g., reciprocation, reply).	 $(e_1, r_1, e_2, t_1) \rightarrow (e_2, r_1, e_1, t_2)$
1-hop: Reflexivity	At least one triple has $h = t$.	 $(e_1, r_1, e_1, t_1) \rightarrow (e_1, r_1, e_1, t_2)$	**2-hop: Transitive Closure**	Antecedents share one entity; consequence closes the triangle by connecting open ends.	 $(e_1, r_1, e_2, t_1) \wedge (e_2, r_2, e_3, t_2) \rightarrow (e_1, r_x, e_3, t_3)$
2-hop: Reinforcement of Premise	Antecedents share one entity; consequence repeats one antecedent edge.	 $(e_1, r_1, e_2, t_1) \wedge (e_2, r_2, e_3, t_2) \rightarrow (e_1, r_1, e_2, t_3)$	**2-hop: Bridge**	Antecedents are disjoint components; consequence must connect them.	 $(e_1, r_1, e_2, t_1) \wedge (e_3, r_2, e_4, t_2) \rightarrow (e_2, r_x, e_3, t_3)$
2-hop: Multi-derivation	Both antecedents use the same unordered entity pair.	 $(e_1, r_1, e_2, t_1) \wedge (e_1, r_2, e_2, t_2) \rightarrow (e_1, r_x, e_2, t_3)$	**3-hop: Multi-step Transitive Closure**	Three antecedents form a length-3 path; consequence closes cycle by connecting endpoints.	 $\bigwedge_{i=1}^{3}(e_i, r_i, e_{i+1}, t_i) \rightarrow (e_1, r_x, e_4, t_4)$
3-hop: Shortcut Entailment	Length-3 path; consequence links an endpoint to an interior node.	 $\bigwedge_{i=1}^{3}(e_i, r_i, e_{i+1}, t_i) \rightarrow (e_1, r_x, e_3, t_4)$	**3-hop: Premise Re-derivation**	Length-3 path; consequence repeats one earlier edge (reinforces a subedge).	 $\bigwedge_{i=1}^{3}(e_i, r_i, e_{i+1}, t_i) \rightarrow (e_2, r_2, e_3, t_4)$
3-hop: Bridging Composition	Early antecedents are disjoint; later antecedent or consequence join them.	 $(e_1, r_1, e_2, t_1) \wedge (e_3, r_2, e_4, t_2) \wedge (e_2, r_3, e_3, t_3) \rightarrow (e_1, r_x, e_4, t_4)$	**3-hop: Cyclic Entailment**	All four edges form a single simple cycle.	 cycle$(e_1, e_2, e_3, e_4) \wedge$ Consequence completes cycle

Table 4: Shape taxonomy of our synthetic data generator. This covers all of our valid pattern shapes, up to variation in relation orientation and temporal ordering of the quadruples. **Legend**: Antecedents are solid; consequence is dashed.

Algorithm 5 identifies all satisfied antecedent configurations of a given pattern at a specific time. It anchors on candidate consequence events and performs a backward temporal search to recover antecedent bindings that respect relation constraints, temporal windows, and variable consistency. The output enables spontaneous consequence emission without explicitly forcing pattern instances.

Algorithm 6 performs recursive backtracking over antecedent positions to enumerate consistent variable bindings. At each step, it restricts candidate edges using temporal constraints and already-bound entities, pruning inconsistent assignments early. This procedure ensures exact matching of multi-hop temporal patterns.

Algorithm 7 retrieves candidate temporal edges for a single antecedent position under a relation constraint, time window, and partial variable assignment. By querying a relation-indexed, time-sorted structure, it avoids scanning the full graph and enables efficient backtracking for antecedent matching.

Category	Description	Logical form
Deterministic / tight	Each lag interval is a single value ($\delta_i^- = \delta_i^+$), producing fixed-delay patterns.	$(\cdot, t_1) \wedge (\cdot, t_2) \to (\cdot, t_3)$ with $t_2 = t_1 + \delta,\ \ t_3 = t_2 + \delta'$
Loose / interval	Lags are wide intervals; valid instances tolerate timing noise: $t_{i+1} - t_i \in [\delta_i^-, \delta_i^+]$.	$t_2 - t_1 \in [\delta_1^-, \delta_1^+], \quad t_3 - t_2 \in [\delta_2^-, \delta_2^+]$
Front-loaded	Early lag(s) are larger in expectation than later lag(s): "slow setup then fast trigger."	$\mathbb{E}[\delta_1] > \mathbb{E}[\delta_2]$ (e.g., $[5, 10]$ then $[0, 2]$)
Back-loaded	Later lag(s) larger than early: "fast evidence then slow outcome."	$\mathbb{E}[\delta_1] < \mathbb{E}[\delta_2]$ (e.g., $[0, 2]$ then $[5, 10]$)
Recency vs long-range	All lags small (recency window) vs some large (long-range dependency).	Recency: $\forall i, \delta_i^+ \leq W$; Long-range: $\exists i, \delta_i^+ \gg W$

(a) Temporal constraint regimes induced by the per-step interval sequence Δ_p.

Category	Description	Formal Constraint
No new entities in consequence	Consequence entity set must be a subset of antecedent entity set (prevents exogenous "teleporting" entities).	$\{h_c, t_c\} \subseteq \bigcup_i \{h_i, t_i\}$
No self-connections (optional)	Exclude triples with $h = t$ (removes self-loops / collapsed edges).	$\forall$ triples $(h, r, t) :\ h \neq t$
No duplicate triples (optional)	Require all triples in the pattern to be distinct (removes degenerate repetitions).	$\lvert\{\text{antecedent triples}\} \cup \{\text{consequence triple}\}\rvert = k + 1$
2-hop connectivity	Require antecedents connected OR consequence bridges disjoint antecedents.	$(E_1 \cap E_2 \neq \emptyset) \ \vee\ \big(h_c \in E_1 \wedge t_c \in E_2\big) \ \vee\ \big(h_c \in E_2 \wedge t_c \in E_1\big)$
3-hop rescue connectivity	Allow patterns where the third antecedent or the consequence restores connectivity among earlier parts (join).	If early antecedents are disjoint, require a later edge to connect components.
Single-cycle enforcement (optional)	Enforce that the edge multiset forms one simple cycle.	$\texttt{forms\_single\_cycle}(\{\{h_i, t_i\}\}_{i=1}^k \cup \{\{h_c, t_c\}\}) = \texttt{True}$
Subpattern filtering at instantiation	Avoid instantiating patterns that are subpatterns of previously selected ones (encourages diversity).	Reject p if $\exists p' \in \mathcal{P}$ such that $p \subseteq p'$ (ignoring timing).

(b) Constraint regime: generator "knobs" that define which motifs are considered and which are excluded.

Table 5: Temporal constraints and more general constraints in our synthetic TKG generating process.

Algorithm 3 GENERATENHOPPATTERNS

Require: $n_{\text{hops}} \in \{1, 2, 3\}$; flags REQUIREUNIQUE, PROHIBITSELFCONNECTIONS, PROHIBITNEW-CONSEQUENCERELATIONS, REQUIRESEQUENTIALRULE

Ensure: Table PatternTable of canonical 1-, 2-, or 3-hop templates

Set $(n_E, n_R) \leftarrow (2, 2)$ if $n_{\text{hops}} = 1$; $(4, 3)$ if $n_{\text{hops}} = 2$; $(5, 4)$ if $n_{\text{hops}} = 3$
$\mathcal{E} \leftarrow \{e_1, \ldots, e_{n_E}\}$, $\mathcal{R} \leftarrow \{r_1, \ldots, r_{n_R}\}$
$\mathcal{T} \leftarrow \{(h, r, t) : h, t \in \mathcal{E}, r \in \mathcal{R}\}$ ▷ all possible triples
$\mathcal{P} \leftarrow \emptyset$ ▷ set of canonical patterns
PatternTable $\leftarrow$ empty table
if $n_{\text{hops}} = 1$ **then**
    for all $a_1 \in \mathcal{T}$ **do**
        $\mathcal{E}_A \leftarrow \{a_1.\text{h}, a_1.\text{t}\}$
        for all $c \in \mathcal{E}_A \times \mathcal{R} \times \mathcal{E}_A$ **do**
            if IS-VALID-1-HOP-PATTERN(a_1, c) with current flags **then**
                $(a_1', c') \leftarrow$ CANONICALIZE(a_1, c)
                if $(a_1', c') \notin \mathcal{P}$ **then**
                    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(a_1', c')\}$
                    append row for (a_1', c') to PatternTable
                end if
            end if
        end for
    end for
else if $n_{\text{hops}} = 2$ **then**
    for all $(a_1, a_2) \in \mathcal{T} \times \mathcal{T}$ **do**
        $\mathcal{E}_A \leftarrow$ entities in a_1 and a_2
        for all $c \in \mathcal{E}_A \times \mathcal{R} \times \mathcal{E}_A$ **do**
            if IS-VALID-2-HOP-PATTERN(a_1, a_2, c) with current flags **then**
                $(a_1', a_2', c') \leftarrow$ CANONICALIZE(a_1, a_2, c)
                if $(a_1', a_2', c') \notin \mathcal{P}$ **then**
                    add (a_1', a_2', c') to $\mathcal{P}$ and to PatternTable
                end if
            end if
        end for
    end for
else if $n_{\text{hops}} = 3$ **then**
    for all $(a_1, a_2, a_3) \in \mathcal{T}^3$ **do**
        $\mathcal{E}_A \leftarrow$ entities in a_1, a_2, a_3
        for all $c \in \mathcal{E}_A \times \mathcal{R} \times \mathcal{E}_A$ **do**
            if IS-VALID-3-HOP-PATTERN(a_1, a_2, a_3, c) with current flags **then**
                $(a_1', a_2', a_3', c') \leftarrow$ CANONICALIZE(a_1, a_2, a_3, c)
                if $(a_1', a_2', a_3', c') \notin \mathcal{P}$ **then**
                    add (a_1', a_2', a_3', c') to $\mathcal{P}$ and to PatternTable
                end if
            end if
        end for
    end for
end if
return PatternTable

Algorithm 4 Synthetic TKG Generation

procedure GENERATETKG($|E|, |R|, |T|, w_E, w_R, \mathcal{P}, \rho_{\text{force}}, n_{\text{force}}$)
Require: $|E| > 0,\ |R| > 0,\ |T| > 0$
  $E \leftarrow \{1, \ldots, |E|\}, R \leftarrow \{1, \ldots, |R|\}, T \leftarrow \{0, \ldots, |T| - 1\}$ ▷ Initialize vocabularies
  $G \leftarrow \emptyset$
  Initialize label map $\ell(q) \leftarrow \emptyset$ for all $q \in E \times R \times E \times T$
  for $\tau \in T$ **do** ▷ Iterate over time
    for $p \in \mathcal{P}$ **do** ▷ Pattern-driven structure
      $k \leftarrow$ hop-length of p
      $(A_p, c_p, \Delta_p) \leftarrow p$ ▷ A_p antecedents, c_p consequence
      for $j \leftarrow 1, \ldots, n_{\text{force}}(p)$ **do** ▷ Forced instantiations of p
        Draw $u \sim \text{Uniform}[0, 1]$
        if $u < \rho_{\text{force}}(p)$ **then**
          Sample entity map $\pi_p : V \to E$ by drawing each $\pi_p(v) \sim W_E$
          $(Q_p, \{\tau_i\}_{i=1}^{k+1}) \leftarrow$ INSTANTIATEPAT(p, π_p, τ) ▷ Get quadruples and lags
          for $q \in Q_p$ **do**
            if $\tau(q) \in T$ **then**
              $G \leftarrow G \cup \{q\}$
              $\ell(q) \leftarrow \ell(q) \cup \{p\}$
            end if
          end for
        end if
      end for
      $I_{\text{ante}} \leftarrow$ GETSATISFIEDANTECEDENTS(G, p, τ) ▷ Spontaneous instantiations
      for $(\pi, \{\tau_i\}_{i=1}^{k}) \in I_{\text{ante}}$ **do**
        Let Δ_p's last interval be $[\delta_k^-, \delta_k^+]$
        Sample τ_{k+1} uniformly from $[\tau_k + \delta_k^-, \tau_k + \delta_k^+]$
        if $\tau_{k+1} \in T$ **then**
          Let $c_p = (x_c, r_c, y_c)$
          $q \leftarrow (\pi(x_c), r_c, \pi(y_c), \tau_{k+1})$
          $G \leftarrow G \cup \{q\}$
          $\ell(q) \leftarrow \ell(q) \cup \{p\}$
        end if
      end for
    end for
  end for
  return $G, \ell, E, R, T, \mathcal{P}$
end procedure

Algorithm 5 Enumerating Satisfied Antecedents

procedure GETSATISFIEDANTECEDENTS(G, p, τ_{curr})
Require: Temporal graph $G = (E, R, T, Q)$
Require: Pattern $p = (A_p, c_p, \Delta_p)$ of hop-length k
    Let $A_p = \big((x_1, r_1, y_1), \ldots, (x_k, r_k, y_k)\big)$
    Let $c_p = (x_c, r_c, y_c)$
    Let $\Delta_p = \big([\delta_1^-, \delta_1^+], \ldots, [\delta_k^-, \delta_k^+]\big)$ ▷ k intervals between consecutive events
    $I_{\text{ante}} \leftarrow \emptyset$
    $P \leftarrow$ placeholders appearing in A_p or c_p ▷ e.g., $e_1, e_2, \ldots$
    ▷ Anchor on candidate consequence edges at time τ_{curr}
    $C \leftarrow \{(s, r_c, o, \tau_{\text{curr}}) \in Q\}$ ▷ Candidate consequence quadruples
    for all $(s, r_c, o, \tau_{\text{curr}}) \in C$ **do**
        $m \leftarrow$ empty partial mapping $P \rightharpoonup E$
        if x_c is a placeholder **then** ▷ Populate the mapping
            $m(x_c) \leftarrow s$
        end if
        if y_c is a placeholder **then**
            $m(y_c) \leftarrow o$
        end if
        if ALLDIFFERENT(m) is false **then** ▷ Detect degenerate mappings
            continue
        end if
        $\tau_c \leftarrow \tau_{\text{curr}}$
        BACKTRACKANTE($k, \tau_c, m, [\,]$) ▷ Backtrack over antecedents from most recent to oldest
    end for
    return I_{ante}
end procedure

Algorithm 6 Antecedent Backtracking

function BACKTRACKANTE($i, \tau_{\text{next}}, m, \boldsymbol{\tau}$)
  if $i = 0$ **then** ▷ Recursive base case
    $I_{\text{ante}} \leftarrow I_{\text{ante}} \cup \{(m, \boldsymbol{\tau})\}$
    return
  end if
  $a_i = (x_i, r_i, y_i)$
  $[\delta_i^-, \delta_i^+] \leftarrow \Delta_p[i]$
  $t_{\min} \leftarrow \tau_{\text{next}} - \delta_i^+$
  $t_{\max} \leftarrow \tau_{\text{next}} - \delta_i^-$
    ▷ Retrieve only edges consistent with relation, window, and any fixed entities
  $Q_i \leftarrow$ CANDIDATEEDGES$(G, a_i, [t_{\min}, t_{\max}], m)$
  for all $(s, r_i, o, \tau_i) \in Q_i$ **do**
    $m' \leftarrow m$
    if x_i is a placeholder and $m'(x_i)$ is undefined **then**
      $m'(x_i) \leftarrow s$
    else if x_i is a placeholder and $m'(x_i) \neq s$ **then**
      continue
    else if x_i is an entity and $x_i \neq s$ **then**
      continue
    end if
    if y_i is a placeholder and $m'(y_i)$ is undefined **then**
      $m'(y_i) \leftarrow o$
    else if y_i is a placeholder and $m'(y_i) \neq o$ **then**
      continue
    else if y_i is an entity and $y_i \neq o$ **then**
      continue
    end if
    if ALLDIFFERENT(m') is false **then**
      continue
    end if
    $\boldsymbol{\tau}' \leftarrow (\tau_i) \,\|\, \boldsymbol{\tau}$ ▷ prepend τ_i at the front
    BACKTRACKANTE$(i - 1, \tau_i, m', \boldsymbol{\tau}')$
  end for
end function

Algorithm 7 Retrieving Candidate Edges for an Antecedent

function CANDIDATEEDGES($G, a_i, [t_{\min}, t_{\max}], m$)
Require: Temporal graph $G = (E, R, T, Q)$
Require: Antecedent $a_i = (x_i, r_i, y_i)$
Require: Time window $[t_{\min}, t_{\max}] \subseteq T$
Require: Partial mapping $m : P \rightharpoonup E$ from placeholders to entities
Ensure: Set Q_i of edges (s, r_i, o, τ) that are consistent with a_i and m in the window
    if $t_{\max} < t_{\min}$ **then**
        return $\emptyset$
    end if
    $Q_i \leftarrow \emptyset$
    Let $\mathcal{I}_{r_i} = \big((s_1, r_i, o_1, \tau_1), \ldots, (s_M, r_i, o_M, \tau_M)\big)$ ▷ time-sorted index of edges with relation r_i
    Find smallest index ℓ such that $\tau_\ell \geq t_{\min}$ ▷ e.g., binary search
    Find largest index u such that $\tau_u \leq t_{\max}$
    if $\ell > u$ **then**
        return $\emptyset$
    end if
    for $j \leftarrow \ell$ to u **do**
        $(s_j, r_i, o_j, \tau_j) \leftarrow \mathcal{I}_{r_i}[j]$
        ok $\leftarrow$ **true** ▷ check head entity
        if x_i is a placeholder and $m(x_i)$ is defined **then**
            if $s_j \neq m(x_i)$ **then**
                ok $\leftarrow$ **false**
            end if
        else if x_i is a concrete entity **then**
            if $s_j \neq x_i$ **then**
                ok $\leftarrow$ **false**
            end if
        end if ▷ check tail entity
        if *ok* **then**
            if y_i is a placeholder and $m(y_i)$ is defined **then**
                if $o_j \neq m(y_i)$ **then**
                    ok $\leftarrow$ **false**
                end if
            else if y_i is a concrete entity **then**
                if $o_j \neq y_i$ **then**
                    ok $\leftarrow$ **false**
                end if
            end if
        end if
        if *ok* **then**
            $Q_i \leftarrow Q_i \cup \{(s_j, r_i, o_j, \tau_j)\}$
        end if
    end for
    return Q_i
end function

D Pattern Validity Algorithms

To ensure that generated patterns are structurally meaningful and non-degenerate, we apply a sequence of validity checks during pattern enumeration. These checks enforce connectivity, optional exclusion of self-loops, entity reuse constraints, and (optionally) single-cycle structure.

We implement our structural constraints on 1-, 2-, and 3-hop patterns using the helper routines in Algorithms 10 and 11. Algorithm 11 checks whether a set of undirected entity pairs forms a single simple cycle. Algorithms 8 to 10 apply this cycle test together with optional uniqueness and self-loop constraints to determine whether a candidate 1-, 2-, or 3-hop pattern is valid. The REQUIRE-SEQUENTIAL flag enforces that consecutive edges share at least one endpoint, ruling out disconnected permutations of the same edge multiset.

Algorithm 11 checks whether a set of entity pairs forms a single simple cycle. This constraint is optionally used to restrict pattern templates to cyclic structures, ensuring well-formed relational motifs and excluding disconnected or degenerate configurations.

Algorithm 8 applies structural and semantic constraints to candidate 1-hop patterns. It enforces optional uniqueness, prohibits self-loops when requested, ensures entity reuse between antecedent and consequence, and optionally restricts patterns to sequential or cyclic forms.

Algorithm 9 determines whether a candidate 2-hop pattern is structurally valid. In addition to basic constraints, it enforces that antecedents are either connected or are explicitly bridged by the consequence, preventing disconnected motifs that would trivialize reasoning.

Algorithm 10 extends validity checking to 3-hop patterns, accounting for multiple connectivity regimes. It allows early disjoint antecedents provided later antecedents or the consequence restore connectivity, enabling expressive but controlled multi-hop motifs while excluding structurally incoherent patterns.

D.1 Generator Calibration Algorithms

To obtain ICEWS14-Synth we calibrate the generator hyperparameters so that the resulting synthetic TKG matches a reference TKG (ICEWS14) along simple descriptive statistics and heuristic baseline performance. Algorithms 12 and 13 describe this procedure.

Algorithm 12 uses Bayesian optimization to search over hyperparameters controlling the number and type of patterns, their forcing parameters, and the entity/relation sampling distributions. For each proposed configuration, we run the generator once, compute descriptive statistics on the synthetic TKG (number of entities, relations, timestamps, edges, and degree quantiles), and evaluate the recency and frequency baselines. These measurements are combined into a scalar objective, and the optimizer (a TPE-based sampler) iteratively proposes new configurations that minimize this objective over a fixed trial budget.

Algorithm 13 defines the calibration objective used by the optimizer. The first term is a weighted sum of relative errors between synthetic and reference descriptive statistics, encouraging the generator to match the size, sparsity, and degree structure of the target TKG. The second term compares the Hits@K scores of recency and frequency baselines

Algorithm 8 IsValid1HopPattern

Require: Antecedent triple $a^{(1)} = (h_1, r_1, t_1)$
Require: Consequence triple $c = (h_c, r_c, t_c)$
Require: Booleans: RequireUnique, ProhibitSelf, ProhibitNewConsRel, RequireSequential
Ensure: **true** iff pattern is valid

$E_1 \leftarrow \{h_1, t_1\}$; $E_c \leftarrow \{h_c, t_c\}$
if RequireUnique **and** $\{a^{(1)}, c\}$ has size < 2 **then**
  return false ▷ duplicate triples
end if
if ProhibitSelf **and** $(|E_1| < 2$ **or** $|E_c| < 2)$ **then**
  return false ▷ self-loop not allowed
end if
if RequireSequential **then**
  $\mathcal{E} \leftarrow (E_1, E_c)$
  if not FormsSingleCycle$(\mathcal{E}$, RequireSequential$)$ **then**
    return false
  end if
end if
if $E_c \not\subseteq E_1$ **then**
  return false ▷ consequence entities must appear in antecedent
end if
if ProhibitNewConsRel **and** $r_c \neq r_1$ **then**
  return false ▷ consequence relation must be used in antecedent
end if
return true

Algorithm 9 IsValid2HopPattern

Require: Antecedents $a^{(1)} = (h_1, r_1, t_1)$, $a^{(2)} = (h_2, r_2, t_2)$
Require: Consequence $c = (h_c, r_c, t_c)$
Require: Booleans: RequireUnique, ProhibitSelf, ProhibitNewConsRel, RequireSequential
Ensure: **true** iff pattern is valid

$E_1 \leftarrow \{h_1, t_1\}$; $E_2 \leftarrow \{h_2, t_2\}$; $E_c \leftarrow \{h_c, t_c\}$
if RequireUnique **and** $\{a^{(1)}, a^{(2)}, c\}$ has size < 3 **then**
  return false
end if
if ProhibitSelf **and** $(|E_1| < 2$ **or** $|E_2| < 2$ **or** $|E_c| < 2)$ **then**
  return false
end if
if RequireSequential **then**
  $\mathcal{E} \leftarrow (E_1, E_2, E_c)$
  if not FormsSingleCycle$(\mathcal{E}$, RequireSequential$)$ **then**
    return false
  end if
end if
if $E_c \not\subseteq (E_1 \cup E_2)$ **then**
  return false
end if
if ProhibitNewConsRel **and** $r_c \notin \{r_1, r_2\}$ **then**
  return false
end if
$intersects \leftarrow (E_1 \cap E_2 \neq \emptyset)$
$connectsDisjoint \leftarrow ((h_c \in E_1 \wedge t_c \in E_2) \vee (t_c \in E_1 \wedge h_c \in E_2))$
if not $(intersects \vee connectsDisjoint)$ **then**
  return false ▷ antecedents must touch or be joined by consequence
end if
return true

on synthetic and reference data, penalizing configurations whose heuristic baselines behave very differently from those on the original dataset. A single hyperparameter λ controls the trade-off between matching descriptive statistics and matching baseline behavior. In practice this calibration yields synthetic TKGs that closely resemble ICEWS14 at both the structural and behavioral levels.

E Context Retrieval and Prompting Strategies

This appendix details the retrieval and prompting variants referenced in Section 5.4. Table 6 summarizes each strategy, including what keys are used for retrieval (entity, pair, relation-set) and the resulting context distribution the model sees at inference time.

F Full Experimental Results

This section reports the complete breakdown of Hits@K results across all synthetic variation axes, extending the summarized comparisons shown in the main paper. Table 7 reports the complete Hits@K results for all methods and synthetic variation settings summarized in the main text.

Algorithm 10 IsValid3HopPattern

Require: Antecedents $a^{(1)} = (h_1, r_1, t_1)$, $a^{(2)} = (h_2, r_2, t_2)$, $a^{(3)} = (h_3, r_3, t_3)$
Require: Consequence $c = (h_c, r_c, t_c)$
Require: Booleans: RequireUnique, ProhibitSelf, ProhibitNewConsRel, RequireSequential
Ensure: **true** iff pattern is valid

$E_1 \leftarrow \{h_1, t_1\}$; $E_2 \leftarrow \{h_2, t_2\}$; $E_3 \leftarrow \{h_3, t_3\}$; $E_c \leftarrow \{h_c, t_c\}$
if RequireUnique **and** $\{a^{(1)}, a^{(2)}, a^{(3)}, c\}$ has size < 4 **then**
    return false
end if
if ProhibitSelf **and** ($|E_1| < 2$ **or** $|E_2| < 2$ **or** $|E_3| < 2$ **or** $|E_c| < 2$) **then**
    return false
end if
if RequireSequential **then**
    $\mathcal{E} \leftarrow (E_1, E_2, E_3, E_c)$
    if not FormsSingleCycle($\mathcal{E}$, RequireSequential) **then**
        return false
    end if
end if
if $E_c \not\subseteq (E_1 \cup E_2 \cup E_3)$ **then**
    return false
end if
if ProhibitNewConsRel **and** $r_c \notin \{r_1, r_2, r_3\}$ **then**
    return false
end if
$connectedSeq \leftarrow ((h_2 \in E_1 \vee t_2 \in E_1) \wedge (h_3 \in (E_1 \cup E_2) \vee t_3 \in (E_1 \cup E_2)))$
if not $connectedSeq$ **then**
    $firstSecondDisjoint \leftarrow \neg(h_2 \in E_1 \wedge t_2 \in E_1)$
    $thirdConnects \leftarrow ((h_3 \in E_1 \wedge t_3 \in E_2) \vee (t_3 \in E_1 \wedge h_3 \in E_2))$
    $thirdPartialConnect \leftarrow (h_3 \in (E_1 \cup E_2) \vee t_3 \in (E_1 \cup E_2))$
    $thirdConnectFirst \leftarrow (h_3 \in E_1 \vee t_3 \in E_1)$
    $thirdConnectSecond \leftarrow (h_3 \in E_2 \vee t_3 \in E_2)$
    $consFirstThirdToSecond \leftarrow ((h_c \in (E_1 \cup E_3) \wedge t_c \in E_2) \vee (t_c \in (E_1 \cup E_3) \wedge h_c \in E_2))$
    $consSecondThirdToFirst \leftarrow ((h_c \in (E_2 \cup E_3) \wedge t_c \in E_1) \vee (t_c \in (E_2 \cup E_3) \wedge h_c \in E_1))$
    $consFirstSecondToThird \leftarrow ((h_c \in (E_1 \cup E_2) \wedge t_c \in E_3) \vee (t_c \in (E_1 \cup E_2) \wedge h_c \in E_3))$
    if $firstSecondDisjoint$ **then**
        if not $\big(thirdConnects \vee \big(thirdConnectFirst \wedge consFirstThirdToSecond\big) \vee \big(thirdConnectSecond \wedge consSecondThirdToFirst\big)\big)$ **then**
            return false
        end if
    else if not $thirdPartialConnect$ **then**
        if not $consFirstSecondToThird$ **then**
            return false
        end if
    end if
end if
return true

Algorithm 11 FORMSSINGLECYCLE

Require: Ordered edge sets $\mathcal{E} = (E_1, \ldots, E_L)$, where $E_i \subseteq \mathcal{V}$ and $|E_i| = 2$
Require: Boolean flag REQUIREADJACENCY
Ensure: **true** iff the edges form one simple cycle

if $L = 0$ **then return false**
if REQUIREADJACENCY **then**
  for $i = 1$ to L **do**
    $j \leftarrow (i \bmod L) + 1$ ▷ cyclic successor
    if $E_i \cap E_j = \emptyset$ **then**
      return false
    end if
  end for
end if
Build undirected graph $G = (\mathcal{V}, \mathcal{E}')$ where each $E_i = \{u, v\}$ induces edge (u, v)
Compute degree $\deg(v)$ for all $v \in \mathcal{V}$ in G
if $\exists v \in \mathcal{V}$ s.t. $\deg(v) \neq 2$ **then**
  return false
end if
Run DFS/BFS from an arbitrary node $v_0 \in \mathcal{V}$ to get visited set $\mathcal{V}_{\text{seen}}$
if $|\mathcal{V}_{\text{seen}}| \neq |\mathcal{V}|$ **then**
  return false ▷ graph not connected
end if
if $|\mathcal{E}'| \neq |\mathcal{V}|$ **then**
  return false ▷ simple cycle must satisfy $|E| = |V|$
end if
return true

Algorithm 12 Bayesian calibration of generator hyperparameters

Require: Reference TKG G^{ref}, base configuration template θ_0, search space $\mathcal{H}$ over generator hyperparameters, descriptive metric weights $\{\alpha_d\}$, baseline metric weight λ, set of baseline metrics $\mathcal{K}$ (e.g., recency/frequency Hits@k), evaluation budget B

Ensure: Calibrated configuration $\theta^\star$ and synthetic TKG G^{syn}

$m^{\text{ref}} \leftarrow \textsc{ComputeMetrics}(G^{\text{ref}})$ ▷ e.g., $|E|$, $|R|$, $|T|$, avg deg, quantiles
$b^{\text{ref}} \leftarrow \textsc{EvaluateBaselines}(G^{\text{ref}}, \mathcal{K})$ ▷ recency/frequency baselines on the reference TKG
$\mathcal{S} \leftarrow \textsc{InitBayesOpt}()$ ▷ initialize Bayesian optimizer (e.g., TPE sampler)
for $i = 1$ to B **do**
  $\theta_i \leftarrow \textsc{Suggest}(\mathcal{S}, \mathcal{H})$ ▷ sample hyperparameters: pattern counts, forcing rates, weight dists, ...
  $c_i \leftarrow \textsc{BuildConfig}(\theta_0, \theta_i)$ ▷ merge trial hyperparameters into base config
  $G_i^{\text{syn}} \leftarrow \textsc{GenerateTKG}(c_i)$ ▷ run generator once (Algorithm 4)
  if G_i^{syn} generation failed **then**
    $s_i \leftarrow M$ ▷ M is a large penalty score, e.g., 10^9
  else
    $m_i^{\text{syn}} \leftarrow \textsc{ComputeMetrics}(G_i^{\text{syn}})$
    $b_i^{\text{syn}} \leftarrow \textsc{EvaluateBaselines}(G_i^{\text{syn}}, \mathcal{K})$
    $e_{\text{core}} \leftarrow \sum_d \alpha_d \, \textsc{RelAbsErr}\big(m_{i,d}^{\text{syn}}, m_d^{\text{ref}}\big)$
    $e_{\text{base}} \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \big| b_{i,k}^{\text{syn}} - b_k^{\text{ref}} \big|$
    $s_i \leftarrow e_{\text{core}} + \lambda e_{\text{base}}$
  end if
  $\mathcal{S} \leftarrow \textsc{Update}(\mathcal{S}, \theta_i, s_i)$
end for
$i^\star \leftarrow \arg\min_i s_i$
$\theta^\star \leftarrow \theta_{i^\star}$; $\quad G^{\text{syn}} \leftarrow G_{i^\star}^{\text{syn}}$
return $\theta^\star, G^{\text{syn}}$

Algorithm 13 Objective used for calibration

Require: Synthetic TKG G^{syn}, reference metrics m^{ref}, reference baselines b^{ref}, weights $\{\alpha_d\}$, λ, metric set $\mathcal{D}$, baseline set $\mathcal{K}$

$m^{\text{syn}} \leftarrow \textsc{ComputeMetrics}(G^{\text{syn}})$
$b^{\text{syn}} \leftarrow \textsc{EvaluateBaselines}(G^{\text{syn}}, \mathcal{K})$
$e_{\text{core}} \leftarrow \sum_{d \in \mathcal{D}} \alpha_d \, \frac{\left| m_d^{\text{syn}} - m_d^{\text{ref}} \right|}{\max\{|m_d^{\text{ref}}|, 1\}}$
$e_{\text{base}} \leftarrow \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \big| b_k^{\text{syn}} - b_k^{\text{ref}} \big|$
return $e_{\text{core}} + \lambda e_{\text{base}}$

Method Name	Description	Context Example
Naive Entity	The context consists of historical quadruples that share the same head entity as the query. In the example, all retrieved quadruples share the same head entity 447.	138:[447, 6, 479] 138:[447, 51, 1013] ... 143:[447, 37, 1588] 144:[447, 18, ?]
Naive Pair	The context consists of historical quadruples that share both the head entity and the relation with the query. In the example, all retrieved quadruples share the same head entity 447 and relation 18.	14:[447, 18, 447] 23:[447, 18, 447] ... 137:[447, 18, 447] 144:[447, 18, ?]
Oracle In-Context Learning	Only the ground-truth antecedent triples of the query are provided as context. In the example, the consequence query 144:[447, 18, ?] has two antecedents 137:[588, 70, 470] and 140:[447, 18, 588].	137:[588, 70, 470] 140:[447, 18, 588] 144:[447, 18, ?]
Analogy-head	An analogy and the query are generated from the same pattern but involve different entities. Context for the analogy is retrieved based on the analogy's head entity, while context for the query is retrieved based on the query's head entity. In the example, the analogy 143:[817, 18, 817] and the query 144:[447, 18, ?] share the same underlying pattern.	Example: 1:[817, 41, 1640] 13:[817, 77, 378] ... 143:[817, 18, 817] New Solve: 138:[447, 6, 479] 138:[447, 51, 1013] ... 144:[447, 18, ?]
Analogy-relation	Context is retrieved based on the relations appearing in the antecedent triples of the shared pattern, for both the analogy and the query. In the example, the analogy 143:[817, 18, 817] and the query 144:[447, 18, ?] share the same pattern. Their contexts are retrieved using the relations appearing in the antecedents, namely relation 18, which leads to a context dominated by relation 18.	Example: 138:[1286, 18, 802] 138:[89, 18, 1335] ... 143:[817, 18, 817] New Solve: 139:[34, 18, 34] 139:[290, 18, 1308] ... 144:[447, 18, ?]
Balanced analogy-relation	We use the same relation-based retrieval, but explicitly balance the frequency of relations in the retrieved context. In the example, the analogy 143:[817, 18, 817] and the query 144:[447, 18, ?] share a pattern with relations 18 and 70. We construct the context so that triples with relation 18 and relation 70 appear with comparable frequency, mitigating the bias toward frequent relations.	Example: 125:[638, 70, 638] ... 142:[88, 70, 1315] 143:[817, 18, 817] New Solve: 125:[523, 70, 523] ... 143:[58, 18, 58] 143:[1468, 70, 746] 144:[447, 18, ?]

Table 6: Overview of baseline, oracle context, analogy-based retrieval and prompting strategies with illustrative examples.

Variation	Dataset	Heuristics												Graph-Based									GPT-J-6B																	
		F-E			F-P			R-E			R-P			Cy			Re			TLogic			N-E			N-P			AH			AR			BAR			OICL		
		hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10
Hops	Only 1-hop	0.327	0.448	**0.551**	0.194	0.249	0.293	0.246	0.379	0.508	0.098	0.207	0.281	0.317	0.420	0.533	0.453	0.545	**0.663**	0.018	0.071	0.154	0.535	0.806	**0.851**	0.482	0.767	0.808	0.413	0.691	0.723	0.445	0.735	0.792	0.445	0.735	0.792	0.423	0.651	0.651
	Only 2-hop	0.243	0.360	**0.473**	0.139	0.196	0.234	0.165	0.273	0.412	0.079	0.153	0.221	0.243	0.377	0.514	0.304	0.416	**0.543**	0.026	0.090	0.177	0.437	0.741	**0.810**	0.366	0.668	0.724	0.336	0.574	0.617	0.268	0.454	0.509	0.278	0.457	0.511	0.276	0.754	0.764
	Only 3-hop	0.172	0.298	**0.436**	0.100	0.148	0.194	0.125	0.213	0.358	0.056	0.108	0.176	0.152	0.291	0.420	0.197	0.348	**0.492**	0.029	0.097	0.197	0.406	0.686	0.740	0.321	0.636	0.692	0.300	0.490	0.523	0.203	0.402	0.445	0.208	0.383	0.431	0.236	0.753	**0.812**
Time lag	$\mathcal{U}(1,3)$	0.235	0.358	**0.485**	0.140	0.191	0.230	0.149	0.255	0.410	0.059	0.144	0.219	0.213	0.327	0.459	0.290	0.409	**0.547**	0.024	0.096	0.191	0.405	0.686	0.762	0.384	0.683	0.740	0.301	0.531	0.559	0.281	0.485	0.546	0.285	0.495	0.552	0.290	0.757	**0.764**
	$\mathcal{U}(1,5)$	0.223	0.338	**0.457**	0.131	0.182	0.222	0.137	0.240	0.384	0.071	0.145	0.212	0.222	0.334	0.473	0.312	0.400	**0.536**	0.026	0.086	0.171	0.417	0.726	**0.790**	0.374	0.682	0.734	0.327	0.559	0.596	0.257	0.471	0.522	0.252	0.471	0.527	0.300	0.763	0.769
	$\mathcal{U}(1,8)$	0.248	0.363	**0.482**	0.148	0.207	0.250	0.164	0.259	0.406	0.078	0.160	0.223	0.253	0.365	**0.543**	0.296	0.401	0.531	0.027	0.093	0.190	0.456	0.747	**0.815**	0.384	0.695	0.755	0.332	0.551	0.589	0.290	0.489	0.541	0.283	0.484	0.536	0.278	0.761	0.769
E/R dist	Ent-Γ Rel-Γ	0.204	0.245	**0.270**	0.023	0.034	0.035	0.081	0.174	0.263	0.019	0.034	0.035	0.132	0.213	0.244	0.310	0.338	**0.359**	0.009	0.012	0.013	0.569	0.899	**0.942**	0.214	0.604	0.627	0.568	0.812	0.832	0.244	0.497	0.553	0.241	0.505	0.570	0.281	0.767	0.779
	Ent-Γ Rel-$\mathcal{U}$	0.214	0.273	**0.296**	0.016	0.024	0.024	0.120	0.224	0.292	0.015	0.024	0.024	0.123	0.210	0.234	0.223	0.267	**0.298**	0.010	0.011	0.011	0.539	0.875	**0.943**	0.213	0.549	0.559	0.547	0.835	0.856	0.323	0.588	0.662	0.322	0.597	0.680	0.325	0.809	0.827
	Ent-$\mathcal{U}$ Rel-Γ	0.133	0.203	**0.248**	0.014	0.019	0.019	0.081	0.164	0.247	0.014	0.019	0.019	0.084	0.146	0.167	0.181	0.219	**0.247**	0.009	0.011	0.011	0.504	0.886	**0.938**	0.150	0.496	0.506	0.574	0.825	0.853	0.283	0.504	0.567	0.277	0.498	0.576	0.305	0.816	0.827
	Ent-$\mathcal{U}$ Rel-$\mathcal{U}$	0.145	0.218	**0.274**	0.006	0.008	0.009	0.099	0.189	0.272	0.006	0.008	0.009	0.071	0.127	0.151	0.226	0.262	**0.282**	0.010	0.010	0.010	0.520	0.897	**0.950**	0.150	0.489	0.493	0.582	0.848	0.875	0.281	0.533	0.611	0.274	0.522	0.607	0.275	0.803	0.813

Variation	Dataset	GPT-4.1																	
		N-E			N-P			AH			AR			BAR			OICL		
		hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10	hit@1	hit@3	hit@10
Hops	Only 1-hop	0.369	0.654	0.724	0.508	0.828	0.855	0.557	0.765	0.803	0.581	0.824	0.886	0.579	0.826	0.891	0.470	**0.958**	**0.958**
	Only 2-hop	0.371	0.622	0.691	0.451	0.830	0.866	0.558	0.738	0.770	0.354	0.524	0.585	0.344	0.517	0.579	0.347	0.910	**0.928**
	Only 3-hop	0.368	0.604	0.656	0.450	0.821	0.845	0.446	0.642	0.680	0.232	0.401	0.470	0.224	0.396	0.477	0.216	0.771	**0.888**
Time lag	$\mathcal{U}(1,3)$	0.372	0.605	0.669	0.442	0.803	0.840	0.506	0.712	0.749	0.319	0.519	0.599	0.335	0.552	0.592	0.343	0.909	**0.929**
	$\mathcal{U}(1,5)$	0.385	0.625	0.690	0.469	0.813	0.845	0.535	0.724	0.752	0.336	0.511	0.585	0.317	0.506	0.577	0.341	0.907	**0.922**
	$\mathcal{U}(1,8)$	0.395	0.636	0.698	0.466	0.814	0.848	0.531	0.727	0.758	0.304	0.494	0.562	0.314	0.505	0.568	0.346	0.921	**0.936**
E/R dist	Ent-Γ Rel-Γ	0.547	0.781	0.825	0.345	0.880	0.891	0.812	0.940	**0.952**	0.381	0.583	0.630	0.376	0.594	0.631	0.285	0.903	0.928
	Ent-Γ Rel-$\mathcal{U}$	0.526	0.781	0.845	0.353	0.916	0.921	0.829	0.951	**0.959**	0.483	0.693	0.750	0.467	0.682	0.741	0.328	0.907	0.923
	Ent-$\mathcal{U}$ Rel-Γ	0.544	0.777	0.815	0.290	0.906	0.910	0.813	0.945	**0.964**	0.409	0.597	0.647	0.414	0.617	0.674	0.299	0.925	0.941
	Ent-$\mathcal{U}$ Rel-$\mathcal{U}$	0.555	0.759	0.809	0.307	0.917	0.917	0.818	0.954	**0.970**	0.428	0.641	0.708	0.431	0.628	0.691	0.308	0.933	0.955

Table 7: Performance (hit@1/3/10) across synthetic datasets and methods. **Legend**: (F-E) Frequency entity-based retrieval, (F-P) Frequency pair-based, (R-E) Recency entity-based, (R-P) Recency pair-based, (Cy) CyGNet, (Re) RE-Net, (Tl) Tlogic, (N-E) Naive prompting entity-based, (N-P) Naive prompting pair-based, (AH) Analogy-head retrieval, (AR) Analogy-relation retrieval, (BAR) Balanced analogy-relation retrieval, (OICL) Oracle In-Context Learning. Values are **bolded** where they are the highest for a given model group and underlined for the second-highest.