TIE: A Framework for Embedding-based Incremental Temporal Knowledge Graph Completion

Jiapeng Wu[†]
McGill University, MILA
Montreal, Canada
jiapeng.wu@mail.mcgill.com

Yishi Xu[†]
University of Montreal, MILA
Montreal, Canada
yishi.xu@umontreal.ca

Yingxue Zhang
Montreal Research Center, Huawei
Noah's Ark Lab
Montreal, Canada
yingxue.zhang@huawei.com

Chen Ma[†]
McGill University
Montreal, Canada
chen.ma2@mail.mcgill.ca

Mark Coates
McGill University
Montreal, Canada
mark.coates@mcgill.ca

Jackie Chi Kit Cheung
McGill University, MILA
Montreal, Canada
jcheung@cs.mcgill.ca

ABSTRACT

Reasoning in a temporal knowledge graph (TKG) is a critical task for information retrieval and semantic search. It is particularly challenging when the TKG is updated frequently. The model has to adapt to changes in the TKG for efficient training and inference while preserving its performance on historical knowledge. Recent work approaches TKG completion (TKGC) by augmenting the encoder-decoder framework with a time-aware encoding function. However, naively fine-tuning the model at every time step using these methods does not address the problems of 1) catastrophic forgetting, 2) the model's inability to identify the change of facts (e.g., the change of the political affiliation and end of a marriage), and 3) the lack of training efficiency. To address these challenges, we present the **T**ime-aware **I**ncremental **E**mbedding (TIE) framework, which combines TKG representation learning, experience replay, and temporal regularization. We introduce a set of metrics that characterizes the intransigence of the model and propose a constraint that associates the deleted facts with negative labels.

Experimental[1] results on Wikidata12k and YAGO11k datasets demonstrate that the proposed TIE framework reduces training time by about ten times and improves on the proposed metrics compared to vanilla full-batch training. It comes without a significant loss in performance for any traditional measures. Extensive ablation studies reveal performance trade-offs among different evaluation metrics, which is essential for decision-making around real-world TKG applications.

CCS CONCEPTS

• **Computing methodologies** → **Temporal reasoning**.

KEYWORDS

Temporal Knowledge Graph; Incremental Learning

1 INTRODUCTION

Knowledge graphs (KGs), consisting of triples in the form of *(head entity, relationship, tail entity)*, are effective data structures for representing factual knowledge and lie at the core of many downstream tasks; e.g., question answering [14, 22, 40] and web search [25]. Although KGs enable powerful relational reasoning, they are usually incomplete. As such, inferring new facts based on existing ones in the KG, known as KG completion, is one of the most important tasks in KG research.

Typical KGs represent knowledge facts without incorporating temporal information, which is sufficient under some circumstances [2, 32, 37]. By additionally associating each triple with a timestamp, such as *(Obama, visit, China, 2014)*, temporal knowledge graphs (TKGs) are able to consider the temporal dynamics. Usually, TKGs are assumed to consist of discrete timestamps [16]. They can be represented as a sequence of static KG snapshots. The task of inferring missing facts across these snapshots is referred to as temporal knowledge graph completion (TKGC).

To tackle the TKGC task, two avenues of work have been explored. The first line of models induces time-dependent representation with time-agnostic decoding functions to extend static KGC methods for capturing the temporal dynamics [7, 11]. The second category of methods adopts spatial-temporal models, which leverage graph neural networks (GNNs) to capture the intra-graph structural information and inter-graph temporal dependencies [34]. We argue that there are still several areas for improvement.

arXiv:2104.08419v3 [cs.AI] 9 May 2021

First, previous methods do not explicitly formulate the incremental learning problem, where the change (addition and deletion) of historical information is incrementally available, and the model is expected to adapt to the changes while maintaining its knowledge about the historical facts. Naively, one might fine-tune the TKGC model with all available data at each new time step using gradient descent optimization. This, however, causes the model performance on the historical task to degrade quickly, a phenomenon known as *catastrophic forgetting* [24, 36], which usually occurs because the model loses track of the key static features derived from earlier data. *Second*, previous methods usually only assess overall link prediction metrics such as Hits@10 and Mean Reciprocal Rank (MRR) while omitting the dynamic aspects of the TKG performance. There is an absence of metrics that can evaluate how well a model forgets deleted facts. For example, the quadruple *(Trump, presidentOf, US, 2020)* is no longer true in 2021. Hence we would like the model to rank *Biden* higher than *Trump* given the query *(?, presidentOf, US, 2021)*. We argue that this is an essential measure of a model's effectiveness in modeling the temporal dynamics of TKGs. *Third*, as discussed in Section 3.1, previous TKGC methods [7, 11] conduct training and evaluation once across all the time steps. This does not satisfy the scalability and training efficiency requirements in real-world KG applications, where millions of entities and relations frequently update [1, 33].

Present Work. We introduce a new task, incremental TKGC, and propose TIE, a training and evaluation framework that integrates incremental learning with TKGC. TIE combines TKG representation learning, experience replay, temporal regularization to improve model performance and alleviate catastrophic forgetting.

To measure TKGC models' ability to discern facts that were true in the past but false at present, we propose new evaluation metrics dubbed *Deleted Facts Hits@10* (DF) and *Reciprocal Rank Difference Measure* (RRD). To this end, we explicitly associate deleted quadruples with negative labels and integrate them into the training process, which shows improvement upon the two metrics compared to baseline methods.

Finally, we show that training using added facts significantly improves the training speed and reduces dataset size by around ten times while maintaining a similar ranking performance level compared to vanilla fine-tuning methods.

We adapt HyTE [7] and DE [11], two existing TKGC models, to the incremental learning task on wikidata12k and YAGO11k datasets. Experiments results demonstrate that the proposed TIE framework reduces training time by about ten times and improves some of the proposed metrics compared to the full-batch training. It comes without a significant loss in any traditional measures. Extensive ablation studies reveal the performance trade-offs among different evaluation metrics, providing insights for choosing among model variations.

2 RELATED WORK

2.1 Temporal KG Completion

Existing TKGC methods can be broadly categorized into two lines of work. The first line uses shallow encoders with time-sensitive decoding functions to extend static KGC methods [7, 11, 17, 35].

For example, [7] constrains entity and relation embeddings. The decoded scores of triples lie in different hyperplanes for each timestamp. The second line of methods uses spatiotemporal models, which leverage graph neural networks (GNNs) to capture intragraph neighborhood information and temporal recurrence or attention mechanisms to capture temporal information [18, 28, 34]. The third line of methods leverages temporal point processes to deal with continuous prediction in TKGs [12, 30, 31]. However, this line of work is orthogonal to ours as their focus is the *extrapolation* task in the TKG, which aims at predicting the future interactions among entities and relations.

In our work, we aim to provide an efficient incremental learning framework for TKGC. Hence we focus on the shallow embedding methods.

2.2 Incremental Learning

As knowledge graphs evolve, more graph snapshots become available. However, deep learning models suffer from *catastrophic forgetting* when existing models are incrementally fine-tuned according to the newly available data [3, 19]. Various incremental learning techniques have been introduced to combat this issue for deep learning models. Our work is closely related to the experience replay and regularization-based methods. Experience replay, also referred to as reservoir sampling, retains an additional set of the most representative historical data. Rehearsal methods [6, 15, 26, 27] explicitly maintain a pool of historical data when training the model on new tasks. One of the earliest methods, iCarLR [27], sets the fixed number of samples for each task and selects samples that best approximate the feature mean of each class. Constrained optimization methods also belong to this category. Previous work [5, 21] exploits the stored samples to project the gradient of the current task's loss to a desired region. The objective is to ensure that the loss on the historical samples will decrease after training on the current task. This is equivalent to projecting the gradients of the current data to a direction that aligns with the gradients of the previous data. Regularization-based approaches consolidate previous knowledge by introducing regularization terms in the loss when learning on new data [3, 19, 38, 39].

More recent work has explored applying incremental learning techniques for training deep graph neural networks. GraphSAIL [36] tackles the GNN-based recommendation system's forgetting issue using knowledge distillation at both node and graph levels. ER-GNN [41] proposes node importance metrics and selects the most influential nodes in the graph as reservoir data. The model is fine tuned on the new data as well as the selected nodes during the training. A more relevant work [29] applies the regularization-based method to enrich embeddings in knowledge graphs. However, the method in [29] focuses on data synthesized by subdividing a static knowledge graph into multiple snapshots.

In our work, we propose an end-to-end framework combining experience replay and regularization-based methods that are specifically tailored for incrementally training TKGC tasks.

3 PROBLEM SETUP AND FORMULATION

In this section, we introduce notations, specify assumptions, and describe the encoder-decoder framework for the standard TKGC [34]. This is the foundation of our TIE framework for incremental TKGC.

3.1 Problem Formulation

A TKG is a sequence of KG snapshots: $\mathcal{G} = \{G^1, G^2, ..., G^T\}$, where T denotes the total number of time steps in the TKG and G^t is the KG snapshot at time step t. Each graph is represented as a triple, i.e. $G^t = (E^t, R^t, D^t)$. Here, D^t denotes the set of all *observed* quadruples (s, r, o, t) occurring at time t; E^t and R^t denote the sets of entities and relations that are involved in at least one fact in D^t. Each quadruple contains the subject s, the relation r, the object o and the time t. Let $\overline{D}^t$ denote the set of *true* quadruples at time t such that $D^t \subseteq \overline{D}^t, \forall t$. The set of missing facts can therefore be written as $D_{test}^t = \overline{D}^t \setminus D^t$.

For a quadruple $(s, r, o, t) \in D_{test}^t$ and its related object query $(s, r, ?, t)$, the goal of TKGC is ranking o as high as possible. The goal of answering a subject query $(?, r, o, t)$ is similarly defined.

Standard TKGC and incremental TKGC differ in terms of 1) the scope of the input, 2) the scope of the time steps targeted for evaluation, and 3) the set of candidate entities on which the score function is applied to produce the final ranking.

Standard TKGC. In this setting, both training and evaluation are conducted once over time steps 1 to T. During training, the model takes $D^1, D^2, \ldots, D^T$ as input and simultaneously answers queries in each of $D_{test}^1, D_{test}^2, \ldots, D_{test}^T$. The set of candidate entities are those present from the beginning to the end, i.e., $E = \bigcup_{i=1}^{T} E^i$.

Incremental TKGC. Under this setting, both training and evaluation are conducted at each time step upon the available new data D^t. Hence, the input is the sequence $D^1, D^2, \ldots, D^t$ and the goal is to answer queries in $D_{test}^1, D_{test}^2, \ldots, D_{test}^t$. As opposed to standard TKGC, the model only has access to entities present from the beginning to the current time step t, i.e., $E_{known}^t = \bigcup_{i=1}^{t} E^i$

3.2 Encoder-Decoder Framework

TeMP [34] proposes a TKGC framework with a temporal multi-relational message passing encoder and static KG decoders. However, their main focus is the encoder's design, which combines multi-relational message passing and commonly used temporal models (RNN and transformer). In TIE, we instead emphasize the time-aware embedding, which is composed of single-layer embedding matrices coupled with a time-agnostic decoding function ϕ designed for static KGC.

Encoder. Let $E \in \mathbb{R}^{|E| \times d}$ and $R \in \mathbb{R}^{|R| \times d}$ denote the entity and relation embedding matrices (d is the embedding dimension for both entities and relations), The static entity representations for entity i and relation r are defined as $z_i = E[i]$ and $z_r = R[r]$.

Temporal KG embedding models derive *time-aware* representations for entities and relations, so their temporal representations at time t are denoted as z_i^t and z_r^t. For example, the Diachronic Embedding (DE) proposed in [11] applies a time-dependent function on static entity embeddings but does not differentiate between relation embeddings in different time steps:

$$z_i^t[n] = \begin{cases} z_i[n]\sigma(w_i[n]t + b_i[n]) & if \, 1 \le n \le \gamma d, \\ z_i[n] & if \, \gamma d < n \le d. \end{cases} \quad (1)$$

Here w_i and b_i are entity-specific vectors with learnable parameters. The first γd elements of the vector capture temporal features while the last $(1 - \gamma)d$ elements capture static features. The *sine* function σ is used as the activation function enabling the model.

Decoder. Static KG models such as TransE [2], DistMult [37] and ComplEx [32] propose *time-agnostic* score functions for each triple (s, r, o). We denote these score functions by "DEC".

In the temporal KG representation learning methods, *time-dependent* representations are taken as input to the *time-agnostic* decoding function. Let ϕ^t denote the model with the parameters at time step t. The score for a quadruple (s, r, o, t) is defined as follows:

$$\phi^t(s, r, o, t) = DEC(z_s^t, z_r^t, z_o^t). \quad (2)$$

Connection to incremental TKGC. The encoder-decoder framework can be naturally adapted to incremental learning with simple fine-tuning using D^t or full-batch training using $D^1, \ldots, D^t$ at each time step. In the following sections, we define key metrics and propose a set of incremental learning techniques based on the encoder-decoder framework.

4 METRICS

We start by introducing commonly used evaluation metrics in standard TKGC, followed by the notions of current, historical average, and intransigence measures in the context of TKGC to quantify the different aspects of model capacity.

4.1 Standard TKGC Metrics

For each quadruple $(s, r, o, t) \in D_{test}^t$, we evaluate an object query $(s, r, ?, t)$ and a subject query $(?, r, o, t)$. Regarding the object query, we calculate the scores for all known entities, i.e., $\phi(s, r, o', t), \forall o' \in E^t$. The ranks are obtained by sorting the scores in descending order. Thereafter this is used to compute commonly used metrics such as Mean Reciprocal Rank (MRR) and Hits@k (k is usually 1, 3, and 10). The Hits@k is the percentage of test facts for which the correct entity's rank is at most k. For $k = 10$, we have the Hits@10 metrics, defined for object queries as:

$$\frac{1}{|D_{test}^t|} \sum_{(s,r,o,t) \in D_{test}^t} I(rank(o|s, r, t) \le 10), \quad (3)$$

where I is the indicator function.

4.2 Incremental TKGC Metrics

Since the objective of incremental TKGC is to incorporate facts from new time steps while preserving knowledge derived from the previous ones, an incremental learning approach should be evaluated based on its performance on both the *current* and *historical* quadruples. Additionally, we would like them to measure a model's ability to discern changes in the validity of facts at a different point in time, e.g., change of political affiliation or end of a marriage.

Current and Historical Average Measure. Let $\alpha_{t,j}$ be the Hits@10 value specified in Equation (3) evaluated on $D_{test}^j, (j \le t)$, using the model incrementally trained after time step t. The current performance measure (C) is written as $C_t = \alpha_{t,t}$.

We adapt the Average Accuracy Measure proposed in [4] to the TKGC setting, replacing the accuracy with the Hits@10 measure. The Average Hits@10 (A) at time step t is defined as $A_t = \frac{1}{t}\sum_{i=1}^{t}\alpha_{t,i}$. The higher the value of A_t, the better the model in terms of historical average performance, which is an important aspect for TKGC evaluation. This, to some degree, also measures whether a model is prone to *catastrophic forgetting*. A model that cannot retain past knowledge would yield a much lower A_t than a model trained using all the historical data.

Intransigence Measure. In the context of TKGC, we define intransigence as the inability of an algorithm to identify knowledge that was true in the past but false at present. For example, after graduating from a college, a student is no longer associated with the college.

We categorize the measure into the model's ability to 1) assign a low rank to the deleted facts and 2) rank the currently valid facts above the deleted facts. We propose Deleted Facts Hits@10 (*DF*) and Reciprocal Rank Difference (*RRD*) to measure the two aspects. The DF is analogous to the false positive rate in the classification setting, measuring the rank of the deleted triples' current time step as their time attributes. A lower DF value suggests that a model has a better capability to exclude deleted facts from the top 10 results.

The RRD is defined as the pairwise difference of reciprocal ranks between each positive quadruple in the test set and each deleted fact in the previous data. RRD implicitly focuses on the cases where the rank value of either the positive object o or the negative object o' is low, e.g., $\frac{1}{1} - \frac{1}{10} = 0.9$, while discounting the cases where both rank values are high, e.g. $\frac{1}{1000} - \frac{1}{1010} \approx 9.9 \times 10^{-6}$.

We define a time window ranging from $t - \tau_d$ to $t-1$ to limit the scope of evaluation. For every quadruple (s, r, o, t), we aim to find and then evaluate the related deleted facts from this time window. We define the DF and RRD metrics for object queries at time step t:

$$\mathrm{DF}_t \triangleq \frac{1}{Z_t}\sum_{(s,r,o,t)\in D_{test}^t}\sum_{o'\in O'_{s,r,t}} I(\mathrm{rank}(o'|s,r,t)\le k), \quad (4)$$

$$\mathrm{RRD}_t \triangleq \frac{100}{Z_t}\sum_{(s,r,o,t)\in D_{test}^t}\sum_{o'\in O'_{s,r,t}}\left(\frac{1}{\mathrm{rank}(o|s,r,t)}-\frac{1}{\mathrm{rank}(o'|s,r,t)}\right),$$
$$(5)$$

where $O'_{s,r,t}$ is the collection of negative objects and Z_t is the normalizing constant:

$$O'_{s,r,t} = \{o'|\exists t'\in\{t-\tau_d...t-1\}, \exists o'\in E_{known}^{t'}, (s,r,o',t')\in D^{t'}\},$$

$$Z_t = \sum_{(s,r,o,t)\in D_{test}^t}|O'_{s,r,t}|.$$

In practice, the RRD values are very close to zero. Hence we multiply the RRD by a factor of 100 for better presentation. The intransigence metrics for subject queries can be defined analogously.

5 PROPOSED FRAMEWORK: TIE

We provide an overview of TIE before describing the proposed methods in detail in the following sections.

5.1 Overview

We establish the TIE framework that augments the TKGC encoder-decoder framework (Section 3.2) with incremental learning techniques, a method to overcome intransigence, and an efficient training strategy. The overall architecture of TIE model is depicted in Figure 1. Algorithm 2 outlines the representation learning procedure of TIE.

A key insight of our framework is that we adapt experience replay and temporal regularization techniques (Sections 5.2 and 5.3) to address the catastrophic forgetting issues of fine-tuning methods using TKG representation learning models. Additionally, we propose to use the deleted facts from the recent time steps as a subset of negative training examples to address the *intransigence* issue of the state-of-the-art TKGC methods. Finally, we propose to use newly added facts only for fine-tuning at each time step. This is based on the finding that the particular type of TKGs of most interest is composed primarily of persistent facts, i.e., the average duration of facts is typically long enough that no drastic changes occur between adjacent time steps.

5.2 Experience Replay

Inspired by iCaRL [27], we propose adapting the principle of experience replay — to update the model parameters for the current task, we use not only the training data for the current time step but also the quadruples from earlier time steps. The data in the recent time steps can be made available with a replay buffer confined by a sliding window.

We denote the current time as t and the time window length for experience replay as τ. At time step t, we define a time window spanning from $\max(t-\tau, 1)$ to $t-1$. The historical facts in the most recent τ time steps are stored in memory, which also satisfies the resource constraints in a large-scale knowledge graph application, as it is too costly to store and load data from all time steps.

Line 3 of Algorithm 2 constructs the replay buffer B^t. We first introduce two strategies for replay fact sampling, then specify the loss function, which combines a standard cross-entropy loss and a knowledge distillation loss.

5.2.1 Replay Fact Sampling. We extract P^t, a set of replay samples with positive labels, from B^t, using the following sampling strategies:

Uniform Sampling. A simple yet powerful sampling strategy is to uniformly sample triples from the replay buffer B^t.

Frequency-based Sampling. We extend the notion of *pattern frequency* introduced in [34] to gauge the sampling probability of each quadruple in the time window and develop two approaches dubbed *frequency-based sampling* and *inverse frequency-based sampling*. A *pattern* of the triple (s, r, o) refers to a regular expression with some of the elements replaced with the wildcard symbol '$*$'. We use *historical pattern frequency (HPF)* and *current pattern frequency (CPF)* to represent the number of quadruples matching a pattern occurring before t (within B^t) and at t (within D^t) respectively. The set of patterns P is defined as:

$$P = \{(s, r, o), (s, *, o), (s, r, *), (*, r, o), (s, *, o), (s, *, *), (*, *, o)\}.$$
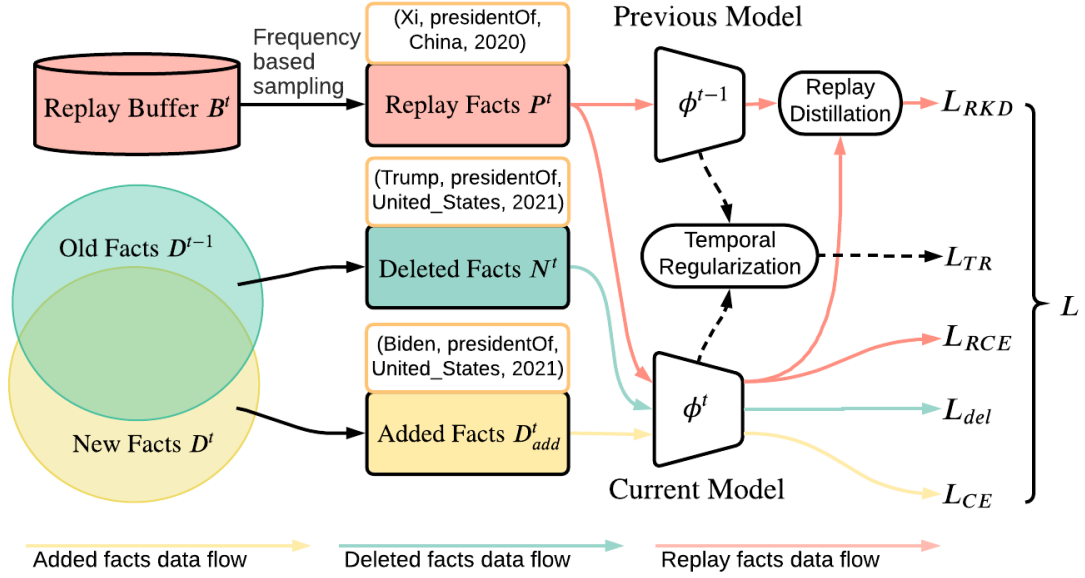
Figure 1: A high-level illustration of the full TIE model. The four types of arrows represent the process of producing different loss terms.

Taking $(s, *, o)$ for example, the HPF $h_{s,*,o}^t$ and CPF $c_{s,*,o}^t$ are calculated as:

$$h_{s,*,o}^t = |\{(s, r', o, t') | \exists r', t', (s, r', o, t') \in B^t\}|, \quad (6)$$

$$c_{s,*,o}^t = |\{(s, r', o, t) | \exists r', (s, r', o, t) \in D^t\}|. \quad (7)$$

The HPF and CPF for the rest of the patterns are analogously defined. The process for calculating the pattern frequencies over all quadruples and defining their sampling probabilities is highlighted in Algorithm 1.

The sampling probability for each quadruple (s, r, o, t') in B^t involves two separate terms: a frequency-dependent probability $fp(s, r, o)$ and a time-dependent probability $tp(t')$. We define $fp(s, r, o)$ as the weighted sum of pattern frequencies in the log scale:

$$fp(s, r, o) = \sum_{p \in P} \lambda_p \left[log(h_p^t + 1) + \gamma \tau log(c_p^t + 1) \right], \quad (8)$$

where the scalar value λ_p denotes the weight associated with the frequency of pattern p. Recall that τ is the window length. We additionally introduce γ as a discount factor controlling the ratio of h_p^t and c_p^t. We take pattern frequencies in their log forms (after adding 1 to avoid zero-values) to downscale the patterns with particularly large frequency, in order to avoid repeatedly sampling quadruples with a few very frequent patterns (examples are presented later).

The term $tp(t')$ is formulated as an exponential decay function on the temporal distance to the current time step, aiming at downweighting the quadruples from the older time steps relative to the more recent ones. It is defined as $tp(t') = \exp(\frac{t'-t}{\sigma})$, where σ is a scalar parameter controlling the smoothness of the function. The resulting unnormalized sampling rate $s(s, r, o, t')$ is written as $fp(s, r, o)tp(t')$ (line 10, Algorithm 1).

This design encourages the sampling of quadruples with *higher* pattern frequencies and discourages the sampling of quadruples with *lower* pattern frequencies. For example, in the Wikidata12k dataset, *(La Chapelle-sur-Oudon, instance of, commune of France,*

76) would get a high sampling probability because the pattern (*, *instance of, commune of France*) has a large number of matches from time steps 62 to 77. Moreover, every quadruple with this pattern is assigned a higher sampling probability, resulting in a non-representative subset sampled by the algorithm, leading to higher intransigence.

We thus propose a simple alternative that instead encourages the sampling of quadruples with *lower* pattern frequencies. We name this *inverse frequency-based sampling*, and define the sampling rate as $\frac{tp(t')}{fp(s,r,o)}$ (line 8, Algorithm 1). This design results in a more diverse set of replay samples, with the quadruples with high pattern frequencies discarded.

After computing the sampling rates for all the quadruples, we normalize the sampling probability for each quadruple and use them to sample P^t (lines 13 – 17, Algorithm 1).

5.2.2 Representation Learning. For each quadruple in P^t, we use *time-dependent negative sampling* [7] to collect a negative set of entities to approximate the cross-entropy loss. Each negative entity is collected from the set of known entities up to time step t'. For each triple $(s, r, o, t') \in P^t$, the negative entities set is written as $D_{s,r,t'}^- = \{o' | o' \in E_{known}^{t'}, (s, r, o', t') \notin D^{t'}\}$.

As in iCaRL, we additionally use the knowledge distillation loss [13] to ensure that the previously learned discriminative information is not lost during the new current learning step. Before training the parameters at time step t, we store the output of the model with the network parameters after training at time step $t-1$ as:

$$q_{s,r,o,t'}^{t-1} = \frac{\exp(\phi^{t-1}(s, r, o, t'))}{\sum_{o' \in D_{s,r,t'}^-} \exp(\phi^{t-1}(s, r, o', t'))}. \quad (9)$$

After each training iteration, we cache the output logits $q_{s,r,o,t'}$ using the current decoding function ϕ^t:

$$q_{s,r,o,t'}^t = \frac{\exp(\phi^t(s,r,o,t'))}{\sum_{o' \in D_{s,r,t'}^-} \exp(\phi^t(s,r,o',t'))}. \tag{10}$$

We use D_{KL} to denote the Kullback–Leibler divergence. The replay knowledge distillation (RKD) loss and replay cross-entropy (RCE) at each iteration are defined as follows:

$$L_{RKD} = \sum_{s,r,o,t' \in P^t} D_{KL}(q_{s,r,o,t'}^{t-1} || q_{s,r,o,t'}^t)$$
$$= \sum_{s,r,o,t' \in P^t} q_{s,r,o,t'}^{t-1} log\left(\frac{q_{s,r,o,t'}^{t-1}}{q_{s,r,o,t'}^t}\right), \tag{11}$$

$$L_{RCE} = - \sum_{s,r,o,t' \in P^t} log(q_{s,r,o,t'}^t). \tag{12}$$

Algorithm 1: Sampling Probability for Frequency-based Sampling at Time Step t

> **Input** : Replay buffer B^t and current facts D^t
> 1 **for** $(s,r,o,t') \in B^t$ **do**
> 2   Define the complete set of patterns P from equation (5.2.1) **for** $p \in P$ **do**
> 3   Calculate h_p^t and c_p^t by variations of equations (6) and (7)
> 4   **end for**
> 5   Calculate $fp(s,r,o)$ using equations (8)
> 6 $tp(t') = \exp(\frac{t'-t}{\sigma})$
> 7   **if** *inverse frequency sampling* **then**
> 8 $\psi(s,r,o,t') = \frac{tp(t')}{fp(s,r,o)}$
> 9   **else**
> 10 $\psi(s,r,o,t') = tp(t')fp(s,r,o)$
> 11   **end if**
> 12 **end for**
> 13 /* Normalize sampling probabilities for all $(s,r,o,t') \in B^t$ */
> 14 **for** $(s,r,o,t') \in B^t$ **do**
> 15 $p(s,r,o,t') = \frac{\psi(s,r,o,t')}{\sum_{\eta \in B^t} \psi(\eta)}$
> 16 **end for**
> 17 Sample P^t from B^t by the normalized probability distribution

---

5.3 Temporal Regularization

Let θ^{t-1} denote the models parameters after training at time $t-1$. We use Diachronic Embedding (DE) for demonstration, and a similar procedure can be applied to HyTE. The parameter set is $\theta^{t-1} = \{E^{t-1}, R^{t-1}, W^{t-1}, B^{t-1}\}$, where W and B are the matrices of weight and bias parameters in Equation (1). We use θ^{t-1} to initialize θ^t before training at time step t. The representations for entities and relations known at time step $t-1$ are initialized using

---

Algorithm 2: TIE Representation Learning at Time Step t

> **Input** : Quadruples in the current and historical time steps: $D^1, \ldots, D^{t-1}, D^t$
> **Require** : Model parameter θ^{t-1}
> 1 Construct the set of added quadruples D_{add}^t using Equation (17)
> 2 Initialize model parameters θ^t by Equation 13
> 3 Construct a replay buffer $B^t = \bigcup_{i=\max(1,t-\tau)}^{t-1} D^i$
> 4 Sample positive replay samples P^t by Algorithm 1
> 5 Sample a set of deleted facts N^t by Equation 15
> 6 Run network training and update model parameter with loss function (Equation 19)

---

the corresponding entry in θ^{t-1}, or with Glorot Initialization [10] if an entity appears for the first time. The initialization for the entity embedding matrix E^t is as follows:

$$E_i^t = \begin{cases} E_i^{t-1}, & \text{if } i \in E_{known}^{t-1}, \\ uniform(-\sqrt{\frac{12}{d}}, \sqrt{\frac{12}{d}}), & \text{otherwise.} \end{cases} \tag{13}$$

The initialization for other parameters are defined similarly.

Inspired by previous work [29], we propose temporal regularization on the parameter space to alleviate catastrophic forgetting. We impose an L_2 regularization constraint in the context of TKGC to smooth drastic change in the current representations compared to the previous task's parameters.

Taking the entity embedding matrix for example, we only regularize the representation for those entities that are also present in E_{known}^{t-1}. We use the hat symbol to denote such subsets in an embedding matrix, e.g., $\hat{E}^t = E^t[E_{known}^{t-1}]$. The temporal regularization loss for DE is defined as:

$$L_{TR} = ||\hat{E}^t - E^{t-1}||_2 + ||\hat{W}^t - W^{t-1}||_2$$
$$+ ||\hat{B}^t - B^{t-1}||_2 + ||\hat{R}^t - R^{t-1}||_2. \tag{14}$$

5.4 Learning with Deleted Facts

To reduce the intransigence of the model as defined in Equations (4) and (5), we propose training the model using a set of deleted quadruples from the perspective of time step t:

$$N^t = \{(s,r,o,t)|(s,r,o,t) \notin D^t \wedge \exists t', (s,r,o,t') \in B^t\}. \tag{15}$$

We associate each quadruple in N^t with a negative label and calculate the binary cross entropy loss as:

$$L_{del} = - \sum_{(s,r,o,t) \in N^t} log(1 - \sigma(\phi(s,r,o,t))), \tag{16}$$

where σ denotes the sigmoid function. We do not include knowledge distillation since the labels of $(s,r,o,t-1)$ and (s,r,o,t) are not necessarily identical.

It is optional to add a positive example (s,r,o,t') associated with each negative example (s,r,o,t) to N^t. However, we do not find it helpful experimentally in terms of alleviating intransigence.
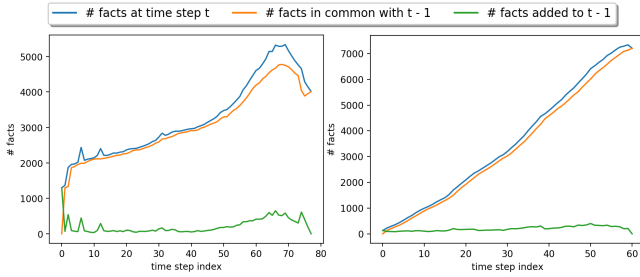
Figure 2: Dataset statistics of Wikidata12k (left) and YAGO11k(right). The three curves represent the numbers of facts (total, common and added) at every time step.

5.5 Learning with Added Facts

We observe via statistics of Wikidata12k and YAGP11k (Figure 2) that most facts are common between time steps $t-1$ and t, besides the time attributes. This suggests that fine-tuning using all the quadruples within D^t essentially re-emphasizes the majority of the facts that the model has previously seen.

We propose a novel training strategy that uses only the added facts at each time step, i.e., facts that just become true at t despite being false at $t-1$. This can significantly reduce the size of training data, thus accelerating training by orders of magnitude. It also allows us to incorporate other complementary techniques introduced in Sections 5.2 – 5.4 without sacrificing training efficiency, compared to fine-tuning with D^t alone.

The added facts at time t and the corresponding loss functions are defined as follows:

$$D_{add}^t = \{(s, r, o, t) | (s, r, o, t) \in D^t \wedge (s, r, o, t-1) \notin D^{t-1}\}, \quad (17)$$

$$L_{CE} = - \sum_{(s,r,o,t) \in D_{add}^t} log(q_{s,r,o,t}^t). \quad (18)$$

5.6 Optimization

5.6.1 Summation of loss functions. The final loss function for TIE is defined as the weighted combination of the loss terms defined in Sections 5.2–5.5:

$$L = \alpha_1 L_{CE} + \alpha_2 L_{del} + \alpha_3 L_{RCE} + \alpha_4 L_{RKD} + \alpha_5 L_{TR}, \quad (19)$$

where the α's are weight parameters controlling the relative emphasis placed on each loss term.

6 EXPERIMENTS

We evaluate the performance of our models on two standard TKGC benchmark datasets using our proposed evaluation protocol. We also conduct various ablation studies investigating the effectiveness of individual and combined components of the proposed methods.

6.1 Datasets

Common TKGs such as YAGO3 [20] and Wikidata [8] have valid time intervals associated with a subset of the facts. We use the two instances in these datasets, i.e., YAGO11k and Wikidata12k released in [7] for the experiments. The statistics of the two datasets are summarized in Table 1.

YAGO11k. This is a subset of YAGO3 [23]. In the YAGO3 dataset, the valid time of a fact is represented as a time interval, e.g., *(Pétala*

Dataset	$|E|$	$|R|$	T	$|train|$	$|valid|$	$|test|$	$|total|$
Wikidata12K	12554	24	78	257,542	20,764	19,746	298,052
YAGO11K	10623	10	61	215,894	23,197	22,567	261,658

Table 1: Statistics of Wikidata12K and YAGO11K.

Monteiro, isAffiliatedTo, Democratic Labour Party (Brazil), [1999-##-##, 2014-##-##]). We follow the same preprocessing procedures described in [7] including subgraph extraction and conversion from time interval to discrete time steps.

We also follow the practice of merging multiple adjacent time steps to balance the number of triples in different time steps, resulting in 61 different time steps in total. However, an issue with the merging is that redundant facts occur within both training and validation sets. We fix the problem by retaining only one occurrence for each quadruple in the dataset.

For a time interval with a missing start or end date, e.g., *(Ann Shoemaker, isMarriedTo, Henry Stephenson, [1956-##-##, ####-##-##]),* we use the first and the last time step in the entire dataset to represent the missing start time or end time.

Wikidata12k. This is a subset of Wikidata [8]. Similar to YAGO11k, Wikidata12k associates each fact with a time interval. We create a TKG with 77 time-steps by applying the same processing as YAGO11k.

6.2 Experiment Setting
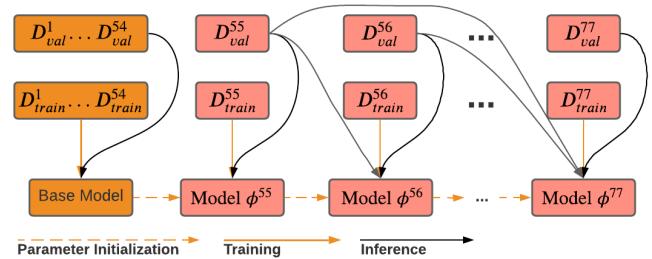
Figure 3: Training and evaluation diagram for Wikidata12K dataset.

6.2.1 Training and Evaluation Protocol. Each set of quadruples D^t is partitioned into D_{train}^t, D_{val}^t and D_{test}^t. Motivated by the experiment setting in [36], we pretrain a base model using the quadruples from the first 70% of the time steps following standard TKGC training and evaluation protocols. We then perform incremental training and evaluation on the last 30% of the time steps. Figure 3 illustrates the training and evaluation procedure for Wikidata12k.

At each time step t, we use the current performance measure C as the criterion for early-stopping on the D_{val}^t with patience set to 20. We use the best model at the current time step to compute all metrics defined in Section 4 on the test set to obtain the final performance. For evaluation, the time window length τ_d for calculating DF and RRD is set to 10.

We report results averaged across 4 randomized runs, along with standard deviation.

6.2.2 Evaluation Metrics. We report the following metrics defined in section 4:

- **C@10**, or C-Hits@10 ($\uparrow$): Current Hits@10.

- **DF@10**, or DF-Hits@10 (↓) : Deleted Facts Hits@10.
- **RRD** (↑) : Reciprocal Rank Difference.
- **A@10**, or A-Hits@10 (↑) : Average Hits@10.

The symbol ↑ suggests the higher the metric value, the better the performance. The opposite is true for ↓.

For each evaluation metric, we calculate the average across the incremental learning time steps. For example, the reported current Hits@10 on Wikidata12k is $\frac{1}{22}\sum_{t=55}^{77}C_t$, as shown in figure 3, time step 55 is the beginning of the incremental learning.

We also report the average training time per epoch, and the size of training data at each time step. Note that the training time includes only the GPU computation time and omits any sampling conducted on the CPU to ensure a fair comparison. Dataset size represents the total number of quadruples involved in the training, including both current facts, experience replay facts, and the sampled negative quadruples.

6.2.3 Representation Learning Base Models. We directly adapt **DE** [11] and **HyTE** [7] to the incremental learning setting. The exact formulations are introduced in Section 3.2. We use ComplEx [32] and TransE [2] as the decoding functions for DE and HyTE, respectively.

6.2.4 Baselines and Skylines Algorithms. To demonstrate the effectiveness of our incremental training framework, we compare three incremental training strategies, including Fine-tune (**FT**), Temporal regularization (**TR**), and the complete proposed model **TIE**.

(1) Fine-tune (**FT**): this is a naive baseline that only utilizes added facts D_{add}^t and corresponding cross-entropy loss to fine-tune the model at each time step.
(2) Temporal Regularization (**TR**): TR uses temporal regularization loss (Section 5.3) in addition to FT, and it uses the same data for training.
(3) The proposed complete model (**TIE**): TIE follows all the procedures described in Algorithm 2. It includes training using added facts, deleted facts, temporal regularization and experience replay with positive facts. The choice of experience replay sampling strategy (random, frequency-based or inverse-frequency based) is conditioned on the validation set performances for each experiment setting.

We also provide two skyline training mechanisms — Full Batch (**FB**) and Full Batch training with future facts (**FB_future**) to showcase the approximate accuracy upper-bound of our proposed incremental training algorithm.

(1) Full-batch (**FB**): In this setting, we use all the quadruples in B^t and D^t to fine-tune the model. Negative sampling is based on facts at different time steps. No regularization or distillation is involved.
(2) Full-batch with future data (**FB_future**): This setting acts as an oracle with access to all data in $D^1, ..., D^T$. The training and evaluation protocol is described in Section 3.1.

6.3 Implementation and Hyperparameters

Our models are implemented using PyTorch with the support of the PyTorch-lightning library [9]. We set the learning rate to 10^{-3} and embedding sizes to 128 for all experiments. We use 2048 as the batch size. When training using both current quadruples, experience replay, and deleted facts, we ensure that the total number of

these samples in each batch is at most 2048. For the training data at the current time step, we use 500 as a negative sampling rate, meaning that we sample 500 negative entities for each query. Because we corrupt subjects and objects separately, there are in total 1000 negative samples collected to estimate the probability of a factual triple. For the replay samples, we set the negative sampling rate to 50, i.e., sampling 100 negative entities for each replay fact. We find that this ratio achieves an appropriate trade-off between task performance and training time. The above hyperparameters are selected based on the validation set performances. For experience replay experiments, the training time window length τ is set to 10, and we sample 1,000 historical samples per time step.

For positive reservoir sampling (described in Section 5.2), we set $\sigma = 10$ and $\gamma = 0.5$. We list the λ values associated with each pattern in P (Equation 5.2.1): $\lambda_{(s,r,o)} = 2, \lambda_{(s,*,o)} = 1.5, \lambda_{(s,r,*)} = \lambda_{(*,r,o)} = 1.3, \lambda_{(s,*,*)} = \lambda_{(*,*,o)} = 1, \lambda_{(*,r,*)} = 0$. We set all α's to 1, thus imposing all the loss terms with equal weights.

6.4 Comparative Study

Table 2 reports the performances of the proposed methods and baselines. We make the following observations:

(1) Our proposed method **TIE** achieves better C-Hits@10 and A-Hits@10 than both the FT and TR baselines (besides Wikidata12K + DE combination). It also performs better in terms of intransigence measures in DE experiments.
(2) Although two full-batch skylines achieve better C-Hits@10 and A-Hits@10 than our proposed methods, they perform poorly on intransigence measures. This indicates the full-batch methods fail to identify obsolete facts promptly. Moreover, **FB_future** is impractical in our setting because the model does not have access to future data. The practical skyline **FB** yields a relatively small advantage compared to the proposed methods on C-Hits@10 but performs dramatically worse on the intransigence measures.
Despite the improvement, the RRD results of all methods are negative, meaning that on average, the deleted facts rank higher than the currently valid facts. Further research is required to develop better strategies to reverse this.
(3) Comparing the different base models, HyTe outperforms DE in terms of C-Hits@10 and A-Hits@10 but under-performs for DF@10 and RRD. Moreover, augmenting the HyTE fine-tuning baseline with either TR or TIE does not result in an improvement. Both TR and experience replay emphasize remembering the previously valid facts and hence improve the A-Hits@10 metric significantly. However, due to the limited representation capacity of HyTE's time-embedding approaches, the model is unable further to distinguish the deleted facts from the previously true facts. This is partly because all the entity and relation embeddings are projected using the same time embedding at each time step.
(4) **TIE** significantly improves both the time efficiency and data efficiency compared to full-batch training settings. The proposed **TIE** sacrifices time efficiency compared to the TR method as the loss computation of the experience replay component is costly. Nevertheless, **TIE** and the **TR** methods reduce the training time by about 10 and 100 times, respectively comparing to **FB**.

Dataset	Base model	Algo.	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)	Training time(s)[1]	Data size
Wikidata12k	DE	FT	31.90 ± 0.42	12.50 ± 0.10	-4.16 ± 0.07	36.05 ± 0.52	0.89	424.7K
		TR	34.14 ± 0.17	13.05 ± 0.08	-4.45 ± 0.09	**38.43 ± 0.13**	1.27	424.7K
		TIE	**34.90 ± 0.18**	**8.76 ± 0.67**	**-1.89 ± 0.36**	37.57 ± 0.07	15.79	1.41M
		FB	35.93 ± 0.29	64.42 ± 0.56	-16.79 ± 0.39	38.09 ± 0.35	91.53	49.31M
		FB_future	51.55	64.51	-14.87	54.52	540.19	257.54M
	HyTE	FT	39.01 ± 0.53	33.49 ± 0.44	**-11.62 ± 0.20**	43.00 ± 0.37	0.77	424.7K
		TR	41.67 ± 0.09	**33.13 ± 0.51**	-11.64 ± 0.14	46.15 ± 0.14	1.08	424.7K
		TIE	**42.41 ± 0.24**	35.02 ± 0.80	-12.04 ± 0.21	**47.00 ± 0.01**	9.63	1.41M
		FB	43.34 ± 0.16	68.34 ± 0.55	-18.13 ± 0.38	46.14 ± 0.12	88.59	49.31M
		FB_future	56.88	69.27	-15.38	60.43	922.53	257.54M
YAGO11k	DE	FT	18.79 ± 0.29	7.59 ± 0.25	-0.23 ± 0.11	22.33 ± 0.19	0.84	296.2K
		TR	20.82 ± 0.08	7.39 ± 0.03	-0.12 ± 0.02	23.44 ± 0.02	1.19	296.2K
		TIE	**21.32 ± 0.18**	**7.28 ± 0.20**	**-0.05 ± 0.04**	**23.95 ± 0.23**	11.27	1.28M
		FB	29.73 ± 0.06	57.61 ± 0.43	-12.23 ± 0.03	31.23 ± 0.09	158.05	62.69M
		FB_future	33.26	32.90	-6.37	34.93	300.98	215.89M
	HyTE	FT	30.39 ± 0.10	**36.68 ± 0.50**	**-10.75 ± 0.08**	34.09 ± 0.07	0.59	296.2K
		TR	**31.99 ± 0.24**	38.93 ± 1.31	-11.26 ± 0.50	35.80 ± 0.05	0.89	296.2K
		TIE	**31.99 ± 0.25**	39.93 ± 0.90	-11.63 ± 0.41	**35.91 ± 0.18**	8.82	1.28M
		FB	35.83 ± 0.15	63.83 ± 0.20	-13.36 ± 0.07	37.43 ± 0.07	106.64	62.69M
		FB_future	39.48	61.37	-11.90	41.29	663.36	215.89M

Table 2: The overall performance comparison, averaged over four runs using different random seeds (except FB_future). The mean and standard deviation results on test set are reported.

6.5 Ablation Study

To better understand the effectiveness of the proposed methods, we conduct the following four ablation experiments. We use DE as the base model on the Wikidata12k and YAGO11k datasets for ablation analysis. The test results are averaged over four randomized runs. Standard deviations are sometimes omitted due to space constraints. We use "Wiki" and "YAGO" as abbreviates of the two datasets.

6.5.1 Experience Replay Sampling Methods (5.2.1). As described in Sec 5.2.1, we proposed multiple sampling strategies for sampling positive facts from the replay buffer. Table 3 shows the results of applying each sampling strategy, as well as no experience replay. We first observe that using positive facts leads to a better performance in almost all measures compared to not using experience replay, irrespective of the specific sampling strategy.

Comparing among the different sampling strategies, the frequency-based sampling performs statistically better on C-Hits@10 and the intransigence measure than other alternatives for the Wikidata12k dataset. The inverse frequency-based sampling performs better on the C-Hits@10 and A-Hits@10 on average, while worse on the DF and RRD measures by average. However, the results for the YAGO11k data set are inconclusive. Considering the standard deviation, no method achieves a significant outperformance.

As the best sampling strategy in each combination of the base model and dataset varies, we treat the sampling strategy as a categorical hyperparameter and choose the one that yields the best validation performance for each experiment setting to apply on the test set.

Dataset	Replay	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)
Wiki	None	33.33 ± 0.13	9.60 ± 0.23	-2.57 ± 0.13	36.59 ± 0.05
	Uniform	34.70 ± 0.17	9.36 ± 0.50	-2.12 ± 0.16	**37.65 ± 0.15**
	Freq	**34.90 ± 0.18**	**8.76 ± 0.66**	**-1.89 ± 0.36**	37.57 ± 0.07
	Inv-Freq	34.48 ± 0.12	9.37 ± 0.72	-2.07 ± 0.43	37.62 ± 0.12
YAGO	None	20.65 ± 0.23	**7.18 ± 0.14**	**-0.03 ± 0.03**	23.25 ± 0.18
	Uniform	21.07 ± 0.15	**7.19 ± 0.14**	-0.13 ± 0.03	23.70 ± 0.12
	Freq	21.29 ± 0.15	7.22 ± 0.11	-0.13 ± 0.07	23.85 ± 0.19
	Inv-Freq	**21.32 ± 0.18**	7.28 ± 0.20	**-0.05 ± 0.04**	**23.95 ± 0.20**

Table 3: Ablation analysis on different experience replay strategies. All experiments use added facts, deleted facts and temporal regularization. Uniform, Freq, and Inv-Freq stand for uniform sampling, frequency-based sampling, and inverse frequency based sampling respectively.

6.5.2 Learning with Deleted Facts (5.4). To evaluate the effectiveness of the deleted fact sampling component, we compare the performance of adding the deleted facts (Del) to the following baseline models: 1) temporal regularization (TR); 2) the combination of temporal regularization and experience replay (TR + Replay). All models use added facts by default. As shown in Table 4, adding sampled deleted facts consistently improves the model's intransigence measure in DF-Hits@10 and RRD. Meanwhile, it slightly impedes the model's performance in terms of C-Hits@10 and A-Hits@10 for most cases. This demonstrates that the deleted fact sampling helps the model differentiate triples with low ranks from triples with high ranks with a limited sacrifice in the model's ranking performance. Practitioners should be mindful of this trade-off when deciding whether to use deleted facts.

6.5.3 Learning with Added Facts (5.5). To evaluate the impact of only using newly added facts for incremental training, we conduct experiments to compare 1) only using added facts D_{add}^t and 2) using all the new facts D^t for training at each time step. Table 5

---

[1]Training time refers to the average time needed to train the model for one epoch. To compare the efficiency fairly, we conduct experiments of different models on the same machine (with a single NVIDIA Tesla V100 GPU).

Dataset	TR	Replay	Del	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)
Wiki	✓			**34.14**	13.05	-4.45	**38.43**
	✓		✓	33.33	**9.60**	**-2.57**	36.60
	✓	✓		**3614**	15.07	-0.45	**39.87**
	✓	✓	✓	34.90	**8.76**	**-1.89**	37.57
YAGO	✓			**20.82**	7.39	-0.12	**23.44**
	✓		✓	20.65	**7.18**	**-0.03**	23.25
	✓	✓		**22.09**	7.77	-0.25	**24.29**
	✓	✓	✓	21.32	**7.28**	**-0.05**	23.95

Table 4: Ablation analysis of applying deleted facts sampling. Replay and Del stand for the experience replay and deleted facts training respectively.

shows that training with all facts generally yields slightly better or comparable performance across all measures to training with only added facts. More specifically, training with all facts consistently performs better on C-Hits@10. However, there exists a trade-off between sampling efficiency and other metrics. The sizes of training data required by added-facts experiments are only 9% and 4.62% of those required by the all-facts experiments for Wikidata12k and YAGO11k, respectively.

Dataset	Algo.	C@10 (↑)	RRD (↑)	A@10 (↑)	Data size
Wiki	FT-Add	31.90 ± 0.42	-4.16 ± 0.07	**36.05** ± 0.52	**424.7K**
	FT-All	**33.54** ± 0.39	**-4.13** ± 0.55	35.11 ± 0.36	4.71M
	TR-Add	34.14 ± 0.17	**-4.45** ± 0.09	38.43 ± 0.13	**424.7K**
	TR-All	**34.92** ± 0.20	-4.63 ± 0.12	**38.62** ± 0.08	4.71M
YAGO	FT-Add	18.79 ± 0.29	**-0.23** ± 0.11	22.33 ± 0.34	**296.2K**
	FT-All	**23.45** ± 0.22	-2.73 ± 2.18	**25.07** ± 0.27	6.40M
	TR-Add	20.82 ± 0.08	-0.12 ± 0.02	23.44 ± 0.03	**296.2K**
	TR-All	**21.50** ± 0.05	**-0.00** ± 0.02	**23.91** ± 0.10	6.40M

Table 5: Ablation analysis of only using added facts for incremental learning. "Add" in the Algo. column means the model uses only added facts at each incremental training step. In contrast, "All" means the model uses all facts.

6.5.4 Experience Replay Sampling Size(5.2). To understand how the size of replay data affects the model's performance, we conduct experiments altering the size of the sampled replay facts on Wiki dataset. For Table 6, we fix the window length $\tau = 10$ and vary the replay sampling size for each step. For Table 7, we vary τ and the sample size jointly.

Table 6 shows all measures tend to improve as the sampling size increases while keeping τ fixed. On the contrary, Table 7 shows that larger replay data size does not always lead to better performance as the model's performance is sensitive to the value of τ, which is the window length of the old data. Among experiments from Table 7, best performance is achieved when τ and replay size are 15 and 15,000, respectively. Meanwhile, from both tables we notice that training time increases as the replay sample size grows. Performance and training time trade-off needs to be considered while choosing hyper-parameters.

sample size	τ	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)	Training time
1000	10	36.17	11.60	-3.04	38.84	6.07
3000	10	36.02	10.07	-2.09	38.64	8.02
10000	10	34.90	**8.76**	-1.89	37.57	15.79
20000	10	36.71	9.08	-1.72	39.32	20.30
40000	10	**37.21**	9.34	**-1.59**	**39.53**	31.83

Table 6: Ablation analysis of using different experience replay sampling size.

sample size	τ	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)	Training time
5000	5	36.87	11.09	-2.45	39.43	8.79
10000	10	34.90	**8.76**	**-1.89**	37.57	15.79
15000	15	**36.94**	9.65	-2.24	**39.53**	19.61
20000	20	36.36	9.80	-1.94	35.02	25.95
25000	25	36.41	10.43	-2.22	35.12	28.92

Table 7: Ablation analysis on different experience replay data size.

7 CONCLUSION

We present a novel incremental learning framework named TIE for TKGC tasks. TIE combines TKG representation learning, frequency-based experience replay, and temporal regularization to improve the model's performance on both current and past time steps. TIE leverages pattern frequencies to select among reservoir samples and uses only the deleted and added facts at the current time step for training, which significantly reduces training time and the size of training data. Moreover, we propose DF and RRD metrics to measure the intransigence of the model. Extensive ablation studies shows each proposed component's effectiveness. They also provide insights for deciding among model variations by revealing performance trade-offs among various evaluation metrics.

This work serves as a first attempt and exploration to apply incremental learning to TKGC tasks. Future work might involve exploring other incremental learning techniques, such as constrained optimization, to achieve more robust performance across datasets and metrics.

REFERENCES

[1] Kian Ahrabian, Daniel Tarlow, Hehuimin Cheng, and Jin LC Guo. 2020. Software Engineering Event Modeling using Relative Time in Temporal Knowledge Graphs. *arXiv preprint arXiv:2007.01231* (2020).

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. 2787–2795.

[3] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*. 233–248.

[4] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. 2018. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 532–547.

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2019. Efficient Lifelong Learning with A-GEM. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hkf2_sC5FX

[6] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. 2019. Continual learning with tiny episodic memories. (2019).

[7] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. 2018. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2001–2011.

[8] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. 2014. Introducing Wikidata to the linked data web. In *International semantic web conference*. Springer, 50–65.

[9] WEA Falcon et al. 2019. Pytorch lightning. *GitHub. Note: https://github.com/williamFalcon/pytorch-lightning Cited by* 3 (2019).

[10] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international*

conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 249–256.

[11] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3988–3995.

[12] Zhen Han, Yuyi Wang, Yunpu Ma, Stephan Guünnemann, and Volker Tresp. 2020. The Graph Hawkes Network for Reasoning on Temporal Knowledge Graphs. *arXiv preprint arXiv:2003.13432* (2020).

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[14] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 105–113.

[15] David Isele and Akansel Cosgun. 2018. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[16] Tingsong Jiang, Tianyu Liu, Tao Ge, Lei Sha, Baobao Chang, Sujian Li, and Zhifang Sui. 2016. Towards time-aware knowledge graph completion. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 1715–1724.

[17] Tingsong Jiang, Tianyu Liu, Tao Ge, Lei Sha, Baobao Chang, Sujian Li, and Zhifang Sui. 2016. Towards Time-Aware Knowledge Graph Completion. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, 1715–1724.

[18] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. 2020. Recurrent Event Network: Autoregressive Structure Inference over Temporal Knowledge Graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6669–6683.

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* (2017).

[20] Julien Leblay and Melisachew Wudage Chekol. 2018. Deriving validity time in knowledge graph. In *Companion Proceedings of the The Web Conference 2018*. International World Wide Web Conferences Steering Committee, 1771–1776.

[21] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*. 6467–6476.

[22] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. 2017. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of the 26th international conference on World Wide Web*. 1211–1220.

[23] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. 2013. Yago3: A knowledge base from multilingual wikipedias.

[24] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[25] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web* 8, 3 (2017), 489–508.

[26] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. 2020. GDumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*. Springer, 524–540.

[27] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*.

[28] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 519–527.

[29] Hyun-Je Song and Seong-Bae Park. 2018. Enriching Translation-Based Knowledge Graph Embeddings Through Continual Learning. *IEEE Access* 6 (2018), 60489–60497.

[30] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3462–3471.

[31] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyePrhR5KX

[32] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*. 2071–2080.

[33] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2020. Composition-based Multi-Relational Graph Convolutional Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BylA_C4tPr

[34] Jiapeng Wu, Meng Cao, Jackie Chi Kit Cheung, and William L Hamilton. 2020. TeMP: Temporal Message Passing for Temporal Knowledge Graph Completion. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 5730–5746.

[35] Chengjin Xu, Mojtaba Nayyeri, Fouad Alkhoury, Jens Lehmann, and Hamed Shariat Yazdi. 2019. Temporal Knowledge Graph Embedding Model based on Additive Time Series Decomposition. *arXiv preprint arXiv:1911.07893* (2019).

[36] Yishi Xu, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, and Mark Coates. 2020. GraphSAIL: Graph Structure Aware Incremental Learning for Recommender Systems. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2861–2868.

[37] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).

[38] Yang Yang, Da-Wei Zhou, De-Chuan Zhan, Hui Xiong, and Yuan Jiang. 2019. Adaptive Deep Models for Incremental Learning: Considering Capacity Scalability and Sustainability. In *Proc. ACM Conf. Knowledge Discovery and Data Mining*.

[39] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual Learning Through Synaptic Intelligence. *Proceedings of machine learning research* 70 (2017), 3987–3995.

[40] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[41] Fan Zhou, Chengtai Cao, Ting Zhong, Kunpeng Zhang, Goce Trajcevski, and Ji Geng. 2020. Continual Graph Learning. *arXiv:2003.09908* (2020).

A APPENDIX

A.1 A-GEM Formulation and Adaptation

In this section, we introduce the A-GEM formulation and the its adaptation to our task in detail. Figure 4 presents the architecture of A-GEM variation in details.

Formulation. The goal of A-GEM [5] is to optimize the loss on the current samples without increasing the loss on the previous samples, which is approximated by the *average* loss. At every time step, A-GEM insures that the average loss on the reservoir samples does not increase. We slightly modify the original formulation to adapt to the TKGC task. At time step t, the objective of A-GEM is:

$$\text{minimize} \quad L(\phi^t, D^t) \quad \text{s.t.} \quad L(\phi^t, P^t) \leq L(\phi^{t-1}, P^t). \quad (20)$$

Replacing the loss terms using our proposed loss terms in the previous section, the above objective is written as:

$$\text{minimize} \quad L_{CE} + L_{del} + L_{TR} \quad \text{s.t.} \quad L_{RCE}^t \leq L_{RCE}^{t-1}, \quad (21)$$

where L_{RCE}^t is the same loss term as in equation 12, and L_{RCE}^{t-1} is written as:

$$L_{RCE}^{t-1} = -\sum_{s,r,o,t' \in P^t} log(q_{s,r,o,t'}^{t-1}). \quad (22)$$

Let g denote the gradient update on the current loss, i.e. $L_{CE} + L_{del} + L_{TR}$, and g_{ref} denote the gradient of L_{RCE}^t. The above optimization problem is shown to be equivalent to:

$$\text{minimize}_{\tilde{g}} \frac{1}{2}||g - \tilde{g}||^2 \quad \text{s.t.} \quad \tilde{g}^T g_{ref} \geq 0. \quad (23)$$

When violating the constraints, i.e. $\tilde{g}^T g_{ref} < 0$, the solution is derived as:

$$\tilde{g} = g - \frac{g^T g_{ref}}{g_{ref}^T g_{ref}} g_{ref}. \quad (24)$$

Modified A-GEM parameter update. A-GEM was developed and evaluated on image classification tasks, where parameters updates are performed on the neural network parameters. When optimizing the embedding based models, however, problems arise. Suppose the entity *united states* is involved in some facts in P^t but can't be found in D^t. In the standard optimization setting, the gradient of the embedding of *united states* will be applied. However, in A-GEM, if the constraint $g^T g_{ref}^t \geq 0$ is violated, which is computed using gradients of all parameters in the model, then the embedding of all entities likes *united states* will not be updated. This will incur huge cost of gradient computation and less optimal parameter optimization.

To address this issue, we shift the target of the constraint from global gradient matrix to each vector representation. Let g_i and

g_{ref_i} denote the i-th row of the gradient matrices g and g_{ref} respectively. The optimization goal is modified to:

$$\text{minimize}_{\tilde{g}_i} \frac{1}{2}||g_i - \tilde{g}_i||^2 \quad \text{s.t.} \quad \tilde{g}_i^T g_{ref_i} \geq 0, \forall i. \quad (25)$$

The projected gradient matrix $\tilde{g}$ is:

$$\tilde{g}_i = \begin{cases} g_i, & \text{if } \tilde{g}_i^T g_{ref_i} \geq 0, \\ g_i - \frac{g_i^T g_{ref_i}}{g_{ref_i}^T g_{ref_i}} g_{ref_i}, & \text{otherwise.} \end{cases} \quad (26)$$

A.2 Supplementary Ablation Study

A.2.1 Experience replay buffer size(5.2). To examine the effectiveness of the experience replay component, we conduct experiments by altering the maximum number of the positive replay facts sampled from the replay buffer. We alter the replay sample sizes in two separate experiments. First, in Table 6, we fix the window length $\tau = 10$ and vary the replay sampling size. Second, in Table 7, we vary both τ and the sample size, where the sample size grows linearly with the τ.

As observed from Table 6, all measures tend to improve as the sampling size increases. The largest sampling size of 40,000 achieves the best result for all measures except DF-Hits@10. At the same time, we notice that each time step's training time also increases as the replay sample size grows. There exists a trade-off between performance and training time.

Interestingly, unlike the result from table 6, we observe in table 7 that larger data size does not lead to a better performance in terms of neither Hits@10 measure nor intransigence measure. The best C-Hits@10, A-Hits@10, and DF-Hits@10 are reached when sampling 15,000 positive replay facts with the time window size of 15. Further increasing the sampling size beyond 20,000 and time window size beyond 20 leads to a decrease in all measures. Combining with the previous observation, we conclude that a larger sampling size generally leads to better performance. Meanwhile, replaying the historical facts within a certain time range benefits the model, and the effectiveness of the experience replay component is sensitive to the time window size.

A.3 A-GEM optimization

We report the comparative study between the original TIE model and the A-GEM variation of TIE model in Table 8. The difference between the model performance with and without A-GEM are not statistically significant. This echos the finding in [6] that simple experience replay methods are empirically better than A-GEM on incremental image classification tasks. Our result can be seen an extended piece of evidence in the domain of incremental knowledge graph completion.
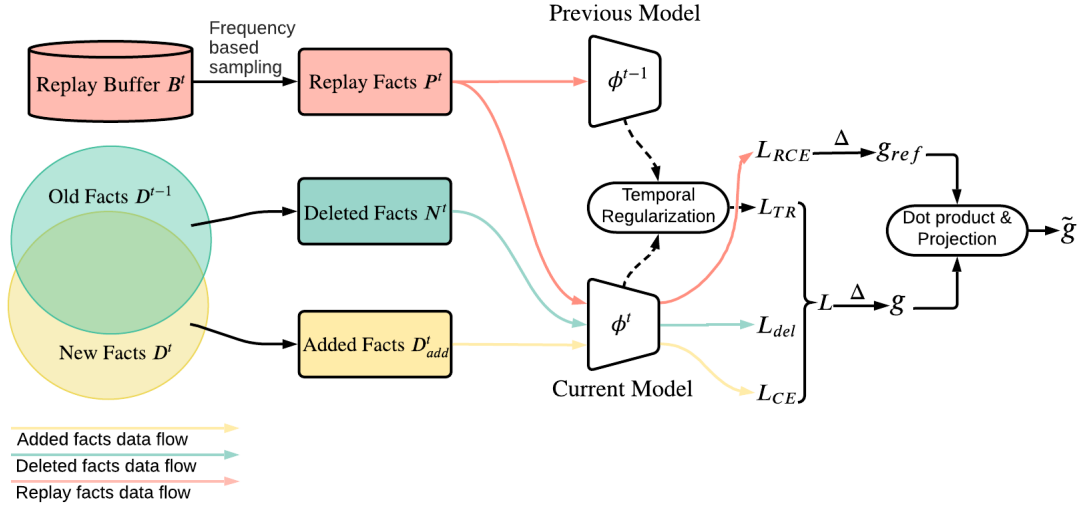
Figure 4: Architecture of the A-GEM variation of TIE model.

Dataset	Replay	A-GEM	C@10 (↑)	DF@10 (↓)	RRD (↑)	A@10 (↑)
Wiki	Uniform		34.70 ± 0.17	9.36 ± 0.50	**-2.12** $\pm$ 0.16	**37.65** $\pm$ 0.15
	Uniform	✓	**34.84** $\pm$ 0.16	**9.23** $\pm$ 0.30	-2.22 ± 0.23	37.66 ± 0.15
	Freq		**34.90** $\pm$ **0.18**	**8.76** $\pm$ 0.66	**-1.89** $\pm$ 0.36	**37.57** $\pm$ 0.07
	Freq	✓	34.76 ± 0.20	8.87 ± 0.59	-1.98 ± 0.25	37.45 ± 0.11
	Inv-Freq		34.48 ± 0.12	**9.37** $\pm$ 0.72	**-2.07** $\pm$ 0.43	**37.62** $\pm$ 0.12
	Inv-Freq	✓	34.47 ± 0.35	9.61 ± 0.26	-2.29 ± 0.20	37.47 ± 0.18
YAGO	Uniform		21.07 ± 0.15	**7.19** $\pm$ 0.14	**-0.13** $\pm$ 0.03	23.70 ± 0.14
	Uniform	✓	**21.55** $\pm$ 0.20	7.35 ± 0.15	-0.26 ± 0.10	**23.97** $\pm$ 0.12
	Freq		**21.29** $\pm$ 0.15	7.22 ± 0.11	**-0.13** $\pm$ 0.07	**23.85** $\pm$ 0.16
	Freq	✓	21.21 ± 0.24	**7.16** $\pm$ 0.12	-0.15 ± 0.06	23.76 ± 0.19
	Inv-Freq		21.32 ± 0.18	**7.28** $\pm$ 0.20	**-0.05** $\pm$ 0.04	**23.95** $\pm$ 0.23
	Inv-Freq	✓	21.31 ± 0.36	7.35 ± 0.18	-0.17 ± 0.12	23.87 ± 0.20

Table 8: Ablation analysis of A-GEM under using sampling methods. Replay stands for the type of replay sampling method and A-GEM column indicates whether A-GEM is used or not. All experiments use added facts, deleted facts and temporal regularization bu default.