RESPONSE TO REVIEWS (PAPER 842)

Dear Meta Reviewer and Referees:

We have substantially revised the paper following the Referees' suggestions. For the convenience of Referees, changes to the paper are color-coded in blue.

We would like to thank the Referees for insightful comments and for supporting this work!

Below please find our responses to the reviews.

Response to the revision items raised by the Meta Reviewer.

However, there is still a number of issues that the authors need to clarify in order to bring the paper into a publishable form.
(1) The related work needs to become more comprehensive and the relationship with these works better understood. That may involve in certain cases comparisons and textual explanations (especially the RankBert and DittoRank)
(2) The experiments need to provide more explanations on how they were conducted, the reasoning behind them and more insights.
(3) add experiments with different initial timestamp ratios.
(4) investigate patterns of missing timestamps in practice and see how the proposed solution behaves
(5) explain better what Ditto_Rank does
(6) include Precision and Recall as in [18] and [42] and explain the interplay between them.
(7) Use measures for ranking such as MRR, MAP, etc.
(8) fix format according to VLDB template and also typos
(9) Provide convincing evidences about the reliable orders
(10) provide strong evidences of real life examples where timestamps are missing and in what way (pattern) they are missing
(11) overall, make the motivation stronger. This is an important issue the authors need to elaborate to convince the readers.

[A] Thanks! We have revised the paper substantially and addressed the comments from all referees. Specifically, we have addressed (1) in the response to R3[W1&D1]; (2) in the response to R1[W1, W3], R2 and R3[W2&D2] ; (3) in the response to R3[W2&D2]; (4) in the response to R1[W3] and R3[W2&D2]; (5) in the response to R2[D1]; (6) in the response to R2[D2]; (7) in the response to R2[W1&D3]; (8) in the response to R1[W4]; (9) in the response to R1[W1, W2]; (10) in the response to R1[W1, W3] and (11) in the response to R1[W1].

In addition, we have simplified some discussion, replaced accuracy metric Accu with metrics MRR and MAP@K as suggested by R2, and moved the notation table, the proofs, some experimental results and details of model training (*e.g.,* fine-tuning based on conflicts in Section 4) to the online full version [1], to make room for the newly added motivating examples, statistics and experiments.

Response to the comments of Referee #1.

[W1] *My main concern is the motivation and practical significance. There is an example in the introduction, but there is no discussion about why timestamps might be missing. It would be nice to see real-life examples where these problems come up, which would nicely motivate the proposed solution. Some statistics would be nice. What is the fraction of missing timestamps? Are they missing at random? Why is it necessary to discover order between attributes and not whole tuples (with specific examples)? Is it possible to deduce additional temporal orders based on existing timestamps? How many orders can't be deduced in the end? Without such numbers, it's impossible to tell if the proposed solution is necessary in practice.*

[A] Thanks! We have substantially strengthened the motivation and practical significance, by providing real-life examples and statistics in real-life datasets. We have made the following changes (pp.1-2):

(1) We have further clarified that in this paper, we say that a timestamp is *reliable* if it is precise, correct and moreover, at the time the associated attribute values are *correct* and *up-to-date* (pp. 1).

(2) We have discussed why only partial reliable timestamps are available, including both mechanical reasons (*e.g.,* malicious attacks or hardware failures), which can be avoided by solid backup, and the following logical reasons, which are harder to avoid (pp. 1).

(a) *(Missing timestamps)* Timestamps may simply not be recorded, *e.g.,* in an e-health database [42], only 16 out of 26 relations are timestamped. Even when a relation has timestamps, it may not be complete, *e.g.,* 25.36% missing in [55] and up to 82.28% missing in [42]. As another example, while deep learning [38] has been used to predict missing values, the predicted values may not be timestamped, since its exact correctness is not guaranteed.

(b) *(Imprecise timestamps)* Timestamps may be too coarse, leading to unreliable ordering. For example, an e-form may be submitted multiple times to an OA system by employees during a day. If the forms are recorded in datestamps, it is not clear which form (all on the same day) is the latest. Similar problems are often encountered in hospital data, where only dates are recorded [5]. Another example concerns inconsistent granularity of timestamps (*e.g.,* minutes vs. days [5, 42]). If two attribute values have timestamps "12-8-2021" and "12-8-2021 20:41", respectively, it is not clear which one is more up-to-date. As reported in [55], 90.41% of appointment records have imprecise timestamps.

(c) *(Incorrect timestamps)* Many factors can lead to incorrect timestamps. For example, in medical data [5], an X-ray machine has many asynchronous modules, each of which has a local clock and a local buffer. There is possibly a discrepancy between when a value is actually updated and when it is recorded, since the value is first queued in the buffer before it is recorded. Moreover, as shown in [55], 36.96% of appointments have the overlapping issue (*i.e.,* the next appointment in a particular room appears to have started before the current one has ended), which obviously indicates the incorrectness of some timestamps.

These reasons were observed as the major timestamp-related issues in data quality [5], and account for the absence of reliable timestamps. No matter how desirable, the percentage of reliable timestamps in real-life data is far lower than expected. Hence in our experiments (pp. 9), the default ratio of initial timestamps is intentionally set small (5%, so that the problem is more challenging), to justify the effectiveness of our method. We have also added new experiments by varying the ratio from 4% to 20% (pp. 10-11); GATE works better with higher initial timestamp ratios, as expected.

(3) We have investigated the pattern of missing timestamps and motivated the need for an attribute-level ordering scheme (rather than a tuple-level ordering scheme) with specific examples (pp. 1).

For example, customer data changes frequently as reported by Royal Mail [62]. To prevent the data from being outdated, the sales department may contact the customers and get regular updates on some *critical* information (*e.g.,* emails and phone numbers). Due to the impatience of customers and high cost (*e.g.,* man power) of contact, the rest (*e.g.,* jobs and marital status) may not be frequently updated and may gradually become obsolete, no longer having a reliable timestamp. Add to this the complication that data values are often copied or imported from other sources [22, 23], and even in the same database, values can come from multiple relations (*e.g.,* by joining), where no uniform scheme of timestamps is granted, *e.g.,* some values are more frequently timestamped than the others [42].

In light of this, we need an attribute-level scheme that associates each value with a timestamp. This subsumes the tuple-level scheme (*i.e.,* all values in a tuple have the same timestamp) as a special case.

(4) While a ranking model can learn a total order for *all* values, there is no guarantee to ensure that the learned orders are reliable. When it comes to rule-based methods, it is hard to find enough rules to deduce relative orders on *each and every* pair of attribute values. We tested the existing rule-based methods. We find that even the best of such methods can only deduce 16.3% temporal orders on Career dataset, given limited initial timestamps (5%), leaving the remaining 78.7% undecided (pp. 2). It is to overcome the limitations of the previous methods that we combine deep learning and rule-based methods (please see the response to **[W2]** below).

[W2] *At the end of section 3, there is a remark about a confidence threshold to filter out unreliable orders. I think this is an important point that deserves more attention. In practice (real datasets, real missing timestamps), how many reliable orders can be found? How many pairs cannot be ordered due to lack of evidence? Is it possible to have conflicting evidence for some pairs of tuples, and if so, how is this handled?*

[A] As suggested, we have tested the impact of confidence threshold δ on our ranking model $\mathcal{M}_{\mathsf{rank}}$ on a real-life procedural billing dataset of patients during their ICU stays (pp. 5), with 82.28% *real* missing timestamps [42] (note that this dataset cannot be used in Section 6, due to the lack of ground truth). We find that 22.6% ranked pairs are identified as confident orders when $\delta = 0.55$. This number decreases to 16.7% when δ increases to 0.6. With a reasonable δ, a large percentage of reliable temporal orders can be learned by the ranking model and be justified by currency constraints. More tests on δ are reported in Section 6 (Figure 6(n) & pp. 11-12).

We have also remarked that $\mathcal{M}_{\mathsf{rank}}$ do not predict both $t_1 \preceq_A t_2$ and $t_2 \preceq_A t_1$ as confident orders (pp. 7) *i.e.,* no conflicts by $\mathcal{M}_{\mathsf{rank}}$.

[W3] *Experimental datasets are real but the missing timestamps are not (5% randomly selected, others masked). Going back to O1, is this realistic? It would be better to investigate patterns of missing timestamps in practice and see how the proposed solution behaves for different kinds of patterns. In what situations does it work well? In what situations does it struggle?*

[A] As suggested, we have investigated patterns of missing timestamps (real examples, pp. 1) and added experiments to evaluate the performance for different kinds of patterns (Figures 6(k)-6(l), pp. 11).

As shown on pp. 1, missing timestamps (up to 82.28% [42]), imprecise timestamps (90.41% [55]), incorrect timestamps (36.96% [55]) account for the absence of reliable timestamps. The percentage of reliable timestamps in real-life data is far lower than expected. In the experiments (pp. 9), the default ratio of initial timestamps is set intentionally small (5%, so that the problem is more challenging), to justify the effectiveness of our proposed techniques. In addition, we have varied the initial timestamp ratio from 4% to 20% and conducted new experiments to evaluate its impact (pp. 10-11).

Moreover, as shown by the customer contact example (pp. 1), some attribute values (*e.g.,* the frequently updated ones) may carry more reliable timestamps than the others [42]. In light of this, we have conducted new experiments to study the effect of different distributions of initial timestamps, by *random sampling* (Figure 6(k)) and *weighted sampling* (Figure 6(l)); in the latter setting, we assigned a weight to each attribute, where a larger weight indicates that the values of this attribute is more likely to be selected with initial timestamps, *e.g.,* on dataset Person, values on attribute Status are more likely to carry reliable timestamps than Kids.

When the sampling ratio ts% is increased, the accuracy of GATE is improved, *e.g.,* the F-measure of GATE increases from 0.75 to 0.796 with random sampling (resp. 0.72 to 0.77 with weighted sampling) on Person in Figure 6(k) (resp. Figure 6(l)) when ts% changes from 4% to 20%. We find that the F-measure under weighted sampling (Figure 6(l)) is slightly lower than random sampling (Figure 6(k)), *e.g.,* when ts%=20% on Person, the F-measure under weighted sampling is 2.5% lower than random sampling. This is because weighted sampling retrieves more temporal orders from attributes of larger weights as training instances; as a consequence, the distribution of the training data may differ from the distribution of testing data. This out-of-distribution issue is still a common problem for ML models, and it takes a toll on the creator. In contrast, random sampling does not have such a problem, since the testing data is more likely to be drawn *i.i.d.* (independently and identically) from the same distribution as the training data.

[W4] *Editorial comments*
- The formatting of this paper looks a bit different than other VLDB submissions I've reviewed. Things seem to be more squished.
- page 1 and page 4: "marital status changes from single to married, not the other way around" -> This doesn't sound right. Married changes to single after a divorce.
- pg4: consider using the pronoun "they" instead of he/she to be more inclusive.

[A] Thanks! We have adopted format according to VLDB template and used "this person" instead of "he/she" to be inclusive (pp. 4).

According to Statistic Canada (the official agency of Canada Government who provides statistics service), persons are single if they have *never* married [9]. If a person divorces and has not remarried, the marital status is "Divorced", rather than "Single".

We thank the Referee for constructive suggestions!

Response to the comments of Referee #2.

[W1&D3] *Why to assess the quality of temporal orders with f-measure only? (see detailed evaluation). F-measure is mostly used for measuring accuracy. Since here the goal is to measure how well a*

method ranks I would expect to see measures for raking such as MRR, MAP, etc. Why is that not the case here? The choice of F-score should be motivated beyond citing the other two works that employ it.

[A] Thanks! We have motivated the choice of F-measure (pp. 10). We adopt the pairwise ranking setting to learn and deduce temporal orders (Section 4, pp. 6), due to its consistent semantics with temporal orders and in order to deduce total orders by transitivity. Thus, we adopt the widely-used F-measure to measure the accuracy of the predictions on the ranked pairs.

As suggested, we have also added ranking metrics MRR and MAP@K (pp. 10), to evaluate whether the latest values are ranked properly. We reported the results on Career in Figures 6(d) and 6(e). As shown there (pp. 10), both MRR and MAP@K increase with more rounds in most cases, *e.g.,* MRR and MAP@K increases from 0.786 to 0.857 and from 0.752 to 0.809 after 11 rounds, respectively, and the MRR and MAP@K of GATE outperforms all baselines. This indicates that the latest values are indeed ranked high in the total orders.

Moreover, we have added experiments to further evaluate the ranking of GATE, by varying parameter K of MAP@K from 1 to 5 (pp. 10 & Figure 6(f)). As shown there, GATE consistently achieves the highest MAP@K, *e.g.,* 5% higher than the best baseline on average, up to 10.1%. This verifies that GATE works well on ranking.

Since the performance of GATE is similar under all metrics (*e.g.,* Figures 6(a)-6(e) on Career), we focus on F-measure in most tests for the lack of space; the results of the other metrics are consistent.

[D1] *It is not clear what Ditto_Rank does: From the text: "... then sorts them by the outputted confidences" I might be missing something, but why sorting by the confidence of a binary classifier should approximate the temporal ordering?*

[A] As suggested, we have explained what $\text{Ditto}_{\text{Rank}}$ does (pp. 3 & 10). Note that in NLP, ranking can be handled by a ranking function with a binary classifier as the comparison operator.

Thus, we have implemented $\text{Ditto}_{\text{Rank}}$, which first trains a ditto model [49] to conduct binary classification on attribute values (with contextual information) and then sorts all attribute values using ditto as the comparison operator, *i.e.,* we used bubble-sort to sort the values, where the comparison between two values is done by ditto.

[D2] *[18] and [42] employ Precision and Recall, not just the F-measure. It would help to understand also how Precision and Recall are. Oftentimes in Entity Resolution scenarios, Recall is more important than Precision–but in other scenarios, it could be the other way around.*

[A] Thanks! We have added precision and recall as the accuracy metrics as suggested (pp. 10) and reported the precision and recall of all methods on Career in Figures 6(b) and 6(c), respectively.

As shown there, the precision and recall are 0.859 and 0.873 on Career after 11 rounds, indicating that GATE achieves a good balance between the two and is able to learn/deduce true temporal orders.

Many thanks for your support and constructive suggestions!

Response to the comments of Referee #3.

[W1&D1] *The relevant work should be more comprehensive. The relevant work in the paper should include RankBert and DittoRank, which are baselines in the experiments.*

[A] Thanks! As suggested, we have discussed $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$ in the related work (pp. 3). In NLP, ranking (*e.g.,* in question-answering tasks) can be handled by pre-trained language models with cross-entropy loss or by a ranking function with a binary classifier as the comparison operator; we adopted a baseline from each kind (*i.e.,* $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$) in Section 6.

[W2&D2] *More details are needed for the experiment setting. The authors do not specify the reason for choosing the 5% timestamp as the initial partial timestamp. It is better to add experiments with different initial timestamp ratios.*

[A] Thanks! We have added discussion about why only partial reliable timestamps are available, with real-life examples and statistic (pp. 1). As remarked there, missing timestamps (up to 82.28% [42]), imprecise timestamps (90.41% [55]), incorrect timestamps (36.96% [55]) account for the absence of reliable timestamps. The percentage of reliable timestamps in real-life data is far lower than expected. In our experiments (pp. 9), the default ratio of initial timestamps is set intentionally small (5%, so that the problem is more challenging), to justify the effectiveness of our method.

As suggested, we have also added experiments by varying initial timestamp ratio from 4% to 20% (in our original submission, the ratio ts% was varied from 1% to 5%, pp. 10-11). In addition, we have conducted new experiments to test the effect of different patterns of missing timestamps in Figures 6(k) and 6(l) by random sampling (provided in our original submission) and weighted sampling (new in the revised submission), respectively. Initial timestamps help both the creator and the critic in both settings. As shown in Figure 6(k), the F-measure of GATE increases (from 0.75 to 0.796 on Person) when ts% changes from 4% to 20%, as expected.

Many thanks for your support and helpful suggestions!

Learning and Deducing Temporal Orders

Wenfei Fan[1,2,3], Resul Tugay[2], Yaoshu Wang[1], Min Xie[1], Muhammad Asif Ali[1]
Shenzhen Institute of Computing Sciences, China[1], University of Edinburgh, UK[2] Beihang University, China[3]
wenfei@inf.ed.ac.uk,resul.tugay@ed.ac.uk,yaoshuw@sics.ac.cn,xiemin@sics.ac.cn,asif.ali@sics.ac.cn

ABSTRACT

This paper studies how to determine temporal orders on attribute values in a set of tuples that pertain to the same entity, in the absence of complete timestamps. We propose a creator-critic framework to learn and deduce temporal orders by combining deep learning and rule-based deduction, referred to as GATE (Get the lATEst). The creator of GATE trains a ranking model via deep learning, to learn temporal orders and rank attribute values based on correlations among the attributes. The critic then validates the temporal orders learned and deduces more ranked pairs by chasing the data with currency constraints; it also provides augmented training data as feedback for the creator to improve the ranking in the next round. The process proceeds until the temporal order obtained becomes stable. Using real-life and synthetic datasets, we show that GATE is able to determine temporal orders with F-measure above 80%, improving deep learning by 7.8% and rule-based methods by 43.8%.

1 INTRODUCTION

Real-life data keeps changing. As reported by Royal Mail, on average, "9590 households move, 1496 people marry, 810 people divorce, 2011 people retire and 1500 people die" each day in the UK [62]. It is estimated that inaccurate customer data costs organizations 6% of their annual revenues [62]. Outdated data incurs damage not only to Royal Mail. When the data at a search engine is out of date, a restaurant search may return a business that had closed three years ago. When the data about the condition of infrastructure assets is obsolete, it may delay the maintenance of equipment and cause outage. Moreover, data-driven decisions based on outdated data can be worse than making decisions with no data [51]. Indeed, "as a healthcare, retail, or financial services business you cannot afford to make decisions based on yesterday's data" [26]. Unfortunately, "82% of companies are making decisions based on stale information" [8].

These highlight the need for determining the currency of data, *i.e.,* how up-to-date the information is. This is, however, highly nontrivial. Consider a set of tuples that pertain to the same entity. Their attribute values may become obsolete and inaccurate over the time. Worse yet, only *partial* reliable timestamps might be available, where a timestamp is *reliable* if it is *precise*, *correct* and moreover, it indicates that at the time, the values are *correct* and *up-to-date*.

Apart from mechanical reasons (malicious attacks or hardware failures), the logical reasons below are the major temporal issues in data quality [5], and account for the absence of reliable timestamps.

(1) Missing timestamps. Timestamps may simply not be recorded, *e.g.,* in an e-health database [42], only 16 out of 26 relations are timestamped. Even when a relation has timestamps, it may not be complete, *e.g.,* 25.36% missing in [55], up to 82.28% in [42].

(2) Imprecise timestamps. Timestamps may be too coarse, leading to unreliable ordering. An e-form may be submitted multiple times to an OA system by employees during a day. If the forms are recorded in datestamps, it is not clear which form (all on the same day) is the latest. Similar problems are often encountered in hospital data, where only dates are recorded [5]. Another example concerns inconsistent granularity of timestamps (*e.g.,* minutes vs. days [5, 42]). If two values have timestamps "12-8-2021" and "12-8-2021 20:41", it is not clear which one is more up-to-date. As reported in [55], 90.41% of appointment records have imprecise timestamps.

(3) Incorrect timestamps. Many factors can lead to incorrect timestamps. Taking medical data [5] as an example, an X-ray machine has many asynchronous modules, each of which has a local clock and a local buffer. There can be a discrepancy between when a value is actually updated and when it is recorded since the value is first queued in the buffer before it is recorded. Moreover, 36.96% of appointments have the overlapping issue (*i.e.,* the next appointment in a particular room appears to have started before the current one has ended) [55], indicating incorrect timestamps.

As remarked earlier by Royal Mail [62], customer data changes frequently. To prevent the data from being outdated, the sales department may contact the customers and get regular updates on some *critical* information (*e.g.,* email and phone). Due to the impatience of customers and high cost (*e.g.,* man power) of contact, the rest (*e.g.,* jobs, marital status) may not be frequently updated and may gradually become obsolete, no longer having a reliable timestamp. Add to this the complication that data values are often copied or imported from other sources [22, 23], and even in the same database, values may come from multiple relations (*e.g.,* by join), where no uniform scheme of timestamps is granted, *e.g.,* some values are more frequently timestamped than the others [42]. This calls for an attribute-level time-stamping scheme (*i.e.,* each value has a timestamp). It subsumes the tuple-level scheme (*i.e.,* all values in the same tuple have the same timestamp) as a special case.

No matter how desirable, the percentage of reliable timestamps is far lower than expected. How can we determine the temporal orders on the attribute values, *i.e.,* for tuples t_1 and t_2, whether the value in the A-attribute of t_1 is more current than the value in the A-attribute of t_2, denoted by $t_2 \prec_A t_1$, in the absence of complete timestamps?

Example 1: Consider customer records t_1-t_6 shown in Figure 1, which have been identified to refer to the same person Mary. Each tuple t_i has attributes FN (first name), LN (last name), sex, address, marital status, job, kids and SZ (shoe size). Some attribute values of these tuples have become stale since Mary's data changes over the years, *e.g.,* her job, address and last name have changed 4 times, five times and twice, respectively. Only some attribute values might be associated with reliable timestamps, *e.g.,* the timestamp of t_5[job] and t_6[job] are 2016 and 2019 (not shown), respectively, indicating that at that time, the values are up-to-date. In the absence of complete timestamps, it is hard to know whether $t_2 \prec_{\mathsf{LN}} t_6$, *i.e.,* whether the value of t_2[LN] is more up-to-date than t_6[LN]? Moreover, what Mary's current job title is, *e.g.,* whether $t_i \prec_{\mathsf{job}} t_6$ for $i \in [1, 4]$? □

	FN	LN	sex	address	status	job	kids	SZ
t_1, e_1:	Mary	Goldsmith	F	19 xin st	single	n/a	-	5
t_2, e_1:	Mary	Taylor	F	6 gold plaza	married	journalist	1	6
t_3, e_1:	Mary	Taylor	F	19 mall st	espoused	assoc editor	2	7
t_4, e_1:	Mary	Taylor	F	7 ave	divorced	chief editor	2	7
t_5, e_1:	Mary	Taylor	F	7 ave	detached	chief editor	2	7
t_6, e_1:	Mary	Goldsmith	F	6 const. ave	wed	producer	3	7

Figure 1: Customer records dataset

One may want to approach this problem by training a ranking model that sorts objects according to their degrees of relevance or importance [53]. The state-of-the-art systems in this regard employ deep learning [7, 37, 59, 65] or reinforcement learning [39, 70], and have been used in search engine [11] and machine translation [25, 72]. By means of ranking models one can learn temporal orders and decide whether $t_1 \prec_A t_2$ for *all* tuples t_1, t_2 and attribute A. However, it is hard to justify whether the ranking conforms to the temporal orders in the real world. For data-driven decision making, we need to ensure that the learned orders are reliable. Moreover, these approaches cannot explain the ranking of objects that follow a complex interlinked structure (*e.g.,* address in Figure 1).

Another approach is to employ logic rules, *e.g.,* currency constraints (CCs) [20, 24, 29, 31, 46, 50, 68]. These rules help us deduce temporal orders. Taking Example 1 as an example, the shoe sizes of the same person typically monotonically increases (before 20 years old), and the address of a person may be associated with the marital status (marital status changes from single to married, not the other way around [9]). As will be seen in Section 2, using CCs one can deduce that $t_2 \prec_{\mathsf{SZ}} t_3$ and $t_1 \prec_{\mathsf{address}} t_2$. However, it is hard to find enough rules to deduce relative orders on *each and every* pair of values. For instance, we cannot find rules to decide whether $t_2 \prec_{\mathsf{LN}} t_6$. When testing existing rule-based methods on a dataset with 5% initial timestamps, even the best of them can only deduce 16.3% temporal orders (Section 6), leaving 78.7% undecided. Besides, it is hard to generalize rules to handle lexically different but semantically similar values, since tuples might be extracted from different source (*e.g.,* status in Figure 1: married vs. wed).

Neither deep/ML learning nor logic rules work well when used separately. A natural question is whether it is possible to combine ML models and logic rules in a uniform framework, such that we can learn temporal orders, and use the rules to validate the ranking and improve the learning? How well can this framework improve the accuracy of the ML models and logic rules when being used alone?

Contributions & organization. This paper makes a first effort to answer these questions, all in the affirmative.

(1) Temporal orders (Section 2). We define the notion of temporal orders $t_1 \prec_A t_2$ and $t_1 \preceq_A t_2$ on attributes, and formulate the problem for determining temporal orders. We also review the currency constraints (CCs) of [31], for deducing temporal orders by our framework. We show that CCs can specify interesting temporal properties, *e.g.,* the monotonicity, comonotonicity and transitivity.

(2) GATE (Section 3). We propose GATE (Get the lATEst), a creator-critic framework to determine temporal orders by combining deep learning and logic deduction. GATE iteratively invokes a creator to rank the temporal orders on attribute values, followed by the critic that validates the ranking of the creator and deduces more ranked pairs via discovered CCs. The critic also produces augmented training data for the creator to improve its ranking in the next round. This process proceeds for the creator and critic to mutually enhance each other, until the temporal order cannot be further improved.

(3) Creator (Section 4). We propose a deep learning model underlying the creator of GATE, to learn temporal orders on attribute values. It departs from previous ranking models in that it learns the temporal orders based on contexts (*i.e.,* attribute correlations) and calculates the confidence of the ranking, by employing chronological embeddings and adaptive pairwise ranking strategies.

(4) Critic (Section 5). The critic complements GATE with discovered rules (CCs). We show how it justifies the ranked pairs learned by the creator, and (incrementally) deduces latent temporal orders with the chase [64] using CCs. The chase has the Church-Rosser property (cf. [2]), *i.e.,* it guarantees to converge at the same result no matter in what orders the CCs are applied. Moreover, the critic augments the training data for the creator to improve its model in the next round.

(5) Experimental study (Section 6). Using real-life and synthetic data, we find that (a) the F-measure of GATE on dataset Career [43] is 0.866, versus 0.35 and 0.36 by rule-based UncertainRule [46] and Improve3C [19] (resp. 0.54 and 0.53 by ML-based $\mathsf{RANK}_{\mathsf{Bert}}$ [58] and $\mathsf{Ditto}_{\mathsf{Rank}}$ [49]). (b) On average, GATE is 43.8% and 7.8% more accurate than the critic and the creator, respectively, verifying the effectiveness of combining deep learning and logic rules. (c) GATE is feasible in practice; it only takes 7 rounds to terminate on a real-life dataset of 1,983,698 tuples, with a single machine.

Related work. We categorize the related work as follows.

Rule-based methods. Currency constraints (CCs) were first studied in [30, 31] by employing partial currency orders and later extended in [28, 29] for conflict resolution, by considering both CCs and conditional functional dependencies (CFDs) [27]. A class of currency repairing rules (CRRs) was proposed in [45], which combines logic rules and statistics. A two-step approach was developed in [18] to determining the currency of dynamic data, by means of a topological graph. A class of uncertain currency rules was supported by UncertainRule [46] that considers both temporal orders and data certainty. Improve3C [19] and Imp3C [20] are 4-step data cleaning frameworks, including data consistency, completeness and currency, which use a metric to repair noisy data using CFDs [27] and currency orders. There has also been work [24] on parallel incremental updating algorithms by employing traditional currency rules.

This work differs from the prior work as follows. (a) To the best of our knowledge, we make the first effort to combine deep learning and logic rules for determining currency, for the two to enhance each other, while the prior work at most extends rules with statistic. (b) We propose a deep-learning model for inferring temporal orders and use logic deduction to derive more ranked pairs.

ML models. There have also been efforts on inferring temporal orders by ML models [6, 10, 36, 56, 57, 66]. A related topic is to incorporate temporal information into knowledge graphs, *e.g.,* for temporal link predictions [21, 63] and reasoning [17, 67, 71]. These methods often embed temporal information in ML models, for inference varying over time. Learning temporal orders has been modeled as a learning-to-rank problem (pointwise, pairwise and listwise), where global orders are learned for currency attributes. Temporal problems have also been studied in, *e.g.,* search and recommenda-

tion [32, 69], IR [52] and NLP [49, 58]. In particular, ranking in NLP (*e.g.,* in question-answering tasks) can be handled by pre-trained language models with cross-entropy loss or by a ranking function with a binary classifier as the comparison operator; we adopted a baseline from each kind (*i.e.,* $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$) in Section 6.

In contrast, we propose a creator-critic framework that not only learns temporal orders via deep learning, but also adopts rules to deduce new ones and help ML models learn better. Our creator learns temporal orders based on contexts and iteratively improves its model with augmented training data from the critic (see Section 4 for novelty). The prior models can be embedded into our framework.

Entity resolution (ER). Temporal clustering methods were proposed in [48] based on a concept of time decay, to improve ER with timestamps. [12] dynamically assesses and adjusts similarities among records based on their attributes and temporal differences. [44] estimate the probability that a value changes, and shows how to link tuples in a correct time period. To cope with incorrect timestamps, [68] adopts matching dependencies (MDs) and currency constraints to capture the attributes that were changed over time.

This work focuses on deducing temporal orders of each entity, while the prior work mainly adopts temporal information for ER.

2 DETERMINING TEMPORAL ORDERS

In this section we first formulate temporal orders and the problem for determining temporal orders (Section 2.1). We then review currency constraints (CCs) of [31] and show that such rules are able to express monotonicity, comonotonicity and transitivity (Section 2.2).

2.1 The Problem

Consider a relation schema $R = (\mathsf{EID}, A_1, \ldots, A_n)$, where A_i is an attribute ($i \in [1, n]$), and EID is an entity id as introduced by Codd [15]. For a tuple t of schema R, we denote by $t[A]$ the value in the A-attribute of t, and by $t[\mathsf{EID}]$ the entity represented by tuple t. We assume that each tuple is associated with a tuple id, denoted by tid. We refer to a relation D of R as a *normal instance* D *of* R.

Temporal orders and temporal instances. An *entity instance* is (D_e, T_e), where (a) D_e is a normal instance of schema R such that all tuples t_1 and t_2 in D_e refer to the same real-life entity e and hence, $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$; and (b) T_e is a partial function that associates a timestamp $T_e(t[A])$ with the A-attribute of a tuple t in D_e. We refer to (D_e, T_e) as an *entity instance pertaining to* e.

Here the timestamp indicates that at the time $T_e(t[A])$, the A-attribute value of tuple t is correct and up-to-date; it does not necessarily refer to the time when $t[A]$ was created or last updated. If $T_e(t[A])$ is undefined, a reliable timestamp is not available for $t[A]$.

Intuitively, an entity instance extends a normal instance with available timestamps. Its tuples may be extracted from a variety of data sources, and are identified to refer to the same entity e via entity resolution. In the same tuple t, $t[A]$ and $t[B]$ may bear different timestamps (or even no timestamp) for different A and B. Note that we do not assume a timestamp for the entire tuple, since we often find only parts of a tuple to be correct and up-to-date.

Temporal orders. A *temporal order* on attribute A of D_e is a partial order $\preceq_A$ such that for all tuples t_1 and $t_2 \in D_e$, $t_2 \preceq_A t_1$ if the value in $t_1[A]$ is at least as current as $t_2[A]$. We also use a strict partial order $t_2 \prec_A t_1$ if $t_1[A]$ is more current than $t_2[A]$. To simplify the discussion we focus on $\preceq_A$ in the sequel; $\prec_A$ is handled analogously.

In particular, if $T_e(t_1[A])$ and $T_e(t_2[A])$ are both defined and if $T_e(t_2[A]) \leq T_e(t_1[A])$, *i.e.,* when timestamp $T_e(t_1[A])$ is no earlier than $T_e(t_2[A])$, then $t_2 \preceq_A t_1$, *i.e.,* $t_1[A]$ is confirmed at a later timestamp and is thus considered at least as current as $t_2[A]$.

Temporal order $\preceq_A$ is represented as a set of tuple pairs such that $(t_2[\text{tid}], t_1[\text{tid}]) \in \preceq_A$ iff $t_2 \preceq_A t_1$. We write $(t_2[\text{tid}], t_1[\text{tid}])$ as (t_2, t_1) if it is clear in the context. Note that the same value may bear different timeliness in different tuples, *e.g.,* Mary's marital status changed from married (t_2) to divorced (t_4) to married (t_7, not shown in Figure 1). While $t_2[\text{status}] = t_7[\text{status}]$, $t_2 \prec_{\text{status}} t_7$. Here $t_2 \prec_{\text{status}} t_7$ ranks the timeliness of the status-attributes of tuples t_2 and t_7, not values (married vs. married) detached from the tuples.

We say that a temporal order $\preceq_A^1$ *extends* $\preceq_A^2$, written as $\preceq_A^2 \subseteq \preceq_A^1$, if for all tuples t_1, t_2 in D_e, if $t_2 \preceq_A^2 t_1$ is defined, then so is $t_2 \preceq_A^1 t_1$. That is, $\preceq_A^1$ includes all tuple pairs in $\preceq_A^2$ and possibly more.

Temporal instances. A *temporal instance* D_t *of* R is given as $(D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$, where each $\preceq_{A_i}$ is a temporal order on A_i ($i \in [1, n]$), $D = \bigcup_{i \in [k]} D_{e_i}$, $T = \bigcup_{i \in [k]} T_{e_i}$, and for all $i \in [1, k]$, (D_{e_i}, T_{e_i}) is an entity instance of R. Here tuples t_1 and t_2 in D are *compatible* under $\preceq_A$ if they pertain to the same entity, *i.e.,* $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$.

Intuitively, D_t is a collection of entity instances, such that each (D_{e_i}, T_{e_i}) pertains to the same entity e_i. We do not rank the currency of tuples if they refer to different entities. A temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ is said to *extend* another temporal instance $D_t' = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T)$ if for all $i \in [1, n]$, $\preceq_{A_i}$ extends $\preceq'_{A_i}$.

Problem statement. We study the problem for *determining the temporal orders* of a temporal instance, stated as follows.

- *Input*: A temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$.
- *Output*: An extended temporal instance $D_t' = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T)$ such that for all $i \in [1, n]$, (a) $\preceq'_{A_i}$ extends $\preceq_{A_i}$ and (b) $\preceq'_{A_i}$ is a total order on all compatible t_1 and t_2 with $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$.

Intuitively, our goal is to extend $\preceq_{A_i}$ such that for all tuples t_1 and t_2 in D, if $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$, we can decide which of $t_1[A_i]$ and $t_2[A_i]$ is more up-to-date. As a consequence, we can deduce the latest value for all attributes. Note that we define a total order on *each* attribute, not a global order on tuples. The total orders on different attributes can be different. This said, we can use the temporal orders learned on one attribute to help the deduction on other attributes, via correlation expressed as currency constraints (see below).

2.2 Currency Constraints

We next review the class of currency constraints proposed in [31]. The predicates over a relation schema R are defined as:

$$p ::= t[A] \oplus c \mid t_1[A] \oplus t_2[A] \mid t_1 \preceq_A t_2,$$

where t, t_1, t_2 are tuple variables denoting tuples of R, A is an attribute of R, c is a constant, $\oplus$ is an operator from $\{=, \neq, >, \geq, <, \leq\}$; $t[A] \oplus c$ and $t_1[A] \oplus t_2[A]$ are defined on attribute values, while $t_1 \preceq_A t_2$ compares the timeliness of $t_1[A]$ and $t_2[A]$. In particular, $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$ says that t_1 and t_2 refer to the same entity.

A *currency constraint* (CC) over schema R is defined as follows:

$$\varphi = X \rightarrow p_0,$$

where X is a conjunction of predicates over R with tuple variables

$t_1, \ldots, t_m$, and p_0 has the form $t_u \preceq_{A_i} t_v$ for $u, v \in [1, m]$. We refer to X as the *precondition* of φ, and p_0 as the *consequence* of φ.

As defined in [31], a CC can be equivalently expressed as a universal first-order logic sentence of the following form:

$$\varphi = \forall t_1, \ldots, t_m \big(\bigwedge_{j \in [1,m]} (t_1[\mathsf{EID}] = t_j[\mathsf{EID}]) \wedge X \rightarrow t_u \preceq_A t_v \big).$$

Semantics. Currency constraints are defined over temporal instances $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ of R. A *valuation* of tuple variables of a CC φ in $\mathcal{D}_t$, or simply *a valuation of* φ, is a mapping h that instantiates variables $t_1, \ldots, t_m$ with tuples in D that refer to the same real-world entity, as required by $t_1[\mathsf{EID}] = t_j[\mathsf{EID}]$.

A valuation h *satisfies* a predicate p over R, written as $h \models p$, if the following is satisfied: (1) if p is $t[A] \oplus c$ or $t_1[A] \oplus t_2[A]$, then it is interpreted as in tuple relational calculus following the standard semantics of first-order logic [2]; and (2) if p is $t_1 \preceq_A t_2$, then $t_2[A]$ is at least as current as $t_1[A]$ *i.e.,* (t_1, t_2) is in $\preceq_A$.

For a conjunction X of predicates, we write $h \models X$ if $h \models p$ for *all* p in X. A temporal instance D_t *satisfies* CC φ, denoted by $D_t \models \varphi$, if for *all* valuations h of $X \rightarrow p_0$ in D_t, if $h \models X$ then $h \models p_0$.

Properties. Currency constraints are able to specify interesting temporal properties. Below we exemplify some properties.

Monotonicity. A temporal order $\preceq_A$ over relation schema R is *monotonic* if for any tuples t_1 and t_2 of R that refer to the same entity, if $t_1[A] \leq t_2[A]$ then $t_1 \preceq_A t_2$. For instance, consider the SZ (shoe size) attribute of the customer relation of Example 1. Then $\preceq_{\mathsf{SZ}}$ is monotonic. It can be expressed as the following CC:

$$\varphi_1 = t_1[\mathsf{SZ}] \leq t_2[\mathsf{SZ}] \rightarrow t_1 \preceq_{\mathsf{SZ}} t_2.$$

As another example, marital status only changes from single to married, not the other way around [9]. This is expressed as CC:

$$\varphi_2 = t_1[\mathsf{status}] = \text{“single”} \wedge t_2[\mathsf{status}] = \text{“married”} \rightarrow t_1 \preceq_{\mathsf{status}} t_2.$$

With slight abuse of terminologies by considering timestamps as an “attribute” associated with attribute A, we have:

$$\varphi_3 = T_e(t_1[A]) \leq T_e(t_2[A]) \rightarrow t_1 \preceq_A t_2,$$

since $t_2[A]$ is confirmed at a later timestamp.

Comonotonicity. For attributes A and B of R, we say $\preceq_A$ and $\preceq_B$ are *comonotonic* in a temporal instance D_t of R if for all tuples t_1 and t_2 in D_t that refer to the same entity, $t_1 \preceq_A t_2$ if and only if $t_1 \preceq_B t_2$.

Intuitively, $\preceq_A$ and $\preceq_B$ are comonotonic if the two are correlated such that when one is updated, the other will also change. For instance, $\preceq_{\mathsf{status}}$ and $\preceq_{\mathsf{address}}$ are often comonotonic: when the marital status of a person changes from single to married, this person may move to a larger house. This can be expressed as a CC:

$$\varphi_4 = t_1 \preceq_{\mathsf{status}} t_2 \rightarrow t_1 \preceq_{\mathsf{address}} t_2.$$

Transitivity. Transitivity can also be expressed for any attribute A:

$$\varphi_5 = t_1 \preceq_A t_2 \wedge t_2 \preceq_A t_3 \rightarrow t_1 \preceq_A t_3.$$

Correlating different attributes. One can correlate multiple different attributes to capture implicit temporal ordering. For example, marital status may change from “married" to “divorced" and further from “divorced" to “married" again. To deduce an order on $t[\mathsf{status}]$, we can use the number of kids as an additional condition:

$$\varphi_6 = t_1[\mathsf{status}] = \text{“married”} \wedge t_2[\mathsf{status}] = \text{“divorced”} \wedge t_1[\mathsf{kids}] < t_2[\mathsf{kids}] \rightarrow t_1 \preceq_{\mathsf{status}} t_2.$$

Deduction. Making use of CCs, we can deduce certain temporal orders, *e.g.,* from φ_1-φ_6 and Figure 1, we can deduce the following:

- $t_1 \preceq_{\mathsf{status}} t_2$ by φ_2, $t_2 \preceq_{\mathsf{status}} t_4$ by φ_6 and $t_1 \preceq_{\mathsf{status}} t_4$ by φ_5;
- $t_1 \preceq_{\mathsf{address}} t_2$ by φ_4; hence $t_2[\mathsf{address}]$ is more current for Mary;
- $t_1 \preceq_{\mathsf{SZ}} t_2 \preceq_{\mathsf{SZ}} t_3$ by φ_1; hence Mary's current shoe size is 7.

However, we cannot determine whether $t_2 \preceq_{\mathsf{LN}} t_6$ or $t_6 \preceq_{\mathsf{LN}} t_2$ by deduction with the currency constraints φ_1-φ_6.

Discovery of currency constraints. As noted in [31], CCs can be considered as a special case of denial constraints (DCs) [3] extended with temporal orders $\preceq_A$. Several methods have been developed for discovering DCs automatically, *e.g.,* FastDC [13], Hydra [4], DCFinder [60] and ADCMiner [54]. We can readily extend these algorithms for discovering CCs (see [1] for details). Note that the discovery algorithm is executed once for each relation schema R by sampling its temporal instances. The set Σ of discovered CCs is then applied to different temporal instances. In other words, GATE does not have to discover CCs for each input temporal instance.

3 GATE: A CREATOR-CRITIC SYSTEM

In this section, we propose a creator-critic framework for determining temporal orders, and develop system GATE to implement it. A unique feature of GATE is its combined use of deep learning and logic deduction. Below we start with the architecture of GATE, and then present its overall workflow, with termination guarantee.

Architecture. The ultimate goal of GATE is to obtain a total order $\preceq_A$ for each attribute A. As shown in Figure 2, GATE first discovers a set Σ of CCs on D_t *offline* for performing logic deduction. Then it takes a temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ as input, and learns and deduces more temporally ranked pairs for D_t *online*. For simplicity, we assume *w.l.o.g.* that D consists of a single entity instance D_e, *i.e.,* all tuples in D pertain to the same entity e and thus their attributes can be pairwisely compared; the methods of this paper can be readily extended to D_t with multiple entity instances.

More specifically, the learning and deducing process in GATE *iteratively* executes two phases, namely, creator and critic, as follows.

(1) Creator (Sections 4). In this phase, GATE (incrementally) trains a ranking model $\mathcal{M}_{\mathsf{rank}}$ via deep learning. By taking D_t and the augmented training data D_{aug} (see its definition shortly) from the critic as input, it predicts new orders for extending each $\preceq_A$ of D_t. Our model has three features: (a) For each tuple t in D, the embedding for $t[A]$ is created using both $t[A]$ and other correlated values in t, so that $t[A]$ can be ranked comprehensively. (b) The attribute embeddings [16] are arranged to preserve chronological orders. (c) We adopt an *attribute-centric adaptive pairwise ranking* strategy in $\mathcal{M}_{\mathsf{rank}}$, so that the ranking result can be justified semantically. Moreover, the temporal instance D_t will also be extended based on D_{aug}.

Given an attribute A, we associate each $(t_1, t_2) \in \preceq_A$ with a *confidence*, denoted by $\mathsf{conf}(t_1 \preceq_A t_2)$, indicating how likely $t_1 \preceq_A t_2$ holds. If $t_2[A]$ has a later timestamp than $t_1[A]$, then $\mathsf{conf}(t_1 \preceq_A t_2)$ is 1. If $t_1 \preceq_A t_2$ is predicted by $\mathcal{M}_{\mathsf{rank}}$, its confidence is from 0 to 1. We only consider (t_1, t_2) predicted by $\mathcal{M}_{\mathsf{rank}}$ with $\mathsf{conf}(t_1 \preceq_A t_2) \geq \delta$, where δ is a predefined threshold, as *candidate pairs* to be extended to $\preceq_A$. We denote the set of all candidate pairs of $\preceq_A$ by $\preceq_A^{\mathcal{M}}$.

The input and output of the creator are as follows:

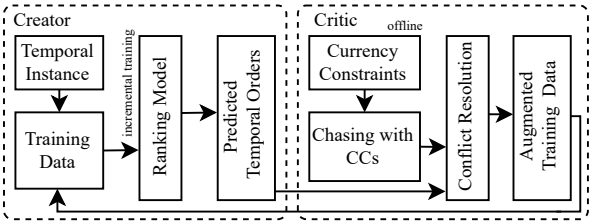


Figure 2: Architecture of GATE

- *Input*: A temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ and the augmented training data D_{aug} from the critic.
- *Output*: An extended temporal instance $D'_t = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T)$ based on D_{aug} and the predicted orders $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n})$.

(2) Critic (Sections 5). In this phase, the critic of GATE justifies and deduces more temporally ranked pairs, by applying CCs in Σ via the *chase* [64]. Denote the result of chasing by $\preceq^{\Sigma}_{A}$. Depending on the validity of $\preceq^{\Sigma}_{A}$, we construct an augmented training data, denoted by D_{aug}, containing the ranked pairs justified (resp. conflicts caught by CCs); D_{aug} will be fed back to the creator, for the next round of model learning, so that the creator can learn from more unseen data and get higher accuracy iteratively. Specifically, D_{aug} is a temporal instance extended with a validity flag $\mathsf{f}_{\text{valid}}$, *i.e.,* $D_{\text{aug}} = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}})$: (a) if $\mathsf{f}_{\text{valid}}$ is true, $\preceq^{\Sigma}_{A}$ is valid and all temporal order deduced by the chase will be added to D_{aug}; and (b) otherwise, the chasing is invalid, *i.e.,* both $(t_1, t_2) \in \preceq_A$ and $(t_2, t_1) \in \preceq_A$ are deduced, and either $t_1 \prec_A t_2$ and $t_2 \prec_A t_1$. In this case, these two *conflicting* orders will be added to D_{aug}, and the creator will be asked to resolve this conflict, by revising its model accordingly.

Formally, the input and output of the critic are as follows:

- *Input*: A temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$, the predicted temporal orders $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n})$ and a set Σ of CCs.
- *Output*: Augmented data $D_{\text{aug}} = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}})$.

The novelty of the critic consists of (a) the deduction using the chase, and (b) an efficient algorithm implementing the chase.

Workflow. As shown in Figure 3, GATE takes a temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ and a set Σ of CCs discovered offline as input, and it outputs an extended temporal instance $D'_t = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T)$ with a total order $\preceq'_A$ defined for each attribute A.

GATE first initializes the augmented training data $D_{\text{aug}} = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}} = \text{true})$ (Line 1, details omitted) for the first round of GATE, by deducing its initial temporal orders $\preceq'_A$ via CCs in Σ whose preconditions do not involve timeliness comparison, *e.g.,* we can create temporal orders for those tuples with available timestamps using φ_3. The initialization can be done efficiently by [34].

Then GATE iteratively executes the creator and the critic in *rounds*. In the i-th round, (a) the creator *incrementally* trains $\mathcal{M}_{\mathsf{rank}}$ (Line 4, see Section 4) based on the augmented training data D_{aug} returned by the critic in the $(i-1)$-th round. Besides, the creator extends the temporal instance D_t (Line 5, see Section 4) based on D_{aug}, by possibly revising its model to resolve the conflicts. The creator then predicts new temporal orders $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n})$ (Line 6) based on the fine-tuned model, which are candidate pairs (t_1, t_2) with confidences at least δ; and (b) the critic deduces more ranked pairs $(\preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n})$ by chasing with CCs in Σ (Line 8). Based on

Input: A temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$, and a set Σ of CCs.
Output: An extended temporal instance $D'_t = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T)$ such that for all $i \in [1, n]$, (a) $\preceq'_{A_i}$ extends $\preceq_{A_i}$ and (b) $\preceq_{A_i}$ is a total order.

1. $D_{\text{aug}} :=$ Initialize$(D_t, \mathsf{f}_{\text{valid}} = \text{true}, \Sigma)$; Initialize the ranking model $\mathcal{M}_{\mathsf{rank}}$;
2. **while** true **do**
3. /* The Creator of GATE */
4. Train $\mathcal{M}_{\mathsf{rank}}$ incrementally based on D_{aug};
5. $D_t :=$ Extend(D_t, D_{aug});
6. $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n}) :=$ the predicted temporal orders by $\mathcal{M}_{\mathsf{rank}}$;
7. /* The Critic of GATE */
8. $(\preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n}) :=$ Chase(D_t, Σ);
9. $D_{\text{aug}} :=$ ConstructAugmented$(D, \preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n}, \preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n}, T)$;
10. **if** D_t no longer changes **then**
11. **break**;
12. $D_t =$ Extend$(D_t, \mathcal{M}_{\mathsf{rank}})$;
13. **return** D_t;

Figure 3: Workflow of GATE

the result of chasing, the critic constructs augmented training data D_{aug} via procedure ConstructAugmented (Line 9, see Section 5); this D_{aug} is fed back to the creator for the next round.

Finally, when D_t no longer changes (Line 10-11), the iteration ends. If D_t is still not a temporal instance with total orders defined on all attributes, we extend it using procedure Extend (Line 12), such that for each pair (t_1, t_2), one of (t_1, t_2) and (t_2, t_1) learned with a higher confidence is in $\preceq_A$, until each $\preceq_A$ becomes a total order.

Termination. We prove in [1] that eventually, GATE will terminate (Line 10), *i.e.,* with more iterations, D_t is gradually extended for more orders and finally, becomes stable and does not change.

Theorem 1: GATE *is guaranteed to terminate.* □

Example 2: Continuing with Example 1, assume that $\Sigma = \{\varphi_1\text{-}\varphi_6\}$ and D_t has empty temporal orders. We first initialize the training data D_{aug} by applying CCs in Σ that do not compare timeliness in their preconditions, *e.g.,* we can deduce $t_5 \preceq_{\mathsf{job}} t_6$ by φ_3. Suppose that after training $\mathcal{M}_{\mathsf{rank}}$ based on the initial D_{aug}, no tuple pair predicted by $\mathcal{M}_{\mathsf{rank}}$ is at least δ-confident in the first round. The creator outputs empty $\preceq^{\mathcal{M}}_{A}$ for each A due to the lack of information. Then by applying the CCs in Σ, the critic can deduce new temporal orders $\preceq^{\Sigma}_{A}$, *e.g.,* $t_1 \preceq_{\mathsf{address}} t_2$ by φ_4 and $t_1 \preceq_{\mathsf{status}} t_4$ by φ_5. Both $\preceq^{\mathcal{M}}_{A}$ and $\preceq^{\Sigma}_{A}$ will be used to construct the augmented training data D_{aug}, based on which the creator extends D_t and incrementally trains $\mathcal{M}_{\mathsf{rank}}$ in the second round. This time the creator might be able to predict confident temporal orders, *e.g.,* $t_1 \preceq_{\mathsf{LN}} t_2$, with the augmented training data D_{aug}. The iteration continues until D_t does not change anymore. Each temporal order $\preceq_A$ in D_t will be extended to a total order by $\mathcal{M}_{\mathsf{rank}}$ if it is still not total. □

Remark. We adopt a confidence threshold δ to ensure the reliability of ML predictions, so that only reliable orders are considered. When confident orders cannot be decided for the lack of initial information, we may opt to invite user inspection to ensure the correctness of a few initial ranked pairs, from which more reliable orders can be iteratively deduced/learned. The parameter δ plays an important role in model $\mathcal{M}_{\mathsf{rank}}$, *e.g.,* when testing $\mathcal{M}_{\mathsf{rank}}$ on procedural billing data of patients during their ICU stays (with 82.28% *real* missing timestamps [42]), we find that 22.6% ranked pairs are identified as confident orders when $\delta = 0.55$. The percentage decreases to 16.7% when δ increases to 0.6. We will test the impact of δ in Section 6.

4 CREATOR

In this section, we develop the creator of GATE. Given a temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ and the augmented training data D_{aug} (from the critic), the creator (a) trains a ranking model $\mathcal{M}_{\text{rank}}$ to predict new orders and (b) extends each $\preceq_A$ of D_t based on D_{aug}.

Review on ranking models. *Learning to Rank* [52] aims to learn a ranking model so that objects can be ranked based on their degrees of relevance, preference, or importance (in our setting, timeliness).

We adopt pairwise ranking since (a) its semantics is consistent with temporal orders, which is a set of *tuple pairs* on which partial orders are defined (Section 2); and (b) by transitivity of temporal orders, pairwise ranking helps us get a total order for all attributes.

Challenges. One naturally wants to adopt an existing model. However, it is hard to directly apply one, since the unique features of temporal orders are not well considered, resulting in poor performance.

(1) Attribute correlation. Due to the correlated nature of temporal order, we often need to reference other attributes to determine the orders of a given attribute. Moreover, since a value may change back and forth (*e.g.,* the marital status changes from "married" to "divorced", and back from "divorced" to "married"), it is hard to determine the up-to-date value based on a single attribute only.

(2) Limitation of embedding models. To determine the timeliness, care should be taken for lexically different but semantically similar values (*e.g.,* "baby" vs. "birth" and "dead" vs. "expired" for attribute status). Although existing embedding models (*e.g.,* ELMo [61] or Bert [16]) are widely adopted, they cannot be directly used here since they are not trained to organize data chronologically.

(3) Adaptive margin. Existing ranking strategies do not consider real-life characteristics of timeliness, *e.g.,* the timespan for a person's status to move from "birth" to "engaged" is typically longer than from "engaged" to "married" (Figure 4). Instead of ranking status with a *fixed* margin as most existing strategies did, we need a new methodology to embed values using *adaptive* margins, to conform to their real-life behaviors and justify the semantic of ranking.

Model overview. We propose a ranking model to tackle the above challenges, whose novelty includes (a) a context-aware scheme that embeds each value along with other correlated values, (b) an encoding mechanism to re-organize the embeddings in a chronological manner, and (c) an attribute-centric adaptive ranking strategy.

As shown in Figure 4, our ranking model $\mathcal{M}_{\text{rank}}$ takes the current D_t as input, and outputs new ranked pairs, where each (t_1, t_2) is associated with a *confidence*, indicating how likely $t_1 \preceq_A t_2$ holds.

Our ranking model consists of three stages as follows: context-aware embedding, chronological encoding and order prediction.

(1) $\mathcal{M}_{\text{rank}}$ first builds a context-aware embedding for each attribute value using pre-trained language models (ELMo [61] or Bert [16]), in a way that information of correlated values is also embedded.

(2) Based on the embeddings, $\mathcal{M}_{\text{rank}}$ encodes a target vector ϕ_A for attribute A, via *non-linear transformation* (see below). Similarly, a value vector $\phi_{t[A]}$ is encoded for each A-attribute value of tuple t, so that (a) if $t[A]$ is more current, $\phi_{t[A]}$ is closer to ϕ_A and (b) the gap between $\phi_{t[A]}$ and ϕ_A is trained adaptively, to reflect real semantics.

(3) Finally, given $\phi_{t_1[A]}$ and $\phi_{t_2[A]}$ of t_1 and t_2, $\mathcal{M}_{\text{rank}}$ predicts the order, *i.e.,* whether $t_1 \preceq_A t_2$ holds with high enough confidence.

We next briefly elaborate the context-aware embedding and the chronological encoding scheme with the adaptive margin.

Context-aware embedding. To reference correlated values, we treat tuples as sequences and adopt the idea of serialization [49] (so that tuples can be meaningfully ingested by models) to embed values.

Following [49], given a tuple t in D_t, we serialize its values:

$$\text{serialize}(t) = \langle \text{COL} \rangle A_1 \langle \text{VAL} \rangle t[A_1] \ldots \langle \text{COL} \rangle A_n \langle \text{VAL} \rangle t[A_n],$$

where $\langle \text{COL} \rangle$ and $\langle \text{VAL} \rangle$ are special tokens, denoting the start of attribute and value, respectively (see Figure 4). This serialization is fed as input to a pre-trained language model $\text{emb}(\cdot)$ to compute a d-dimensional embedding for each A-attribute value, denoted by $\text{emb}(t[A]) \in \mathbf{R}^d$. Besides, we average out the embedding vectors for all $t[A]$ to get a context representation of tuple t, *i.e.,* $\text{emb}(t) = \frac{1}{n} \sum_{i=1}^{n} \text{emb}(t[A_i])$. Finally for each A-attribute value of t, we get the context-aware embedding of $t[A]$, denoted by $E_{t[A]} \in \mathbf{R}^{2d}$:

$$E_{t[A]} = [\text{emb}(t[A]); \text{emb}(t)],$$

where $[;]$ denotes vector concatenation. In this way, $E_{t[A]}$ embeds not only the A-attribute value, but also the contextual information from other attribute values, to allow comprehensive ranking. Similarly, a schema embedding for each attribute A is computed: $E_A = \text{emb}(A)$, by feeding the attribute name, *e.g.,* status, to the model.

Chronological encoding with adaptive margins. While pre-trained embeddings are widely adopted to capture semantics, they are not trained to organize temporal orders. Thus we propose chronological encoding to re-organize the embeddings to preserve timeliness. The idea is to use schema embedding as the target and make the embedding of a more current value closer to the target; moreover, instead of ranking in fixed margins, embeddings are ordered adaptively.

Specifically, given the embedding of the A-attribute of t, *i.e.,* $E_{t[A]}$, we encode it using a context encoder $\text{ENC}_{\text{cxtx}}(\cdot)$ as follows:

$$\phi_{t[A]} = \text{ENC}_{\text{cxtx}}(E_{t[A]}) = \sigma(\mathsf{W}_2 * \sigma(\mathsf{W}_1 * E_{t[A]})),$$

where W_1 and W_2 are learnable parameters of the encoder, and σ is the non-linear sigmoid activation function given by $\sigma(x) = \frac{1}{1+e^{-x}}$.

Similarly, the target vector for attribute A is encoded as $\phi_A = \text{ENC}_{\text{attr}}(E_A)$, where $\text{ENC}_{\text{attr}}(\cdot)$ denotes the schema encoder.

To train the encoders with ordered embeddings and adaptive margins, we adopt an attribute-centric adaptive margin-based loss. Given $\preceq_A$ in D_t, the loss on A is formulated as follows:

$$\text{loss}(A) = \sum_{(t_1,t_2)\in\preceq_A} \Big\{ -\tanh(\langle \phi_{t_2[A]}, \phi_A \rangle) + \text{ReLU}(\gamma_{t_1,t_2} + \tanh(\langle \phi_{t_1[A]}, \phi_{t_2[A]} \rangle)) \Big\},$$

where $\langle \cdot, \cdot \rangle$ is the inner product and γ_{t_1,t_2} is the adaptive margin between the two tuples; we set γ_{t_1,t_2} to be $1 - \cos(v_{t_1[A]}, v_{t_2[A]})$ in practice, where $v_{t_1[A]}$ and $v_{t_2[A]}$ are the Word2Vec [14] embeddings which characterize the co-occurrence of $t_1[A]$ and $t_2[A]$.

Intuitively, by minimizing the loss, for each training instance $t_1 \preceq_A t_2$, (a) we make the encoded vector of the more current value $t_2[A]$ closer to target ϕ_A (the first term) and (b) we ensure an adaptive margin γ_{t_1,t_2} between the two tuples (the second term). In other words, the attribute values in the encoded space are not only arranged chronologically by their distances to ϕ_A, from which temporal orders can be easily derived, their margins are also adaptively

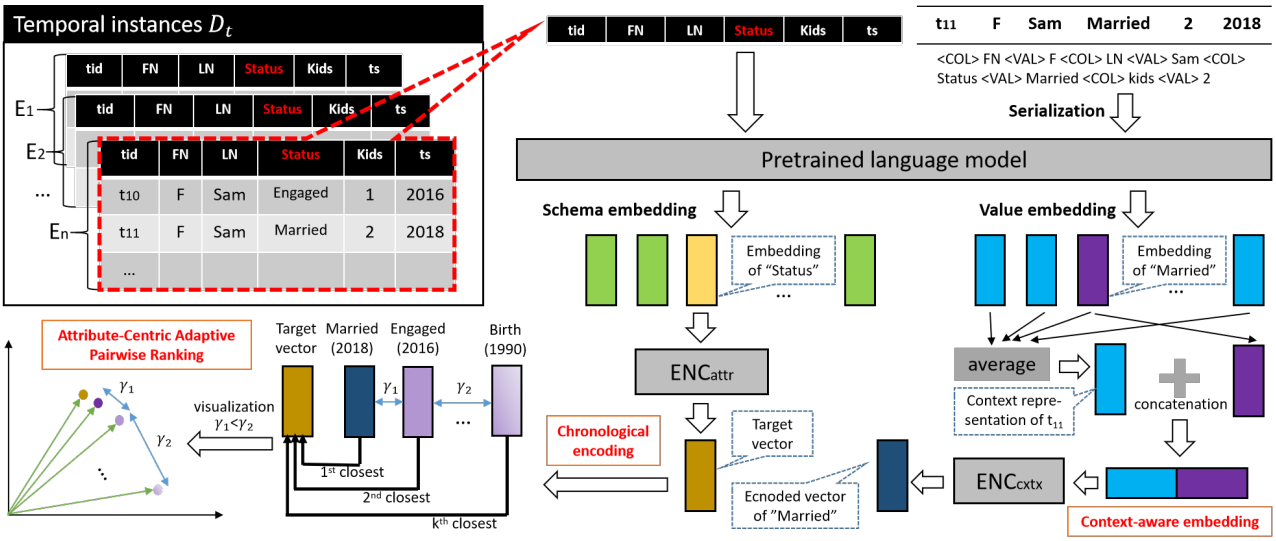


Figure 4: The Network Architecture of Creator

determined, to reflect the semantic of timeliness ranking.

Model training and instance extension. In each round, the creator receives augmented training data $D_{\text{aug}} = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}})$, based on which it (a) incrementally trains $\mathcal{M}_{\text{rank}}$ via back-propagation and (b) extends the temporal instance D_t with D_{aug}.

In the first round, D_{aug} is initialized to be the temporal orders constructed by applying those CCs in Σ without timeliness comparison in their preconditions [34]. In the following rounds, D_{aug} is constructed by the critic, based on the result of chasing with CCs.

Specifically, depending on flag $\mathsf{f}_{\text{valid}}$ in D_{aug}, we have two cases:

(1) If $\mathsf{f}_{\text{valid}}$ is true, the result of chasing is valid and $\mathcal{M}_{\text{rank}}$ is incrementally trained on the orders in D_{aug} (see below). Moreover, the temporal instance $D_t = (D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$ is extended with the ranked pairs in D_{aug}, *i.e.,* for each (t_1, t_2) in $\preceq'_A$ of D_{aug}, (t_1, t_2) is added to $\preceq_A$. In this case, we say that (t_1, t_2) becomes *stable*. Once a temporal order becomes stable, it will not be removed from D_t.

(2) If $\mathsf{f}_{\text{valid}}$ is false, the result of chasing is invalid, *i.e.,* there is an attribute A such that conflicting orders (t_1, t_2) and (t_2, t_1) are both in $\preceq'_A$ of D_{aug} *i.e.,* $\{(t_1, t_2), (t_2, t_1)\} \subseteq \preceq'_A$, and either $t_1 \prec_A t_2$ or $t_2 \prec_A t_1$. In this case, we decide that either (t_1, t_2) or (t_2, t_1) is added to $\preceq_A$ using $\mathcal{M}_{\text{rank}}$, with a higher confidence (and possibly user inspection). Assume *w.l.o.g.* that (t_1, t_2) is added to $\preceq_A$ (*i.e.,* it becomes stable). Then, the creator fine-tunes $\mathcal{M}_{\text{rank}}$ so that t_1 and t_2 are better separated in the encoded space (see [1] for details).

Incremental training. Incremental training of $\mathcal{M}_{\text{rank}}$ might lead to the catastrophic forgetting issue [41], *i.e.,* the model might forget some temporal orders learned in prior rounds. To overcome this, we adopt a simple strategy to retain prior knowledge: In each round, $\mathcal{M}_{\text{rank}}$ first makes inference to previously learned orders, and then extracts those that make wrong predictions. Then, we add these orders to the augmented data D_{aug} and then train the model on D_{aug}. In this way, the model is able to learn from previous training instances and alleviate the impact of the catastrophic forgetting issue.

Monotonicity. One can verify that the number of stable temporal orders in D_t is (strictly) monotonically increasing when more rounds GATE are performed (see a detailed lemma in [1]). The termination of GATE (Theorem 1) partly depends on this monotonicity.

Confidence. Given $\preceq_A$ and a tuple pair (t_1, t_2), $\mathcal{M}_{\text{rank}}$ predicts $(t_1, t_2) \in \preceq_A$ if $\tanh(\langle \phi_{t_2[A]}, \phi_A \rangle) > \tanh(\langle \phi_{t_1[A]}, \phi_A \rangle)$; its confidence indicates how likely $t_1 \preceq_A t_2$ holds. We compute it to be

$$\text{conf}(t_1 \preceq_A t_2) = \sigma(\tanh(\langle \phi_{t_2[A]}, \phi_A \rangle) - \tanh(\langle \phi_{t_1[A]}, \phi_A \rangle))$$

and set $\text{conf}(t_2 \preceq_A t_1)$ to 0. Thus, given a confidence threshold $\delta > 0$, we will not predict both $t_1 \preceq_A t_2$ and $t_2 \preceq_A t_1$ as confident orders.

Intuitively, we use $\tanh(\langle \phi_{t[A]}, \phi_A \rangle)$ to measure the "distance" between $\phi_{t[A]}$ and ϕ_A, where the closer one is more current. The distance gap between $\phi_{t_1[A]}$ and $\phi_{t_2[A]}$ quantifies the confidence: the larger the gap, the larger the confidence, which ranges from 0 to 1.

Example 3: Consider the example in Figure 4, where we focus on attribute status. After creating the context-aware embedding based on a pre-trained model, it chronologically encodes a target vector ϕ_{status} and value vectors for all values, so that they are arranged by their distances to ϕ_{status}. To illustrate, we also label the *unknown* timestamp of each value vector in the figure (*e.g.,* $T_e(\text{Married}) = 2018$). Since ϕ_{Married} is the closest to ϕ_{status}, it is predicted to be the latest status value and new ranked pairs are constructed accordingly for $\preceq_{\text{status}}$, as augmented training data in the next round. □

Remark. Our creator learns temporal orders by utilizing context-aware embedding, chronological encoding and attribute-centric adaptive ranking. However, it does not explicitly take into account of some temporal properties, such as the transitivity. This motivates us to use critic to deduce and justify the temporal orders based on the semantics of the data, as will be presented in the next section.

5 CRITIC

In this section, we develop the critic under GATE for justifying and deducing temporal orders. Taking a temporal instance D_t, the temporal orders $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n})$ predicted by the creator and a set Σ of mined CCs as input, the critic (a) deduces more ranked pairs $(\preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n})$ by applying CCs via the *chase*, and (b) constructs the augmented training data D_{aug} and feeds D_{aug} back to the creator.

5.1 Chasing with CCs

We extend the classic chase in [33] for CCs under GATE. In the following, we first specify fixes and ground truth. We then present the chase for CCs, with the Church-Rosser property.

Fixes. We extend temporal orders in D_t by applying CCs in Σ to deduce *fixes*, which are modeled as sets of ranked pairs, denoted by $\overline{U} = (\overline{U}_{A_1}, \ldots, \overline{U}_{A_n})$, where each ranked pair (t_1, t_2) in $\overline{U}_A$ is referred to as a *fix* and it means that $t_1 \preceq_A t_2$ or $t_1 \prec_A t_2$ is deduced. We only apply a CC if its precondition is satisfied by a collection Γ of "ground truth". Intuitively, the fixes are logical consequences of Σ and Γ, *i.e.,* as long as the CCs in Σ and Γ are correct, so are the fixes.

Validity. We say $\overline{U}$ is *valid* if it has no conflicting fixes, *i.e.,* there exist no attribute A and tuples t_1, t_2 such that $(t_1, t_2) \in \overline{U}_A$ and $(t_2, t_1) \in \overline{U}_A$ at the same time, and either $t_1 \prec_A t_2$ or $t_2 \prec_A t_1$.

Ground truth. To justify the correctness of fixes, we maintain and employ a collection $\Gamma = (\Gamma_{A_1}, \ldots, \Gamma_{A_n})$ of validated data, *enclosed* in $\overline{U}$. In our setting, block Γ is initialized by applying CCs in Σ whose preconditions do not involve timeliness comparison (*e.g.,* initial temporal orders with partial timestamps via φ_3), and is iteratively expanded with the temporal orders learned by the creator with confidence above threshold δ or deduced by the chase in the critic.

The chase. Given a temporal instance D_t, the chase deduces fixes by chasing D_t with CCs in Σ and ground truth in Γ. It uses sets $\preceq^{\Sigma} = (\preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n})$ to keep track of the affected fixes in $\overline{U}$. Specifically, the i-th chase step of D_t by Σ at $(\overline{U}_i, \preceq^{\Sigma}_i)$ is:

$$(\overline{U}_i, \preceq^{\Sigma}_i) \Rightarrow_{(\varphi,h)} (\overline{U}_{i+1}, \preceq^{\Sigma}_{i+1}),$$

where $\varphi : X \rightarrow p_0$ is a CC in Σ, h is a valuation of φ in $\mathcal{D}_t$, and the application of (φ, h) should satisfy the following conditions:

(1) All predicates $p \in X$ are *validated*, *i.e.,* if p is $t[A] \oplus c$ or $t_1[A] \oplus t_2[A]$, then $h \models p$; and if p is $t_1 \preceq_A t_2$, then (t_1, t_2) is in $\overline{U}_A$.

(2) The consequence $p_0 : t_1 \preceq_A t_2$ extends $\overline{U}_i$ to $\overline{U}_{i+1}$, such that (t_1, t_2) is added to $\overline{U}_A$ of $\overline{U}_i$; similarly, p_0 extends $\preceq^{\Sigma}_i$ to $\preceq^{\Sigma}_{i+1}$.

Chasing. Starting from a set $\overline{U}_0$ of fixes, initialized to be Γ, and an empty $\preceq^{\Sigma}_0$, a chasing sequence ξ of D_t by (Σ, Γ) is

$$(\overline{U}_0, \preceq^{\Sigma}_0), \ldots, (\overline{U}_k, \preceq^{\Sigma}_k),$$

where $(\overline{U}_i, \preceq^{\Sigma}_i) \Rightarrow_{(\varphi,h)} (\overline{U}_{i+1}, \preceq^{\Sigma}_{i+1})$ is a valid chase step, *i.e.,* a valuation h of φ extends $(\overline{U}_i, \preceq^{\Sigma}_i)$ to $(\overline{U}_{i+1}, \preceq^{\Sigma}_{i+1})$ where $\overline{U}_{i+1}$ is valid.

The chasing sequence is *terminal* if there exist no CC φ in Σ and valuation h of φ such that (φ, h) leads to another valid chase step.

A chase sequence ξ terminates in one of the following cases:

(1) No more CCs in Σ can be applied. If so, we say that ξ is valid, with $(\overline{U}_k, \preceq^{\Sigma}_k)$ as its result.

(2) Either $\overline{U}_0$ is invalid or there exist $\varphi, h, \overline{U}_{k+1}$ and $\preceq^{\Sigma}_{k+1}$ such that $(\overline{U}_k, \preceq^{\Sigma}_k) \Rightarrow_{(\varphi,h)} (\overline{U}_{k+1}, \preceq^{\Sigma}_{k+1})$ but $\overline{U}_{k+1}$ is invalid. Such ξ is invalid, and the result of the chase is $\perp$ (undefined).

Intuitively, the chase helps us deduce more ranked pairs when it terminates with enriched $(\overline{U}_k, \preceq^{\Sigma}_k)$; moreover, it justifies and explains the learned order if no invalid chase step is taken. When its result is $\perp$, it detects invalid ranked pairs of the learner.

Example 4: Consider D_t in Figure 1. Assume that $\Sigma = \{\varphi_1\text{-}\varphi_6\}$ and $\preceq^{\Sigma} = (\preceq^{\Sigma}_{A_1}, \ldots, \preceq^{\Sigma}_{A_n})$, where each $\preceq^{\Sigma}_{A_i}$ is empty. We initialize $\overline{U}_0$ and Γ by applying CCs without timeliness comparison, as we did in Example 2, *e.g.,* since t_1[status] (resp. t_2[status]) is "single" (resp. "married") in Figure 1, $t_1 \preceq_{\text{status}} t_2$ is initialized in Γ by applying φ_2.

We have the following chase steps of D_t by (Σ, Γ):

(1) By applying (φ_4, h_4), where φ_4 is $t_1 \preceq_{\text{status}} t_2 \rightarrow t_1 \preceq_{\text{address}} t_2$ and h_4 maps the variables of φ_4 to tuples t_1 and t_2 in D_t, we deduce $t_1 \preceq_{\text{address}} t_2$ by the chase step $(\overline{U}_0, \preceq^{\Sigma}_0) \Rightarrow_{(\varphi_4,h_4)} (\overline{U}_1, \preceq^{\Sigma}_1)$, *i.e.,* $\overline{U}_1$ extends $\overline{U}_0$ by adding (t_1, t_2) to $\overline{U}_{\text{address}}$; similar to $\preceq^{\Sigma}_1$.

(2) The chase proceeds to deduce $t_1 \preceq_{\text{status}} t_4$ by applying φ_5.

This chasing sequence is valid since each chase step in the sequence is valid and no more CCs in Σ can be applied anymore. □

Church-Rosser property. Following [2], we say that chasing with CCs is *Church-Rosser* if for any temporal instance D_t, any set Σ of CCs, any collection Γ of ground truth, all chasing sequences of D_t by (Σ, Γ) are terminal and converge at the same result. Below we show that chasing with CCs is Church-Rosser (proven in [1]).

Corollary 2: *Chasing with* CCs *is Church-Rosser.* □

5.2 Deduction with the Chase

No matter how desirable, the chase could be expensive if we enumerate valuations of CCs in an exhaustive manner. Below we provide an efficient algorithm to implement the chase.

Challenges. A brute-force implementation of the chase is by enumerating valuations h of each CC φ in Σ. If (φ, h) can be applied, a chase step is performed, until the chasing sequence terminates. This method is, however, costly since valuation enumeration is inherently exponential. To tackle this challenge, we develop an efficient algorithm to implement the chase; the key idea is to only evoke valuations *pertaining* to the affected fixes in the chase *lazily* (see below).

We assume *w.l.o.g.* that for each $\varphi : X \rightarrow p_0$ in Σ, X has a predicate p in the form of $t_1 \preceq_A t_2$ (*e.g.,* φ_4). For those CCs that do not compare timeliness in their precondition (*e.g.,* φ_1), we apply them in a pre-processing step, to generate initial temporal orders in D_t.

Lazy evocation. To allow lazy evocation, the valuations of CCs in Σ are generated only when they are evoked by some newly deduced orders, instead of constructing all at the beginning of the chase.

Specifically, when a new temporal order o is deduced, we check each $\varphi : X \rightarrow p_0$ in Σ and evoke a new valuation h of φ if (a) o corresponds to a predicate in X (*i.e., o* is validated in h); in this case, we say that h is a valuation *pertaining* to o since h is "activated" by o, (b) h has not been evoked before and (c) the order that h deduces, *i.e.,* $h(p_0)$, is not deduced by other valuations before. We maintain designated data structures for checking conditions (a), (b) and (c) efficiently. See [1] for a detailed description of the data structures.

Algorithm. Putting these together, we present Chase in Figure 5. It returns new orders $\preceq^{\Sigma}$ if the chase is valid, and $\perp$ otherwise.

Chase starts with the initialization (Line 1). (a) Ground truth Γ is initialized with all stable ranked pair in D_t via procedure Initialize (omitted). (b) It initializes $\overline{U}$ and $\preceq^{\Sigma}$ as stated in Section 5.1 to be Γ and $\emptyset$, respectively. (c) The set Δ of newly validated orders is initialized to be the newly stable orders in Γ via procedure NewStable

(omitted); and they are the temporal orders "triggering" the chase. (d) The set $\mathcal{H}$ of evoked valuations is initialized to be empty.

Then for each order o in Δ, Chase does the following (Line 5-15): (a) Evoke the valuations pertaining to o via the lazy evocation strategy stated above, by calling CCEvoke (omitted), and add them to $\mathcal{H}$ (Line 5). (b) For each valuation h pertaining to o that deduces unknown ranked pair (checked in Line 7-8), mark o as validated in h (Line 9). If all predicates in the precondition of h are validated (Line 10), (h, φ) can be applied and the consequence $t_1 \preceq_A t_2$ of h is deduced (Line 11). The ranked pair (t_1, t_2) is added to $\overline{U}_A$ and $\preceq_A^{\Sigma}$ (Line 12). (c) Check conflicts (Line 13-14): if (t_1, t_2) conflicts with (t_2, t_1) that is already in $\overline{U}_A$ or $\preceq_A^{\mathcal{M}}$, the chase terminates with $\preceq^{\Sigma} = \perp$. In this case, the valuations in $\mathcal{H}$ are kept temporally so that they can be re-used in the next round of GATE when the conflicts are resolved (not shown). (d) Add the newly deduced order to the set Δ^{new} (Line 15) for iterative processing, by assigning Δ^{new} to Δ (Line 16).

Finally, the result of chasing, $\preceq^{\Sigma}$, is returned (Line 17).

Example 5: Recall that φ_4 is $t_1 \preceq_{\text{status}} t_2 \rightarrow t_1 \preceq_{\text{address}} t_2$ and φ_5 is $t_1 \preceq_{\text{status}} t_2 \wedge t_2 \preceq_{\text{status}} t_3 \rightarrow t_1 \preceq_{\text{status}} t_3$. Let $\Sigma = \{\varphi_4, \varphi_5\}$ and $\Delta = \{t_1 \preceq_{\text{status}} t_2\}$. We process $t_1 \preceq_{\text{status}} t_2$ in Δ as follows. It first evokes valuations h_4 and h_5 which map the variables of φ_4 and φ_5 to tuples t_1 and t_2 in D_t, respectively, with $t_1 \preceq_{\text{status}} t_2$ validated. Since the predicate $t_2 \preceq_{\text{status}} t_3$ in h_5 is not validated, h_5 is kept in $\mathcal{H}$ for later processing. In contrast, all predicates in the precondition of h_4 are validated and φ_4 deduces $t_1 \preceq_{\text{address}} t_2$. Suppose that there is no conflicting order in $\overline{U}_{\text{address}}$ and $\preceq_{\text{address}}^{\mathcal{M}}$. Then $t_1 \preceq_{\text{address}} t_2$ forms a new set Δ and the process continues, until Δ is empty. □

Complexity. The loop of Chase executes at most $O(|R||D|^2)$ times, since there are at most $|R||D|^2$ temporal orders to be deduced. For each temporal order o deduced, we evoke CCs based on o and update data structures in $O(c_{\text{val}}|\Sigma|)$ time, where c_{val} denotes the unit cost of constructing valuations for fixed o and φ, and there are at most $|\Sigma|$ CCs. Thus Chase takes at most $O(c_{\text{val}}|\Sigma||R||D|^2)$ time.

Augmented training data construction. Recall that the result of chasing, denoted by $\preceq^{\Sigma}$, is valid or invalid. Based on $\preceq^{\Sigma}$, we construct the augmented training data D_{aug} as follows.

(1) If $\preceq^{\Sigma}$ is valid, both the temporal orders deduced by the chase and predicted by the creator are used to create $D_{\text{aug}} = (D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}} = \text{true})$ where $\preceq'_{A_i} = \preceq_{A_i}^{\mathcal{M}} \cup \preceq_{A_i}^{\Sigma}$ $(i \in [1, n])$.

(2) If $\preceq^{\Sigma}$ is $\perp$, then there exist conflicting ranked pairs, *i.e.,* both (t_1, t_2) and (t_2, t_1) are in $\overline{U}_A$ or $\preceq_A^{\mathcal{M}}$ with $t_1 \prec_A t_2$ or $t_2 \prec_A t_1$. In this case, we construct D_{aug} to be $(D, \preceq'_{A_1}, \ldots, \preceq'_{A_n}, T, \mathsf{f}_{\text{valid}} = \text{false})$ where $\preceq_{A_i} = \{(t_1, t_2), (t_2, t_1)\}$ if $A_i = A$ and $\preceq_{A_i} = \emptyset$ otherwise.

As shown in Section 4, the creator fine-tunes its model by using the deduced ranked pairs or the detected conflicts in D_{aug}.

6 EXPERIMENTAL STUDY

Using real-life and synthetic data, we evaluated (1) the effectiveness and (2) the efficiency of GATE for determining temporal orders. We also (3) conducted a case study to showcase the usefulness of GATE.

Experimental settings. We start with the experimental setting.

Datasets. We used three real-life datasets and one synthetic dataset.

Input: A temporal instance D_t, the set Σ of CCs, the predicted $(\preceq_{A_1}^{\mathcal{M}}, \ldots, \preceq_{A_n}^{\mathcal{M}})$
Output: The result of chasing, $\preceq^{\Sigma}$.

1. Γ := Initialize(D_t); $\overline{U} := \Gamma$; $\preceq^{\Sigma} := \emptyset$; Δ = NewStable(Γ); $\mathcal{H} := \emptyset$;
2. **while** Δ is not empty **do**
3. $\quad\Delta^{\text{new}} = \emptyset$;
4. $\quad$**for each** $o \in \Delta$ **do**
5. $\quad\quad\mathcal{H} := \mathcal{H} \cup$ CCEvoke(D_t, Σ, o);
6. $\quad\quad$**for each** h of $\varphi : X \rightarrow t_1 \preceq_A t_2$ s.t. h pertains to o **do**
7. $\quad\quad\quad$**if** the order between $t_1[A]$ and $t_2[A]$ is already settled **then**
8. $\quad\quad\quad\quad\mathcal{H} := \mathcal{H} \setminus \{h\}$; **continue** ;
9. $\quad\quad\quad$Mark o as validated in h;
10. $\quad\quad\quad$**if** all predicates in the precondition of h are validated **then**
11. $\quad\quad\quad\quad\mathcal{H} := \mathcal{H} \setminus \{h\}$; /* $t_1 \preceq_A t_2$ is a newly deduced order */
12. $\quad\quad\quad\quad\overline{U}_A := \overline{U}_A \cup (t_1, t_2)$; $\preceq_A^{\Sigma} := \preceq_A^{\Sigma} \cup (t_1, t_2)$;
13. $\quad\quad\quad\quad$**if** $(t_2, t_1) \in \overline{U}_A$ or $(t_2, t_1) \in \preceq_A^{\mathcal{M}}$ **then** /* conflict */
14. $\quad\quad\quad\quad\quad\preceq^{\Sigma} := \perp$; **return** $\preceq^{\Sigma}$;
15. $\quad\quad\quad\quad\Delta^{\text{new}} := \Delta^{\text{new}} \cup \{t_1 \preceq_A t_2\}$;
16. $\quad\Delta := \Delta^{\text{new}}$;
17. **return** $\preceq^{\Sigma}$;

Figure 5: Procedure Chase

(1) Career [43], a benchmark about the careers of football players from FIFA-15 to FIFA-22; it contains 108.5K tuples from 27.2K entities with 20 attributes. We determine the timeliness of potential, position, reputation and league name. (2) NBA [19], a dataset that encompasses the careers of basketball players; it contains 10.6K tuples with 12 attributes. We determine the currency of team, pts (points) and weight. (3) COM [35], an open-source dataset about self-employed entrepreneurs in Shenzhen. After removing duplicated tuples and digital attributes (*e.g.,* the organization code), the dataset has 1,983,698 tuples and 8 attributes. We derive the timeliness of entrepreneur names. (4) Person, a synthetic dataset with 12.3K tuples from 1K entities. Just like Figure 1, we adopted 7 attributes and determine the currency of LN, Status, Kids. Here Person is generated by enforcing CCs (*e.g.,* φ_1-φ_6), to simulate real-world scenarios.

All the datasets have ground truth, *i.e.,* all tuples carry timestamps and are grouped by entities. The timestamps of COM are in seconds, while the others are in years; it is reasonable since, *e.g.,* it is uncommon for NBA players to frequently change teams in one year. We randomly selected 5% data with initial timestamps and masked the remaining. This default ratio ts% of initial timestamps is set intentionally small (so that the problem is more challenging); we will test the impact of ts% by varying ts% (Exp-1).

Currency constraints. We extended DCFinder [60] to discover CCs as discussed in Section 2. Note that CC discovery is conducted once on each dataset offline. Besides, we manually checked and adjusted the CCs discovered to ensure their correctness. We found 42, 32, 40 and 36 CCs for Career, NBA, COM and Person, respectively.

ML model. To learn the ranking model $\mathcal{M}_{\text{rank}}$, we used Bert [16] (distilbert) with 768 dimension to initialize the embeddings. We adopted 2 hidden layers in our encoders with sizes 200 and 100, respectively. The margin γ was adaptively computed and the model was trained with 30 epochs using Adam optimizer [40]. The learning rate is 1e-4. We used 5% data with timestamps as training data.

Baselines. GATE was implemented in Python and we compared it with the following baselines: (1) Creator, a variant of GATE with the creator only, *i.e.,* it predicts temporal orders using $\mathcal{M}_{\text{rank}}$; (2)

Critic, a variant of GATE with the critic only, *i.e.,* it deduces temporal orders by chasing with CCs; (3) $\text{Creator}_{\text{itr}}$, a variant of Creator that iteratively updates its training data with predicted but unjustified temporal orders. (4) $\text{Creator}_{\text{NC}}$, $\text{Creator}_{\text{NE}}$, $\text{Creator}_{\text{NA}}$, another three variants of Creator that implement $\mathcal{M}_{\text{rank}}$ without contextual information, without chronological encoding, and using regular cross entropy loss instead of adaptive margin-based loss, respectively; (5) GATE_{NC}, a variant of GATE that adopts the brute-force method for the chase, by enumerating all valuations exhaustively.

We also tested (6) UncertainRule [46], which uses uncertain currency rules to evaluate data currency; (7) Improve3C [19], a data quality framework that combines completeness, consistency and currency [29]; we only compare its accuracy for currency; (8) $\text{RANK}_{\text{Bert}}$ [58], a state-of-the-art ML ranking model based on Bert; and (9) $\text{Ditto}_{\text{Rank}}$, a ranking model that first trains a ditto model [49] to conduct binary classification on attribute values (with contextual information) and then sorts all attribute values using ditto as the comparison operator, *i.e.,* we used bubble-sort to sort the values, where the comparison between two values is conducted by ditto.

Among the baselines, (a) Critic, UncertainRule and Improve3C are rule-based, where rules for the latter two are converted from same CCs mined by DCFinder, (b) Creator, $\text{Creator}_{\text{itr}}$, $\text{Creator}_{\text{NA}}$, $\text{Creator}_{\text{NC}}$, $\text{Creator}_{\text{NE}}$, $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$ are ML-based, and (c) GATE_{NC} is a hybrid method, which produces same results as GATE. Thus, we compared GATE_{NC} mostly for efficiency.

We did the experiments on a single machine powered by 256GB RAM and 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz. We ran each experiment 3 times and report the average.

Experimental results. We next report our findings.

Exp-1: Effectiveness. Since we adopted the pairwise ranking setting to deduce ranked pairs, we evaluated the accuracy of GATE following [19, 28, 47]: (1) precision, the ratio of temporal orders determined correctly to all ranked pairs predicated true, (2) recall, the ratio of temporal orders predicted correctly to all true orders, and (3) F-measure $= 2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$. To evaluate the ranking of GATE, for each entity instance pertaining to entity e, we compute $\text{MRR}(e) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\text{rank}_i}$, the mean reciprocal rank over a set of n ranking results, where n is the number of currency attributes and rank_i denotes the rank of the latest value of the i-th attribute for entity e, and $\text{MAP@K}(e) = \frac{1}{n} \sum_{i=1}^{n} \text{AP@K}(i)$, the mean average precision at K that assesses whether the top-K values predicted are relevant and whether the latest K values are at the top, where $\text{AP@K}(i)$ is the average precision at K of the i-th attribute for entity e. Let D_t be a collection of k entity instances. We report (4) $\text{MRR} = \frac{1}{k} \sum_{j=1}^{k} \text{MRR}(e_j)$ and (5) $\text{MAP@K} = \frac{1}{k} \sum_{j=1}^{k} \text{MAP@K}(e_j)$, with K = 3 by default.

Rounds. We report the performance of GATE from the first round till its termination; at the end of the fixed round, we use current $\mathcal{M}_{\text{rank}}$ in the creator. As shown in Figures 6(a)-6(i), GATE takes 11, 12, 7 and 9 rounds to terminate on Career, NBA, COM and Person, respectively, *i.e.,* GATE converges quickly. Besides, we find the following.

(1) Although the performance of GATE might fluctuate (*e.g.,* Figure 6(h)), which is common in ML models [16], all metrics increase with more rounds in most cases, *e.g.,* F-measure, MAP and MRR increase from 0.767 to 0.866, 0.786 to 0.857, and 0.752 to 0.809, respectively, after 11 rounds on Career, verifying that GATE is able to deduce the latest values and produce good currency ranking. This is because the creator iteratively accumulates training data from the critic such that the model is better trained with more rounds; meanwhile, with better results predicted by the creator, the critic deduces more orders as augmented training data for the creator in subsequent rounds. Moreover, in Figures 6(b) and 6(c), precision and recall are 0.859 and 0.873, respectively, indicating that GATE achieves a good balance between the two and is fairly accurate. Note that $\text{Creator}_{\text{itr}}$ suffers from the accuracy fluctuation (*e.g.,* Figure 6(i)) since its model is affected by noisy (unjustified) temporal orders accumulated over rounds. The performance of other methods does not depend on rounds, as shown in flat lines. Since GATE behaves similarly under all metrics, below we focus on F-measure.

(2) On average GATE outperforms Creator and Critic by 7.8% and 43.8% in F-measure, respectively, up to 11.0% and 50.6%, improving both. The creator and critic benefit each other: (a) Creator produces "hidden" temporal orders for Critic to preform deduction, and (b) Critic deduces and justifies the orders, which are in turn provided as augmented training data to Creator; on average, the critic generates 5733 new training data (tuple pairs) per round on COM, improving F-measure of GATE from 0.701 to 0.748 after 5 rounds.

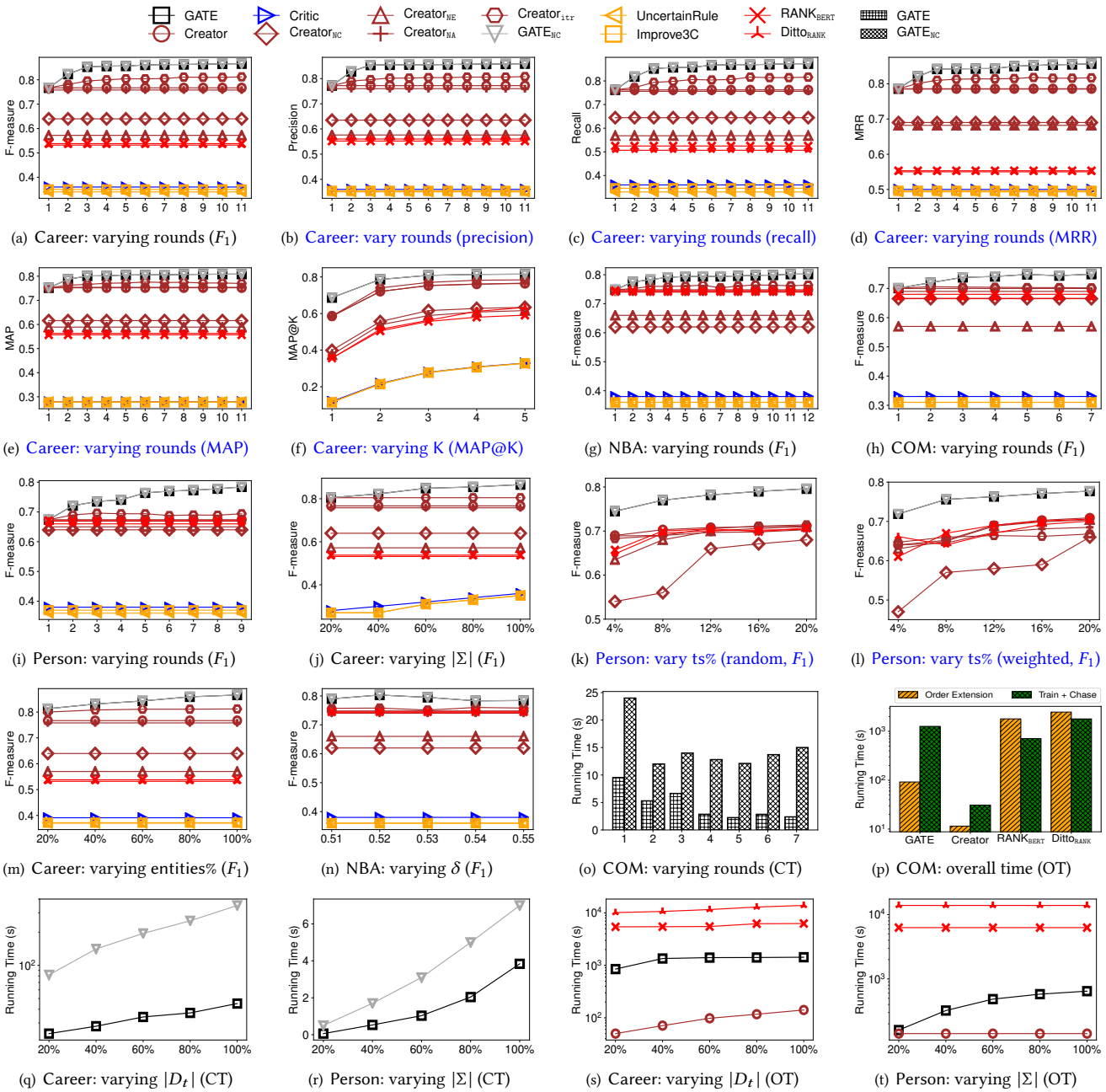
(3) Creator is more accurate than all its variants, *e.g.,* the average F-measure of Creator is 0.722, as opposed to 0.641, 0.613 and 0.714 by $\text{Creator}_{\text{NC}}$, $\text{Creator}_{\text{NE}}$ and $\text{Creator}_{\text{NA}}$, respectively, on Career. Intuitively, (a) without utilizing the contextual information, $\text{Creator}_{\text{NC}}$ cannot reference correlated attributes; (b) $\text{Creator}_{\text{NE}}$ has low accuracy with existing embedding models, and (c) compared to the regular cross entropy loss used in $\text{Creator}_{\text{NA}}$, the adaptive pairwise ranking loss helps by considering the semantics in ranking. Moreover, GATE is 10.1% more accurate than $\text{Creator}_{\text{itr}}$ on average. This verifies the need for justifying the temporal orders learned by Creator.

(4) The accuracy of GATE is higher than UncertainRule, Improve3C, $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$, *e.g.,* the average F-measure of GATE is 0.802 as opposed to 0.344, 0.349, 0.659 and 0.651 for the four, respectively. This shows the benefits of combining deep learning and logic rules: (a) compared with rule-based methods, GATE can learn from unseen data and has better generalizability; and (b) compared with ML-based methods, GATE is able to justify the reliability of deduction and produces more training data for the model.

Varying K. We varied parameter K of MAP@K from 1 to 5 in Figure 6(f). GATE consistently achieves the highest MAP@K, *e.g.,* 5% higher than the best baseline on average, up to 10.1%. This verifies that GATE ranks attribute values better than the baselines.

Varying $|\Sigma|$. Varying $|\Sigma|$, we evaluated the impact of the number of CCs in Figure 6(j). The accuracy of GATE, UncertainRule and Improve3C improves given more rules. For GATE, its F-measure changes from 0.805 to 0.866 when $|\Sigma|$ varies from 20% to 100%. Indeed, the critic deduces more orders with more CCs for the creator to fine-tune its model, to get a higher accuracy in an earlier stage.

Varying initial ts%. We varied the ratio ts% of initial timestamps (randomly selected) from 4% to 20%. More initial timestamps help since (a) the creator has more training data and can have better initial performance; and (b) the critic gets temporal orders as ground



(a) Career: varying rounds (F_1) (b) Career: vary rounds (precision) (c) Career: varying rounds (recall) (d) Career: varying rounds (MRR)

(e) Career: varying rounds (MAP) (f) Career: varying K (MAP@K) (g) NBA: varying rounds (F_1) (h) COM: varying rounds (F_1)

(i) Person: varying rounds (F_1) (j) Career: varying $|\Sigma|$ (F_1) (k) Person: vary ts% (random, F_1) (l) Person: vary ts% (weighted, F_1)

(m) Career: varying entities% (F_1) (n) NBA: varying δ (F_1) (o) COM: varying rounds (CT) (p) COM: overall time (OT)

(q) Career: varying $|D_t|$ (CT) (r) Person: varying $|\Sigma|$ (CT) (s) Career: varying $|D_t|$ (OT) (t) Person: varying $|\Sigma|$ (OT)

Figure 6: Performance evaluation

truth to perform deduction at the beginning of the chase. As shown in Figure 6(k), the F-measure of GATE increases (from 0.75 to 0.796 on Person) when ts% varies from 4% to 20%, as expected.

As remarked earlier, some attribute values may have more reliable timestamps than the others [42]. To study the impact of timestamp distributions, we assigned a weight to each attribute, where a larger weight indicates that the values of this attribute is more likely to be selected with initial timestamps, *e.g.*, on Person, values on Status are more likely to have timestamps than Kids. Consistent with Figure 6(k), Figure 6(l) shows that the accuracy under weighted sampling is also improved with larger ts%. Note that the accuracy under weighted sampling (Figure 6(l)) is slightly lower than random sampling (Figure 6(k)), *e.g.*, when ts%=20% on Person, its F-measure is 2.5% lower than random sampling. This is because weighted sampling is impacted by distribution discrepancy between training and testing data, which is a common out-of-distribution issue for ML.

Varying entities%. As reported in Figure 6(m), we varied the percentage of entities that are used for the chase from 20% to 100%. As expected, GATE improves its F-measure since with more entities, the critic can deduce more orders via the chase, and the creator can have more augmented data to train the model, achieving higher accuracy. For instance, F-measure of GATE is improved by 5.3% on average.

Varying δ. We next tested the impact of confidence threshold δ. As shown in Figure 6(n), (a) although when δ is small, less confident predictions may appear in subsequent deductions, more temporal

orders could be used for training; (b) when δ is too large, few ranked pairs learned are retained, and hence less augmented training data is returned. Since the creator receives less data to fine-tune its model, the accuracy may not be improved and it converges slowly. When $\delta = 0.52$, GATE has the highest accuracy on NBA. Thus, we set $\delta = 0.52$ as its default; for other datasets, δ is set similarly.

Exp-2: Efficiency. We tested the efficiency of GATE, GATE_{NC}, Creator, $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$. Denoted by CT (resp. OT) the chase time (resp. the overall time, for training, chasing and order extension). For GATE, OT is accumulated over all rounds; its order extension time is the time for extending stable orders to total orders by $\mathcal{M}_{\text{rank}}$ (Line 12 of Figure 3). For other baselines, the order extension time is the inference time of models for generating total orders. We did not report rule-based methods, which are fast, since they do not need to train models; but as shown in Exp-1, they are not accurate.

Chase time (rounds). We report CT of GATE and GATE_{NC} in the iterative process. As shown in Figure 6(o), GATE is substantially faster than GATE_{NC} for all rounds; it is 3.99X faster than GATE_{NC} on average, up to 6.30X on COM. The speedup of GATE is due to the lazy evocation strategy we adopted, which accelerates the chase by maintaining designated structures. In contrast, GATE_{NC} enumerates valuations and incurs redundant computation. Since GATE and GATE_{NC} produces the same results, their total rounds are the same.

Overall time. We next report OT in Figure 6(p). Although GATE has multiple rounds, its overall time is comparable to most ML methods, *e.g.,* GATE is 3.15X faster than $\text{Ditto}_{\text{Rank}}$ on average. In particular, the order extension time of GATE is smaller than most baselines except Creator, since GATE has to perform extra checks so that the total order generated is consistent with the known stable orders.

Varying $|D_t|$. We evaluated CT (resp. OT) of GATE and GATE_{NC} (resp. ML methods) by varying $|D_t|$ from 20% to 100% in Figure 6(q) (resp. 6(s)). With larger $|D_t|$, all methods take longer, as expected. Nonetheless, GATE is faster than GATE_{NC} when $|D_t|$ gets larger, since GATE maintains structures to avoid recomputation and does deduction pertaining to affected orders, *e.g.,* GATE is 4.80X faster than GATE_{NC} when $|D_t|$ is 100%; the result for OT is consistent.

Varying $|\Sigma|$. We varied $|\Sigma|$ from 20% to 100% on Person in Figure 6(r) and 6(t). As shown there, GATE is 3.76X faster than GATE_{NC} on average, up to 8.33X, which again verifies the effectiveness of lazy evocation. OT of GATE increases as $|\Sigma|$ is larger, since more training data (*i.e.,* temporal orders) is deduced by CCs to train the ranking model.

Exp-3: Case study. We use Career to illustrate why GATE works.

(1) Initial prediction. In the first round, GATE trains the creator on data with initial timestamps. Due to the limited (5%) training data, its F-measure is only 0.65 and some ranked pairs are mispredicted. One of confident and correct predictions is $t_1 \preceq_{\text{height}} t_2$, where t_1 and t_2 denote the same player with heights 186cm and 188cm, respectively. While this player moved from team PARMA to SPAL, *i.e.,* $t_1 \preceq_{\text{team}} t_2$, the creator makes a wrong prediction $t_2 \preceq_{\text{team}} t_1$.

(2) Critic *helps* Creator. After the creator stage, the critic uses CCs to correct mispredicted temporal orders. By applying $\varphi_7 : t_a \preceq_{\text{height}} t_b \rightarrow t_a \preceq_{\text{team}} t_b$ to the known $t_1 \preceq_{\text{height}} t_2$, it deduces $t_1 \preceq_{\text{team}} t_2$, correcting the mistake of Creator. Intuitively, φ_7 holds since $\preceq_{\text{height}}$ is monotonic, and $\preceq_{\text{height}}$ and $\preceq_{\text{team}}$ correlate for young players.

Moreover, Critic provides augmented training data to Creator. For instance, if $t_0 \preceq_{\text{league\_name}} t_1$ and $t_1 \preceq_{\text{potential}} t_2$ are in the ground truth, the critic could apply CC $\varphi_8 : t_a \preceq_{\text{league\_name}} t_b \wedge t_b \preceq_{\text{potential}} t_c \wedge t_a[\text{height}] \leq t_c[\text{height}] \rightarrow t_a \preceq_{\text{position}} t_c$, and deduces a new pair $t_0 \preceq_{\text{position}} t_2$ that is unknown before. Here φ_8 is learned from the data; intuitively, if a player moves to a new league (as indicated by monotonic $\preceq_{\text{height}}$) and if his potential changes, his position is likely adjusted, *e.g.,* from LM to CAM. After the first round, the critic creates 100,277 new ranked pairs as augmented training data, and the creator improves its model with the new data.

(3) Creator *helps* Critic. Critic cannot correctly deduce total orders from limited initial 5% timestamps. Nonetheless, with augmented training data provided by Critic, Creator can learn more ranked pairs with high confidence. On average it ranks 2843 tuple pairs with high confidence in the first 5 rounds. These ranked pairs are in turn provided to Critic, for Critic to deduce more new ranked pairs.

(4) The creator learns better with more data. ML models are inclined to get more accurate when given more training data. Creator continually receives more augmented training data and incrementally trains its model accordingly. As a consequence, its F-measure increases from 0.65 to 0.74 (resp. 0.81) after the first (resp. last) round.

Summary. We find the following. (1) Combining deep learning and logic rules makes a promising approach to deducing currency. GATE is the most accurate, *e.g.,* 0.866 in F-measure on Career, as opposed to 0.35 and 0.36 by rule-based UncertainRule and Improve3C, and 0.54 and 0.53 by ML-based $\text{RANK}_{\text{Bert}}$ and $\text{Ditto}_{\text{Rank}}$. (2) GATE only takes 7 rounds to terminate on COM, which has 1,983,698 tuples. (3) On average, GATE is 43.8% and 7.8% more accurate than Critic and Creator, respectively, *i.e.,* the creator and critic indeed benefit each other. (4) GATE beats GATE_{NC} in efficiency (with the same accuracy) by 9.62X on average, up to 19.37X, verifying the usefulness of lazy evocation strategies. (5) GATE has competitive overall time against existing ML methods, *e.g.,* 1359s on COM, as opposed to 2514s by the fastest of them. Although rule-based methods are fast (they do not train models), their accuracy are low. (6) Creator is more accurate than its variants $\text{Creator}_{\text{NC}}$, $\text{Creator}_{\text{NE}}$, $\text{Creator}_{\text{NA}}$ and $\text{Creator}_{\text{itr}}$ by 19.5%, 22.3%, 12.2% and 10.1%, respectively, verifying the need for context-aware embedding, chronological encoding, adaptive margin and order justification, respectively.

7 CONCLUSION

The novelty of the work consists of the following. (1) We formulate a new problem for determining the timeliness of attribute values. (2) As a solution to the problem, we propose a creator-critic framework by combining deep learning and logic deduction, for the two to enhance each other. (3) We develop a novel ranking model to learn temporal orders on attribute values. (4) We show how to justify the learned orders, deduce more ranked pairs and provide feedback for the learner, by extending the chase using CCs. The experimental study has verified that GATE is promising in practice.

One future topic is to study how to catch conflicts and missing values given temporal orders. Another topic is to extend CCs [31] by embedding ranking models as predicates, to improve the ranking accuracy with logic conditions and interpret ranking in logic.

REFERENCES

[1] 2022. Full version. https://github.com/yyssl88/Timeliness/blob/main/paper_full_version.pdf.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[3] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*.

[4] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient Denial Constraint Discovery with Hydra. *PVLDB* 11, 3 (2017), 311–323.

[5] Rpjc Jagadeesh Bose, Rs Ronny Mans, and Van Der Wmp Wil Aalst. 2013. Wanna improve process mining results? It's high time we consider data quality issues seriously. In *Computational Intelligence & Data Mining*.

[6] Philip Bramsen, Pawan Deshpande, Yoong Keok Lee, and Regina Barzilay. 2006. Inducing Temporal Graphs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL.

[7] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *international conference on Machine learning*. 89–96.

[8] Businesswire. 2022. Over 80 Percent of Companies Rely on Stale Data for Decision-Making. *https://www.businesswire.com/news/home/20220511005403/en/Over-80-Percent-of-Companies-Rely-on-Stale-Data-for-Decision-Making*.

[9] Statistics Canada. 2022. Classification of legal marital status. https://www23.statcan.gc.ca/imdb/p3VD.pl?Function=getVD&TVD=61748&CVD=61748&CLV=0&MLV=1&D=1.

[10] Nathanael Chambers and Dan Jurafsky. 2008. Jointly Combining Implicit Constraints Improves Temporal Ordering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Honolulu, Hawaii). ACL, 698–706.

[11] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*. PMLR, 1–24.

[12] Peter Christen and Ross W. Gayler. 2013. Adaptive Temporal Entity Resolution on Dynamic Databases. In *PAKDD*. Springer.

[13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[14] Kenneth Ward Church. 2017. Word2Vec. *Natural Language Engineering* 23, 1 (2017), 155–162.

[15] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *TODS* 4, 4 (1979), 397–434.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[17] Ioannis Dikeoulias, Saadullah Amin, and Günter Neumann. 2022. Temporal Knowledge Graph Reasoning with Low-rank and Model-agnostic Representations. *CoRR* abs/2204.04783 (2022).

[18] Xiaoou Ding, Hongzhi Wang, Yitong Gao, Jianzhong Li, and Hong Gao. 2017. Efficient currency determination algorithms for dynamic data. *Tsinghua Science and Technology* 22, 3 (2017), 227–242.

[19] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Jianzhong Li, and Hong Gao. 2018. Improve3c: Data cleaning on consistency and completeness with currency. *arXiv preprint arXiv:1808.00024* (2018).

[20] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2020. Leveraging Currency for Repairing Inconsistent and Incomplete Data. *TKDE* (2020).

[21] Aswathy Divakaran and Anuraj Mohan. 2020. Temporal Link Prediction: A Survey. *New Gener. Comput.* 38, 1 (2020), 213–258. https://doi.org/10.1007/s00354-019-00065-z

[22] X. Dong, L. Berti-Equille, and D. Srivastava. 2009. Truth Discovery and Copying Detection in a Dynamic World. *PVLDB* 2, 1 (2009), 562–573.

[23] X. L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava. 2010. Global detection of complex copying relationships between sources. In *PVLDB*.

[24] Xuliang Duan, Bing Guo, Yan Shen, Yuncheng Shen, Xiangqian Dong, and Hong Zhang. 2020. Research on Parallel Data Currency Rule Algorithms. In *International Conference on Information Science and System*. 24–28.

[25] Kevin K Duh. 2009. *Learning to rank with partially-labeled data*. University of Washington.

[26] Exasol. 2020. Exasol Research Finds 58% of Organizations Make Decisions Based on Outdated Data. *https://www.exasol.com/news-exasol-research-finds-organizations-make-decisions-based-on-outdated-data/*.

[27] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS* 33, 2 (2008), 6:1–6:48.

[28] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2013. Inferring data currency and consistency for conflict resolution. In *ICDE*. IEEE, 470–481.

[29] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *Journal Data and Information Quality (JDIQ)* 5, 1-2 (2014), 6:1–6:37.

[30] Wenfei Fan, Floris Geerts, and Jef Wijsen. 2011. Determining the currency of data. In *PODS*. ACM.

[31] Wenfei Fan, Floris Geerts, and Jef Wijsen. 2012. Determining the Currency of Data. *TODS* 37, 4 (2012), 25:1–25:46.

[32] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *PVLDB* 15, 9 (2022), 1861–1874.

[33] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[34] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.

[35] Shenzhen Municipal Govement. 2022. Self-employed Entrepreneurs. *https://opendata.sz.gov.cn/data/dataSet/toDataDetails/29200_01300931*.

[36] Tanya Goyal and Greg Durrett. 2019. Embedding Time Expressions for Deep Temporal Ordering Models. In *Conference of the Association for Computational Linguistics (ACL)*. ACL.

[37] Shuguang Han, Xuanhui Wang, Mike Bendersky, and Marc Najork. 2020. Learning-to-Rank with BERT in TF-Ranking. *arXiv preprint arXiv:2004.08476* (2020).

[38] Benjamin Hilprecht and Carsten Binnig. 2021. ReStore - Neural Data Completion for Relational Databases. In *SIGMOD*. 710–722.

[39] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *SIGKDD*. 368–377.

[40] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*.

[41] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2016. Overcoming catastrophic forgetting in neural networks. *CoRR* abs/1612.00796 (2016).

[42] Angelina Prima Kurniati, Eric Rojas, David Hogg, Geoff Hall, and Owen A Johnson. 2019. The assessment of data quality issues for process mining in healthcare using Medical Information Mart for Intensive Care III, a freely available e-health record database. *Health informatics journal* 25, 4 (2019), 1878–1893.

[43] Stefano Leone. 2022. FIFA 22 complete player dataset. https://www.kaggle.com/stefanoleone992/fifa-21-complete-player-dataset.

[44] Furong Li, Mong-Li Lee, Wynne Hsu, and Wang-Chiew Tan. 2015. Linking Temporal Records for Profiling Entities. In *SIGMOD*. ACM, 593–605.

[45] Mohan Li and Jianzhong Li. 2016. A minimized-rule based approach for improving data currency. *J. Comb. Optim.* (2016), 812–841.

[46] Mohan Li, Jianzhong Li, Siyao Cheng, and Yanbin Sun. 2018. Uncertain rule based method for determining data currency. *IEICE TRANSACTIONS on Information and Systems* 101, 10 (2018), 2447–2457.

[47] Mohan Li and Yanbin Sun. 2018. Currency Preserving Query: Selecting the Newest Values from Multiple Tables. *IEICE TRANSACTIONS on Information and Systems* 101, 12 (2018), 3059–3072.

[48] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. 2011. Linking Temporal Records. *PVLDB* 4, 11 (2011), 956–967.

[49] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.

[50] Yu Liang, Xuliang Duan, Yuanjun Ding, Xifeng Kou, and Jingcheng Huang. 2019. Data Mining of Students' Course Selection Based on Currency Rules and Decision Tree. In *International Conference on Big Data and Computing*. 247–252.

[51] Ashley Little. 2020. Outdated Data: Worse Than No Data? *https://info.aldensys.com/joint-use/outdated-data-is-worse-than-no-data#: :text=Obsolete%20data%20about%20the%20condition,too%20old%20to%20be%20reliable.*

[52] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (2009), 225–331. https://doi.org/10.1561/1500000016

[53] Tie-Yan Liu. 2010. Learning to rank for information retrieval. In *SIGIR*.

[54] Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. 2020. Approximate Denial Constraints. *PVLDB* 13, 10 (2020), 1682–1695.

[55] Niels Martin, Antonio Martinez-Millana, Bernardo Valdivieso, and Carlos Fernandez-Llatas. 2019. Interactive Data Cleaning for Process Mining: A Case Study of an Outpatient Clinic's Appointment System. In *International Conference on Business Process Management*.

[56] Qiang Ning, Zhili Feng, and Dan Roth. 2017. A Structured Learning Approach to Temporal Relation Extraction. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 1027–1037.

[57] Qiang Ning, Hao Wu, Haoruo Peng, and Dan Roth. 2018. Improving Temporal Relation Extraction with a Globally Acquired Statistical Resource. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*. ACL, 841–851.

[58] Rodrigo Frassetto Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *CoRR* abs/1901.04085 (2019).

[59] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *SIGKDD*. 2970–2978.

[60] Eduardo H. M. Pena, Eduardo Cunha de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *PVLDB* 13, 3 (2019), 266–278.
[61] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.
[62] Royal Mail. 2018. Dynamic Customer Data in a Digital World: Data Services Insight Report. *https://www.royalmail.com/business/system/files/royal-mail-data-services-insight-report-2018.pdf*.
[63] Ali Sadeghian, Mohammadreza Armandpour, Anthony Colas, and Daisy Zhe Wang. 2021. ChronoR: Rotation Based Temporal Knowledge Graph Embedding. In *AAAI*. AAAI Press, 6471–6479.
[64] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *SIGMOD*.
[65] Yi Tay, Minh C Phan, Luu Anh Tuan, and Siu Cheung Hui. 2017. Learning to rank question answer pairs with holographic dual lstm architecture. In *SIGIR*. 695–704.
[66] Julien Tourille, Olivier Ferret, Aurélie Névéol, and Xavier Tannier. 2017. Neural Architecture for Temporal Relation Extraction: A Bi-LSTM Approach for Detecting Narrative Containers. In *ACL*. ACL, 224–230.
[67] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. In *International Conference on Machine Learning (ICML)*, Vol. 70. PMLR, 3462–3471.
[68] Hongzhi Wang, Xiaoou Ding, Jianzhong Li, and Hong Gao. 2018. Rule-based entity resolution on database with hidden temporal information. *TKDE* 30, 11 (2018), 2199–2212.
[69] Jun Xu, Xiangnan He, and Hang Li. 2020. Deep Learning for Matching in Search and Recommendation. *Found. Trends Inf. Retr.* 14, 2-3 (2020), 102–288.
[70] Jing Yao, Zhicheng Dou, Jun Xu, and Ji-Rong Wen. 2021. RLPS: A Reinforcement Learning–Based Framework for Personalized Search. *TOIS* 39, 3 (2021), 1–29.
[71] Jingran Zhang, Fumin Shen, Xing Xu, and Heng Tao Shen. 2020. Temporal Reasoning Graph for Activity Recognition. *IEEE Trans. Image Process.* (2020).
[72] Meng Zhang, Yang Liu, Huanbo Luan, and Maosong Sun. 2016. Listwise ranking functions for statistical machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24, 8 (2016), 1464–1472.

APPENDIX A: NOTATION TABLE

The notations of the paper are summarized as follows.

Symbols	Notations
$R = (A_1, \ldots, A_n)$	relation schema
(D_e, T_e)	entity instance of schema R pertaining to e
temporal order $t_1 \preceq_A t_2$	t_2 is at least as current as t_1 in attribute A
$\mathsf{conf}(t_1 \preceq_A t_2)$	the confidence of $t_1 \preceq_A t_2$
$(D, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$	temporal instance of R
$\varphi = X \rightarrow p_0$	currency constraint
h	valuation of φ in a temporal instance
Γ	the ground truth
D_{aug}	the augmented training data
GATE	Get the lATEst

Table 1: Notations

APPENDIX B: DISCOVERY OF CCS

As noted in [31], CCs can be considered as a special case of denial constraints (DCs) introduced in [3], extended with temporal orders $\preceq_A$. Several algorithms have been developed for discovering DCs, *e.g.,* FastDC [13], Hydra [4], DCFinder [60] and ADCMiner [54].

We extend DCFinder [60] for discovering CCs as follows. (1) We support timeliness comparisons, *i.e.,* $t_1 \preceq_A t_2$, by adding them to the evidence set used in [60] or transforming categorical values to numerical ones using a mapping function $f_{\mathsf{map}}(\cdot)$, such that timeliness is preserved, *e.g.,* given $t_1 \preceq_A t_2$, we ensure $f_{\mathsf{map}}(t_1[A]) \leq f_{\mathsf{map}}(t_2[A])$. (2) We restrict our evidence set on tuples with the same EIDs, instead of using a global evidence set. This accelerates CCs discovery, since CCs are only defined on tuple variables that refer to the same entity. (3) We revise DCFinder by always selecting $t_1[\mathsf{EID}] = t_2[\mathsf{EID}]$ as the first predicate. Note that CCs could also be added manually, *e.g.,* to ensure the transitivity of the data.

APPENDIX C: MISSING PROOFS

Proof of Theorem 1. We prove Theorem 1 by showing that each temporal order $\preceq_A$ will be stable after certain rounds, since (a) once a temporal order $t_1 \preceq_A t_2$ is stable, *i.e.,* if (t_1, t_2) is added to $\preceq_A$ of D_t via procedure Extend (Line 5), it will not be removed from $\preceq_A$ in the subsequent rounds; and (b) the number of stable pairs in $\preceq_A$ is strictly increasing, which is upper-bounded by $O(|D|(|D|-1))$. □

Proof sketch of Corollary 2. CCs is a special case of entity enhancing rules (REEs) [33], extended with temporal orders $\preceq_A$. We prove Church-Rosser for CCs using a similar argument for REEs in [33], by showing (a) the length of any chasing sequence is bounded and (b) all chasing sequences converge at the same results. □

APPENDIX D: MODEL TRAINING

Fine-tuning based on conflicts. Assume *w.l.o.g.* that (t_1, t_2) is added to $\preceq_A$, since it has a much higher confidence score $\mathsf{conf}(t_1 \preceq_A t_2)$ than its conflicting order (t_2, t_1). We adopt a regularization term in the loss as follows, such that t_1 and t_2 can be better separated:

$$\mathsf{loss}_{\mathsf{reg}}(A) = \mathsf{loss}(A) - \lambda \mathsf{conf}(t_1 \preceq_A t_2),$$

where λ is a hyper-parameter, and the second term is the regularization term, penalizing the conflicting order to enlarge their margin. During the incremental training on augmented training data D_{aug} with $\mathsf{f}_{\mathsf{valid}} = \mathsf{false}$, we adopt $\mathsf{loss}_{\mathsf{reg}}(A)$ instead of $\mathsf{loss}(A)$.

APPENDIX E: MONOTONICITY

Monotonicity. We next show that the number of stable temporal orders in D_t is (strictly) monotonically increasing when more rounds GATE are performed (Lemma 3). The termination of GATE (Theorem 1) partly depends on this lemma. Lemma 3 directly follows from the way we expand stable temporal orders.

Lemma 3: *For temporal instances D_t^{j-1} and D_t^j in the j-th round of* GATE *before and after the extension, respectively,* i.e., $D_t^j = \mathsf{Extend}(D_t^{j-1}, D_{\mathsf{aug}})$*, the following holds:*

(a) $\forall i \in [1, n], \preceq_{A_i}^{j-1} \subseteq \preceq_{A_i}^{j}$ *and (b)* $\exists i^* \in [1, n], \preceq_{A_{i^*}}^{j-1} \subset \preceq_{A_{i^*}}^{j}$

where $\preceq_{A_i}^{j-1}$ and $\preceq_{A_i}^{j}$ are the orders in D_t^{j-1} and D_t^j, respectively. □

Proof. By the way we expand stable temporal orders, there are two cases: (1) If $\mathsf{f}_{\mathsf{valid}}$ is true, the result of chasing is valid. Then at least one non-empty $\preceq_A'$ of D_{aug} will be used to extend $\preceq_A$, whose size strictly increases. (2) If $\mathsf{f}_{\mathsf{valid}}$ is false, the result of chasing is invalid, *i.e.,* there is an attribute A such that $t_1[A] \neq t_2[A]$ but conflicting orders (t_1, t_2) and (t_2, t_1) are both in $\preceq_A'$ of D_{aug}. Either (t_1, t_2) or (t_2, t_1) is added to $\preceq_A$, resulting an increased size of $\preceq_A$. Once a temporal order $t_1 \preceq_A t_2$ is added to $\preceq_A$, it will not be removed. □

APPENDIX F: STRUCTURES AND STRATEGIES

To support lazy evocation of valuations, we employ the following:

(1) A set RHS of triples, where each triple (t_1, t_2, A) in RHS indicates that the order between $t_1[A]$ and $t_2[A]$ is not settled, *i.e.,* neither $t_1 \preceq_A t_2$ nor $t_2 \preceq_A t_1$ is stable in D_t yet. Note that we only apply a CC if its consequence has a corresponding triple in RHS. By ensuring this, the length of a chasing sequence is bounded by $O(|\mathsf{RHS}|)$.

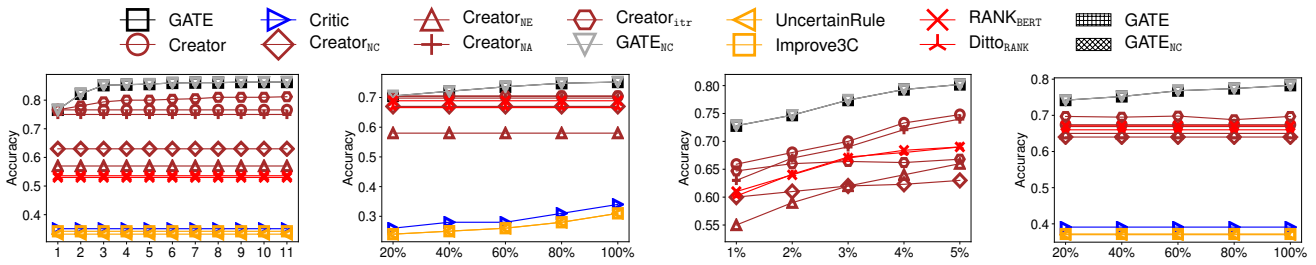
(2) A set $\mathcal{H}$ of *partial valuations*. Specifically, a valuation h of CC $\varphi : X \rightarrow p_0$ is said to be *partial* if some predicates in X are validated, while others are not. If all predicates in X are validated, h becomes *complete* and we can deduce temporal orders by applying (φ, h). The set $\mathcal{H}$ is maintained to avoid repeated predicate validation.

(3) An index $\mathcal{I}$ for the partial valuations in $\mathcal{H}$, *i.e.,* for each temporal order o, $\mathcal{I}[o]$ maintains the partial valuations h of $\varphi : X \rightarrow p_0$ in $\mathcal{H}$ such that there exists a predicate p in X and $o = h(p)$. By maintaining $\mathcal{I}$, every time a temporal order o is deduced, we can efficiently locate the valuations affected by o in $\mathcal{I}[o]$, without scanning the entire Σ. Besides, an inverted index is also built for each h so that once h is removed from $\mathcal{H}$, $\mathcal{I}$ can be updated efficiently.

(4) A set M of triples, where each triple (t_1, t_2, φ) indicates that the ranked pair (t_1, t_2) has been used to evoke the valuations for φ before and thus those valuations will not be evoked again.

Moreover, we adopt the following strategies.

(5) Lazy evocation, where valuations of CCs in Σ are constructed if they are evoked by some newly deduced orders, instead of being generated all at the beginning of the chase. Specifically, when a new temporal order o is deduced, we check each $\varphi : X \rightarrow p_0$ in Σ and evoke a new partial valuation h of φ if o corresponds to a predicate in X (*i.e., o* is validated in h) and h has not be evoked before (checked by M). Such h can only be evoked if the temporal order it deduces, *i.e.,*



(a) Career: varying rounds (Accu) (b) COM: varying |Σ| (Accu) (c) NBA: varying ts% (Accu) (d) Person: varying entities% (Accu)

Figure 7: More Performance evaluation

Input: A temporal instance D_t, the set Σ of CCs, the set $(\preceq^{\mathcal{M}}_{A_1}, \ldots, \preceq^{\mathcal{M}}_{A_n})$ predicted by the creator, the global structures GS = (RHS, $\mathcal{H}$, $\mathcal{I}$, M).
Output: The result of chasing, $\preceq^{\Sigma}$.

1. Γ := Initialize(D_t); $\overline{U}$:= Γ; $\preceq^{\Sigma}$:= $\emptyset$; Δ = NewStable(Γ);
2. **while** Δ is not empty **do**
3. $\Delta^{\text{new}} = \emptyset$;
4. **for each** $o \in \Delta$ **do**
5. $\mathcal{H}$:= $\mathcal{H} \cup$ CCEvoke(D_t, Σ, o, GS); Update $\mathcal{I}$ and M;
6. **for each** $h \in \mathcal{I}[o]$ where h deduces $t_1 \preceq_A t_2$ **do**
7. **if** $(t_1, t_2, A) \notin$ RHS **then**
8. Remove h from $\mathcal{H}$ and $\mathcal{I}$; **continue** ;
9. Mark o as validated in h;
10. **if** h is complete **then** /* $t_1 \preceq_A t_2$ is a newly deduced order */
11. Remove h from $\mathcal{H}$ and $\mathcal{I}$;
12. $\overline{U}_A := \overline{U}_A \cup (t_1, t_2)$; $\preceq^{\Sigma}_A$:=$\preceq^{\Sigma}_A \cup (t_1, t_2)$;
13. **if** $(t_2, t_1) \in \overline{U}_A$ or $(t_2, t_1) \in \preceq^{\mathcal{M}}_A$ **then** /* conflict */
14. $\preceq^{\Sigma}$:= $\perp$; **return** $\preceq^{\Sigma}$;
15. $\Delta^{\text{new}} := \Delta^{\text{new}} \cup \{t_1 \preceq_A t_2\}$;
16. $\Delta := \Delta^{\text{new}}$;
17. **return** $\preceq^{\Sigma}$;

Figure 8: Procedure Chase **(with data structures)**

$h(p_0)$, is not deduced by other valuations before (checked by RHS).

Algorithm. Putting these together, we present Chase in Figure 8 (a complete version of Figure 5, with data structures incorporated). It returns new orders $\preceq^{\Sigma}$ if the chase is valid, and $\perp$ otherwise.

Chase starts with the initialization (Line 1). (a) Ground truth Γ is initialized with all stable ranked pair in D_t via procedure Initialize (omitted). (b) $\overline{U}$ and $\preceq^{\Sigma}$ are initialized as stated in Section 5.1 to be Γ and $\emptyset$, respectively. (c) The set Δ of newly validated orders is initialized to be the newly stable orders in Γ via procedure NewStable (omitted), and they are the temporal orders "triggering" the chase.

Then for each order o in Δ, Chase does the following (Line 5-15): (a) Evoke the valuations h based on o via the lazy evocation strategy stated above, by calling CCEvoke (omitted), and add h to $\mathcal{H}$ (Line 5). (b) For each valuation h in $\mathcal{I}[o]$, mark o as validated in h (Line 9). If h becomes complete (Line 10), the consequence $t_1 \preceq_A t_2$ of h is deduced, and the ranked pair (t_1, t_2) is added to $\overline{U}_A$ and $\preceq^{\Sigma}_A$ (Line 12). (c) Check conflicts (Line 13-14): if (t_1, t_2) conflicts with (t_2, t_1) that is already in $\overline{U}_A$ or $\preceq^{\mathcal{M}}_A$, the chase terminates with $\preceq^{\Sigma}$= $\perp$. In this case, the partial valuations and the temporal orders that have been examined and deduced are kept temporally in the global structures so that they can be re-used in the next round of GATE when the conflicts are resolved (not shown). (d) Maintain the global structures in the three cases below: (i) if new valuation h is evoked by o (Line 5), $\mathcal{I}$ is updated accordingly and M is also updated such that h will not be evoked again; (ii) if h becomes useless, *i.e.,* the ranked pair it deduces is no longer in RHS (Line 7-8), h is removed from $\mathcal{H}$ and $\mathcal{I}$; and (iii) if h becomes complete, we deduce new orders, namely $t_1 \preceq_A t_2$, by applying h; then h is removed from $\mathcal{H}$ and $\mathcal{I}$ (Line 11). (e) Add the newly deduced order to the set Δ^{new} (Line 15) for iteratively processing, by assigning Δ^{new} to Δ (Line 16).

Finally, the result of chasing, $\preceq^{\Sigma}$, is returned (Line 17).

Complexity. The loop of Chase executes at most $O(|R||D|^2)$ times, since there are at most $|R||D|^2$ temporal orders to be deduced (*i.e.,* the length of any chasing sequence is $O(|R||D|^2)$). For each temporal order o deduced, we evoke CCs based on o and update the data structures in $O(c_{\text{val}}|\Sigma|)$ time, where c_{val} denotes the unit cost of constructing the valuations for fixed o and φ, and there are at most $|\Sigma|$ many CCs. Thus, Chase takes at most $O(c_{\text{val}}|\Sigma||R||D|^2)$ time.

APPENDIX G: MORE EXPERIMENTS

We show more experimental results in Figure 7, using the accuracy metric Accu, defined as the ratio of correct predictions of the relative rank between each value and the latest value, for each attribute.

Rounds. We report the accuracy of GATE from the first round till its termination on Career in Figure 7(a). We find: (a) the Accu of GATE increases with more rounds, *e.g.,* increases from 0.766 to 0.863 after 11 rounds, which verifies that GATE is capable to deduce the latest currency values; (b) the Accu of GATE is consistently higher than Creator, Critic and their variants, *e.g.,* it outperforms Creator and Critic by 9.7% and 51.2%, respectively; and (c) GATE is more accurate, in terms of Accu, than rule-based methods UncertainRule and Improve3C and ML-based models $\text{Ditto}_{\text{Rank}}$ and $\text{RANK}_{\text{Bert}}$, *e.g.,* the Accu of GATE is 22.7% higher than the best baseline. This verifies the benefit of combining deep learning and logical rules.

Varying |Σ|. We varied |Σ| from 20% to 100% on COM in Figure 7(b). As expected, the accuracy of GATE, Critic, UncertainRule and Improve3C increase when |Σ| increases, *e.g.,*Accu of GATE increases from 0.705 to 0.752 when |Σ| is from 20% to 100%. This show that more temporal orders can be correctly deducted given more rules.

Varying ts%. In Figure 7(c), we varied the ratio ts% of initial timestamps from 1% to 5% on NBA. As shown there, all methods tends to be more accuracy with larger ts%, since more temporal orders can be deduced based on tuples with initial timestamps and ML models are inclined to get more accurate when given more training data.

Varying entities%. We evaluated all methods by varying the percentage of entities used for chasing from 20% to 100% in Figure 7(d). GATE has higher accuracy with more entities, *e.g.,* its Accu changes from 0.742 to 0.783 when entities% is from 20% to 100%, since more training data can be produced by the critic, benefiting the creator.