Fairness-Aware Continuous Predictions of Multiple Analytics Targets in Dynamic Networks

Ruifeng Liu*
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
rliu@cs.uml.edu

Qu Liu*
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
qliu@cs.uml.edu

Tingjian Ge
University of Massachusetts, Lowell
Lowell, Massachusetts, USA
ge@cs.uml.edu

ABSTRACT

We study a novel problem of continuously predicting a number of user-subscribed continuous analytics targets (CATs) in dynamic networks. Our architecture includes any dynamic graph neural network model as the back end applied over the network data, and per CAT front end models that return results with their confidence to users. We devise a data filtering algorithm that feeds a provably optimal subset of data in the embedding space from back end model to front end models. Secondly, to ensure fairness in terms of query result accuracy for different CATs and users, we propose a fairness metric and a fairness-aware training scheduling algorithm, along with accuracy guarantees on fairness estimation. Our experiments over five real-world datasets show that our proposed solution is effective, efficient, fair, extensible, and adaptive.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**; **Online learning settings**.

KEYWORDS

dynamic networks, continuous analytics targets, representation learning, fairness

ACM Reference Format:
Ruifeng Liu, Qu Liu, and Tingjian Ge. 2023. Fairness-Aware Continuous Predictions of Multiple Analytics Targets in Dynamic Networks. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599341

1 INTRODUCTION

In dynamic networks, attribute values and/or links/nodes constantly change over time. As a powerful and general way to model data entities and their diverse interactions, dynamic networks are common in a large number of applications today. We list but a few examples: (1) messaging and social network interactions as graph edges, (2) clickstreams of users to products and purchase actions/edges from users to products in e-commerce, (3) user-product edges for reviews and feedback in e-commerce, (4) web requests and file transfers among parties on the Web such as Internet service providers and data centers, (5) cloud data systems such as cloud electronic health record (EHR) systems [13, 33, 41], and (6) real time traffic data where road segments are edges [17]. Importantly, the graph model has a flexible data schema, and hence has the inherent advantage of easier data fusion from multiple heterogeneous sources [1].

Example 1. *Figure 1 illustrates a dynamic network about patients in an intensive care unit (ICU) from a dataset [19]. There are many types of entities including patients, diagnoses, procedures, lab events, input events, and prescription drugs. They are shown as nodes of different colors. The edges represent the relations and interactions such as those between patients and diagnosis nodes, between patients and lab events, and so on. Interaction edges have timestamps and nodes may have attributes. Learning from such heterogeneous data sources is instrumental for inference and reasoning.*

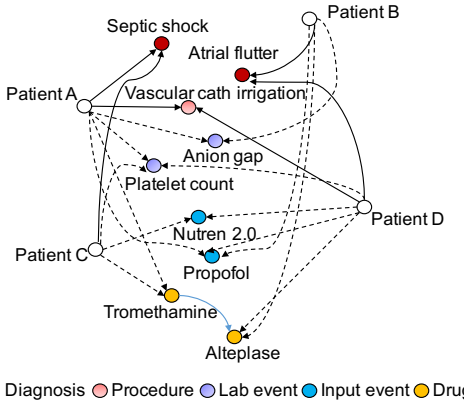

Figure 1: A dynamic network in the healthcare domain.

Continuous Analytics Targets (CATs). Dynamic networks resemble *data streams* [3] and *graph streams* [28], where data is also continuously changing, and a major type of analytics workload is user-subscribed monitoring tasks [4], which we call *continuous analytics targets* (CATs). In this paper, we focus on *predictive* targets, i.e., continuously predicting events/information at a later time. For instance, in Example 1, a hospital manager may subscribe to the following CAT:

> *Notify me when it is predicted that, in the next hour, grouped by the medical procedure, the number of patients tested with abnormal results is above a certain threshold.*

*These authors contributed equally to the paper.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599341

This is so that sufficient resources can be allocated in time to provide urgent care to those ICU patients. In addition, *one-time ad hoc tasks* may also be issued at any time.

Model and Challenges. Graph neural networks have become the state-of-the-art deep learning model for data represented as graphs [7, 22, 45]. In recent years, there has been a number of proposals of *dynamic graph neural networks* (DGNN) [21, 40] that become the dominant approach for *dynamic networks*. Thus, DGNN is the basic model for our problem.

Figure 2 shows the overall architecture. There is a *back end* (BE) model that is applied to the dynamic network; this is a typical DGNN model mentioned above, which typically consists of a conventional graph neural network (GNN) model to capture message-passing at the graph-structure level and a sequence model for temporal correlations. Then there is a *front end* (FE), which has a separate layer of neural network (e.g., MLP) for each continuous analytics target, and which presents the result of each CAT in the form of $(result_i, confidence_i)$ to the user.

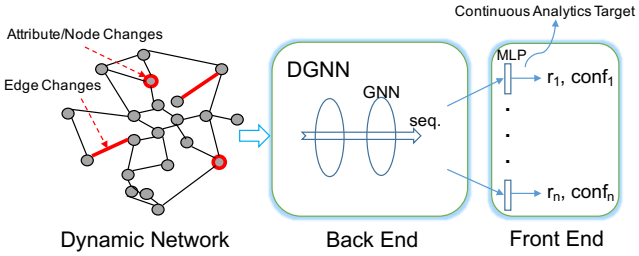

Figure 2: The overall model architecture.

As data constantly changes, the model needs to be continuously trained and updated [35]. The main challenges here have to do with the multiple continuous analytics targets (CATs). Each CAT is an analytical query as in the example above, and it is not clear the embeddings of which nodes of the dynamic network (from the DGNN at BE) should be fed as input to the model at the FE for that CAT. As there can be a large number of nodes in the network, simply aggregating the embeddings of all nodes may be too much and carry unneeded signals.

Secondly, since online continuous training is computational-resource demanding, as observed in our experiments (Section 5), it significantly affects the result accuracy and confidence of different CATs as to how much weight we give to each CAT or how to schedule their training. In summary, we aim to answer the following questions:

> *What is the best way to select latent messages/embeddings to route from the back end model to front end model for each CAT? In the online training of n CATs that may be from different users, how do we ensure a fair schedule/weight assignment of the n CATs?*

Our Contributions. To our knowledge, we are the first to work on the challenging online continuous training problem for multiple continuous analytical targets in dynamic networks.

For the problem of selecting latent messages/embeddings to route from the back end model to front end model for each CAT, we devise a randomized algorithm that adaptively over time determines an optimal subset of nodes in a tree resulted from a community detection algorithm. The embeddings of this subset of nodes will be fed into the front end model of a CAT. We prove that our randomized algorithm has a desired property that its selection of nodes follows a Markov chain and converges to a stationary distribution where the probability of the selection is proportional to its utility (defined as the accuracy of the CAT prediction). In addition, this selection is adaptive to data/pattern drifts.

For the problem of ensuring a fair schedule in training n CATs potentially from different users, we leverage the *individual fairness* [12] notion studied in machine learning and define the *Lipschitz bias* for our problem. Then we propose an algorithm that is fairness-aware and that adapts to the actual data while scheduling the training of n CATs. Importantly, we are able to estimate the Lipschitz bias and prove the theoretical guarantees of the estimation using *martingales* and *Azuma-Hoeffding inequality* [2]. We perform a comprehensive empirical evaluation using five real-world datasets that demonstrate the feasibility, scalability, accuracy, and extensibility of our approach.

2 PROBLEM STATEMENT AND PRELIMINARIES

A *dynamic network* $\mathcal{G} = (N, E)$ can be considered as an infinite sequence of *snapshots* $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_t, ...$, where each snapshot $\mathcal{G}_t$ corresponds to a *time step* t, N is the set of nodes, and E is the set of edges (either directed or undirected). Each snapshot $\mathcal{G}_t = (N_t, E_t)$ satisfies $N_t \subseteq N$ and $E_t \subseteq E$. Each node $v \in N$ may have a set of attributes $X = X_1, ..., X_a$ that may bear different values in different snapshots. In addition, each edge $e \in E$ may be of one of the r types.

$\mathcal{G}$ has n *continuous analytics targets* (CATs), each of which is to *continuously* predict, at every time t, a function of the data in $\mathcal{G}_{t+\delta}$, where $\delta > 0$. That is, each CAT is to keep predicting, at every step t, a value that is a function of the data in a future snapshot. In addition, for each prediction of a CAT, we also return the corresponding confidence value $0 < conf \leqslant 1$, which is the (estimated) probability that the returned result is correct [14, 37, 42].

For such a continuous analytics workload, there is a body of previous models called *dynamic graph neural networks* (DGNN) [21, 40] that can be used to perform the tasks. The overall model architecture is shown in Figure 2. The dynamic network contains constant attribute/node updates as well as edge updates. The back end (BE) model is a DGNN applied to this data, and is connected to the front end (FE), which contains one neural network model (e.g., multi-layer perceptron or MLP) per CAT. FE *continuously* returns to the users, for each CAT i ($1 \leqslant i \leqslant n$), the prediction result r_i along with the confidence $conf_i$.

Under this model architecture, in this paper, we focus on two related problems in the online continuous training for the n CATs: **(P1)** What is the best way (in particular, what node embeddings/latent messages) to select from the output/result of the back end DGNN model as the input to the FE model of each CAT? **(P2)** As the online training is resource demanding, and the n CATs may even be from different users, what is a fairness-aware way to train the CATs, and how to quantify the fairness?

3 SELECTION OF BACK END OUTPUT FOR A FRONT END CAT

In this section, we first study (P1) above. The back end DGNN model produces an embedding vector for each node in $\mathcal{G}$. For a particular continuous analytics target q_i, we aim to select an optimal set of nodes whose embeddings are fed into the FE model for q_i to produce the result with best accuracy/confidence. Intuitively, having too large a set of data items may be sub-optimal as some of them may be irrelevant and would weaken the useful signal to the prediction of q_i. In addition, some nodes/attributes may not be observed (i.e., missing) in the data, and some other nodes' embeddings may be better used for the prediction.

One intuition is that, for a cluster of nodes in $\mathcal{G}$, if we decide to select them to feed into FE, we can use their pooled (i.e., aggregate) embedding. This motivates us to first use a community detection algorithm (e.g., Leiden [43]) to compute a hierarchical tree $\mathcal{T}$ of communities offline, based on a static version $\mathcal{G}'$ of $\mathcal{G}$ from its edge statistics (e.g., an edge (u, v) exists in $\mathcal{G}'$ if the frequency of (u, v)'s appearance in $\mathcal{G}$ is above a threshold). Each internal node of $\mathcal{T}$ encompasses the cluster of nodes in the level below (i.e., its direct children); thus we use pooling to get the embedding of an internal node (e.g., average pooling). We assume that the optimal set we are seeking to feed to FE is from no more than δ nodes of $\mathcal{T}$, where δ is a small constant (which we empirically study in Section 5.2). We call this a *data filter* from BE to FE, as illustrated in Figure 3.

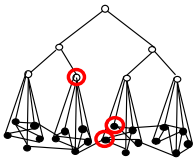

Figure 3: Illustrating a data filter from BE to FE. The solid vertices at the bottom level and the edges connecting them are from $\mathcal{G}$, which are partitioned into a hierarchical tree $\mathcal{T}$ of communities, as indicated by the hollow vertices at upper levels. The red-circled ones are the δ nodes ($\delta = 3$) selected to feed to a CAT's model in FE.

The basic idea of our filtering algorithm is to adaptively sample δ nodes from $\mathcal{T}$. Sequentially, each time we make a small local change to the sample and evaluate the *utility*, which can be flexibly defined, such as the accuracy or confidence of predicting the CAT. Based on the utility, we probabilistically either accept or reject the sample change. We will prove that this online adaptive algorithm has desired behavior. The algorithm is shown in FilterBE2FE.

In line 1, we first randomly choose δ leaf nodes from the embedding tree $\mathcal{T}$ based on the nodes referenced in q_i. This is a starting point of the node set S, with which we get the utility $u(S)$ in predicting q_i (line 2). The loop from line 3 iteratively adjusts the node set S until it follows a distribution that we desire. Specifically, we first choose a node v uniformly at random from S (line 4). Then with probability p, we perform a one-step random walk from v (line 5), in which v' is set to either a neighbor of v, each with probability $\frac{1}{d}$ (where d is the maximum degree of a node in $\mathcal{T}$) or v itself. With probability $1 - p$, we just choose v' uniformly at random from $\mathcal{T}$ (line 6)—which we call *teleport*.

In lines 7-8, the set S' is the tentative new version of S where node v is replaced by v', and we get its utility in predicting q_i. Line 9 probabilistically updates S to be S'—if S' has a higher utility, S is always updated; otherwise the update probability is the utility ratio. We now analyze FilterBE2FE and show that the execution follows a Markov chain with a desired stationary distribution.

Algorithm 1: FilterBE2FE

Input: $\mathcal{T}$: a tree of dynamic embeddings of $\mathcal{G}$ from BE
Output: δ nodes of $\mathcal{T}$ to feed to FE for CAT q_i

1 $S \leftarrow$ random δ leaves of $\mathcal{T}$ among those referenced in q_i
2 feed S to FE when evaluating q_i and get the utility $u(S)$
3 **while** *true* **do**
4 choose a node v uniformly at random from S
5 with probability p, $v' \leftarrow$ one step random walk from v
6 otherwise $v' \leftarrow$ a node chosen uniformly at random in $\mathcal{T}$
7 $S' \leftarrow S \setminus \{v\} \cup \{v'\}$
8 feed S' to FE when evaluating q_i and get the utility $u(S')$
9 with probability $min(1, \frac{u(S')}{u(S)})$, $S \leftarrow S'$
10 **return** S upon request as the current selection

Theorem 1. *The execution of FilterBE2FE follows a Markov chain that has a unique stationary distribution, in which the probability of choosing a set of δ nodes is proportional to the utility of the set.*

Proof. We define the *state* of the execution of FilterBE2FE as the set of nodes in S. It follows a Markov chain because the current content of S only depends on its content in the previous time step, i.e., the previous iteration of the loop in line 3. The chain has a *finite* number of states. It is *irreducible* [30] because any two states of the chain (i.e., two multisets of nodes S_1 and S_2) can reach each other through a number of steps (iterations of the loop from line 3). Moreover, the chain is *aperiodic* because at any state i with set S, there is a positive probability that the next step will be in the same state with the same node set (e.g., the teleport happens to select the same node v). Thus, this finite, irreducible, and aperiodic Markov chain must be an *ergodic* chain, and in turn, it must have a unique stationary distribution [30].

We next show that in its unique stationary distribution, the probability of having a set of δ nodes is proportional to the utility of the set. Precisely, we prove that the unique stationary distribution must be $\pi = (\frac{u_1}{Z}, \frac{u_2}{Z}, \ldots, \frac{u_m}{Z})$, where m is the number of states of the Markov chain, $u_i (1 \le i \le m)$ is the utility of state i, i.e., $u(S_i)$, and Z is a normalization constant $Z = \sum_{i=1}^{m} u_i$. Let π_i be $\frac{u_i}{Z}$, for $1 \le i \le m$. We show that

$$\pi_i P_{ij} = \pi_j P_{ji} \tag{1}$$

where P_{ij} is the transition probability from state i to state j following FilterBE2FE. When states i and j differ by more than one node, then clearly $P_{ij} = P_{ji} = 0$ according to FilterBE2FE, and Eq. 1 is true. When states i and j differ by only one node, there are two cases: (i) the nodes they differ on are neighbors in $\mathcal{T}$, or (ii) the nodes they differ on are not neighbors. Without loss of generality, we assume $u_i \le u_j$.

In case (i), according to FilterBE2FE (in particular line 9), we have $\pi_i P_{ij} = \pi_i p_n \min(1, \frac{u_j}{u_i}) = \pi_i p_n = \frac{u_i}{Z} p_n = \frac{u_j}{Z} p_n \cdot \frac{u_i}{u_j} = \pi_j P_{ji}$,

where $p_n = \frac{1}{\delta}(p\frac{1}{d} + q\frac{1}{|\mathcal{T}|})$ is the probability of picking these two nodes in case (i) for the proposed transition, $q = 1 - p$, and $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$. Thus Eq. 1 holds. Similarly, in case (ii), Eq. 1 holds too by only replacing p_n by $\frac{1}{\delta}q\frac{1}{|\mathcal{T}|}$. Then from Eq. 1, π must be the unique stationary distribution [31]. □

There are some additional details on the mapping between the chosen δ nodes and the positions of input to the FE model for a CAT, which are discussed in Appendix A.1.

4 FAIRNESS-AWARE TRAINING SCHEDULING

We now study the second problem (P2), i.e., how to schedule the training of the n continuous targets (CATs), which may be subscribed by different users, in a fair manner.

4.1 Fairness

We study the notion of fairness for the n CATs $q_1, \ldots, q_n$. *Bias* and *fairness* have been an issue of intense study in machine learning lately. There have been various proposals to define fairness for machine learning algorithms [29]. For our problem, the most relevant definition is the so-called "fairness through awareness" or "individual fairness" [12]. Intuitively, it requires that "similar individuals are treated similarly". In our context, informally, similar CATs should get results with similar accuracy. A CAT query may not get sufficient accuracy if it is not trained enough given the dynamic data updates. Here, we measure CAT similarity conveniently using their embeddings (to be discussed in Section 4.2). Formally, the fairness is defined as achieving the Lipschitz property:

Definition 1 ([12]). *A mapping $M : V \to Y$ satisfies the (D, d)-*Lipschitz property *if for every $x, y \in V$ we have*

$$D(Mx, My) \leq d(x, y)$$

Here, M maps an input (i.e., "individual" x or y) to an algorithm's output, and the outputs' distance as measured by D should not be greater than the difference of the individuals x and y as measured by d, conformant to the intuition of individual fairness as stated above. We extend this notion to a system that predicts n CATs $q_1, \ldots, q_n$, and define the bias as:

Definition 2. *The* Lipschitz bias *of predicting n CATs $q_1, \ldots, q_n$ is*

$$\frac{2}{n(n-1)} \sum_{1 \leq i < j \leq n} \max(0, |c_i - c_j| - d(q_i, q_j)) \tag{2}$$

where c_i (resp. c_j) is the confidence of the result of q_i (resp. q_j), and $d(q_i, q_j)$ is the distance between q_i and q_j in $[0, 1]$.

For example, $d(q_i, q_j)$ may be the cosine distance of their embeddings. Definition 2 performs the average over the $\binom{n}{2}$ pairs of CAT queries q_i and q_j. Each pair satisfies the Lipschitz property in Definition 1 if $|c_i - c_j| \leq d(q_i, q_j)$; otherwise, the difference is the bias of this CAT pair, and the average over all pairs is the Lipschitz bias of the n CAT queries.

4.2 CAT Language and Embedding

Recall that the Lipschitz bias in Definition 2 requires the distance $d(q_i, q_j)$ between two CAT queries q_i and q_j. In the meantime, it is also necessary to define a CAT query language for users. Analogous to work in data streams [3], it is common to use a variant of SQL as the user interface language. Moreover, a CAT query expressed in SQL corresponds to a *query tree* graph, for which we can perform graph embedding and treat it as the embedding of the CAT query. We use the *cosine distance* between the two CAT embeddings as $d(q_i, q_j)$ needed in Definition 2. The following CAT query **Q1** is similar to what is discussed earlier in Example 1 (Figure 1).

```
SELECT CASE WHEN COUNT(*)>0 THEN 'True' ELSE 'False' END
FROM (SELECT 1 FROM lab_events
      WHERE flag = 'abnormal' AND time BETWEEN t1 AND t2
      GROUP BY pid
      HAVING COUNT(*) > τ)
```

Q1 is to predict and notify the user if, in a future time interval $[t_1, t_2]$ (relative to *now*), there will be at least one patient in the *lab_events* subgraph/stream who has more than τ abnormal test results. A general query template is in Appendix A.2. Figure 4 illustrates a query tree template for CATs like Q1.

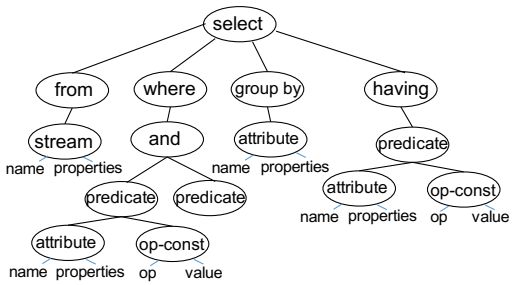

Figure 4: A query tree graph for a CAT used for embedding.

We then apply Graph Attention Networks (GAT) [45]:

$$\mathbf{h_i^{(l+1)}} = \sum_{j \in N(i)} \alpha_{ij} \mathbf{W^{(l)}} \mathbf{h_j^{(l)}} + \mathbf{B^{(l)}} \mathbf{h_i^{(l)}} \tag{3}$$

where $0 \leq l \leq L - 1$ and L is the level (#hops) of neighborhood message passing. α_{ij} is based on an attention mechanism. The boundary condition is $\mathbf{h_i^{(0)}} = \mathbf{x_i}$, the attribute values at vertex i, and $\mathbf{h_i^{(L)}}$ is the final embedding of vertex i.

We set L to be the height of the query tree, so that the root node will have latent information originated from all nodes in the tree. We use the embedding of the root (SELECT) node as the final query embedding. Thus, the root serves as the *anchor node* of the query graph. For a node that does not have any attributes (e.g., most of the internal nodes), we treat it as having one attribute with a fixed value 1. Moreover, we have multiple versions of $\mathbf{B^{(0)}}$, one for each node type (e.g., "select" or "stream" in Figure 4), i.e., $\mathbf{B_t^{(0)}}$ for a node of type t, which is a $d \times A$ matrix where A is the number of attributes of the node type and d is the embedding dimensionality. For a node without attributes, the $d \times 1$ $\mathbf{B_t^{(0)}}$ essentially represents the embedding vector of the node type itself.

CAT Embedding Training. We perform *self-supervised* training of the CAT query tree graph using node types. Specifically, given the current embedding $\mathbf{h_i^{(L)}}$ of node i, we use a cross-entropy loss function over $softmax(\mathbf{W_n h_i^{(L)}})$ to perform the node type classification, where $\mathbf{W_n}$ is a $T \times d$ parameter matrix and T is the number of node

types. All the parameter matrices are shared among the CATs. Thus, the embedding can easily generalize to a new CAT query using the learned parameters. Moreover, the similarity of different CATs can be gauged using the embedding vectors of their anchor nodes (i.e., roots), which will be used in fairness-aware training in Section 4.3.

4.3 Training Scheduling and Fairness Estimation

We devise a novel algorithm to train the whole network, both the FE and BE models, based on the notion of Lipschitz bias/fairness of the n CATs in Definition 2. In addition, we can efficiently estimate the Lipschitz bias of predicting the n CATs with provable guarantees.

4.3.1 The Algorithm. We are now ready to present the algorithm for training. The basic idea is to use the fairness notion as defined in Definition 2 as a guide to iteratively schedule the training of a pair of CATs each time. The algorithm will also attend to several training details such as logging data updates for delayed training, class imbalance, and learning confidence as discussed in Appendix A.3. We present the algorithm in FairnessAwareTraining.

Algorithm 2: FairnessAwareTraining

Input: a dynamic network $\mathcal{G}$
embeddings of CATs $q_1, \ldots, q_n$
Output: notifications whenever q_i returns *true*, and confidence; estimated Lipschitz bias

1 $\mathcal{D} \leftarrow$ sliding window of data updates in $\mathcal{G}$ for the last Δt_m
2 **while** *true* **do**
3 pick 2 pairs of CATs (q_i, q_j),(q_k, q_l) uniformly at random
4 predict these CATs using corresponding data in $\mathcal{D}$ and let the confidence be c_i, c_j, c_k, c_l
5 **for** $q \in \{q_i, q_j, q_k, q_l\}$ *that returns true* **do**
6 **return** notification with confidence
7 add q and its data context to $\mathcal{S}_+(q)$
8 **if** $\mid c_i - c_j \mid - d(q_i, q_j) < \mid c_k - c_l \mid - d(q_k, q_l)$ **then**
9 $i \leftarrow k; j \leftarrow l$ //(q_i, q_j) will be trained
10 $\mathcal{S}_w \leftarrow \mathcal{S}_w \cup \{q_i, q_j\}$
11 **while** *positive sample fraction in* $\mathcal{S}_w$ *is below* τ_+ **do**
12 randomly pick $q \in \mathcal{S}_w$
13 add random sample from $\mathcal{S}_+(q)$ to $\mathcal{S}_w$ //replay
14 do batch SGD using loss function detailed in Appendix A.3
15 report current Lipschitz bias based on Sec. 4.3.2

Line 1 of the algorithm maintains a sliding window of data updates within the past Δt_m time period, which will be used for training (details in Appendix A.3). The continuous training and CAT evaluation are in the loop in lines 2-15. We first randomly pick and evaluate two pairs of CATs in lines 3-4. If any of these 4 CATs returns *true*, we report it with the confidence (Appendix A.3), and add it to the positive instance set $\mathcal{S}_+(q)$ in lines 5-7. Lines 8-10 compare the two pairs of queries and the one with greater bias (Definition 2) is added to the sample set $\mathcal{S}_w$ to be trained. Lines 11-13 handles minority class (positive sample) replays for the class imbalance issue, adding positive instances of queries to $\mathcal{S}_w$.

Batched stochastic gradient descent (SGD) training is performed in line 14. The SGD training will revise the parameters in both the FE and BE models. Finally, we use the average bias of sample pairs in the current time window to continuously estimate the Lipschitz bias of the whole set of n CATs with provable guarantees (Theorem 2), as detailed next.

4.3.2 Estimating the Bias. Let us now analyze the actual *fairness* achieved by FairnessAwareTraining, which is quantified by measuring the Lipschitz bias—Equation 2 as discussed in Section 4.1. The less the Lipschitz bias, the more fairness we have. However, Equation 2 requires a quadratic number of pairs of CATs in a short time interval, as the bias is dynamic (with data) and sensitive to the measurement time window. Thus, we use a sample of pairs of CATs in a short, sliding time window to continuously estimate the current Lipschitz bias. The main challenge is how to analyze the sample size needed and the provided accuracy guarantees, since the biases of different query pairs are *not independent*. This renders the concentration inequalities such as Chernoff bound and Hoeffding inequality [31] inapplicable—it is harder to analyze when the variables are correlated. We perform a novel analysis using the theory of *martingales* and the associated bounded-difference inequality [2].

Preliminaries: Martingales and Concentration Inequalities. A *martingale* is a sequence of random variables for which, at a particular time, the conditional expectation of the next value in the sequence is equal to the present value, regardless of all prior values. More generally, a sequence $Z_0, Z_1, \ldots$ is a martingale w.r.t. the sequence $X_0, X_1, \ldots$ if $\mathbf{E}[Z_{n+1} \mid X_0, \ldots, X_n] = Z_n$. A simple example is a gambler who plays a sequence of fair games, where X_i is the amount she wins on the ith game (negative if a loss), and Z_i is her total winnings at the end of the ith game. A particular easy (and useful) construction of a martingale is called the *Doob martingale* [11], as follows. Let Y be a random variable that depends on $X_0, \ldots, X_n$. Then it can be shown that the sequence $Z_i = \mathbf{E}[Y \mid X_0, \ldots, X_i]$ is a martingale w.r.t. $X_0, \ldots, X_n$. As can be seen here, the notion of martingale may capture a sequence of *arbitrarily correlated* variables.

A particular form of concentration inequality under martingales is the *Azuma-Hoeffding inequality* [2], a general form of which is as follows. Let $Z_0, \ldots, Z_n$ be a martingale such that

$$B_k \leq Z_k - Z_{k-1} \leq B_k + d_k \tag{4}$$

for some constants d_k and random variables B_k that may be functions of $Z_0, \ldots, Z_{k-1}$. Then for all $t \geq 0$ and $\lambda > 0$,

$$Pr(|Z_t - Z_0| \geq \lambda) \leq 2e^{\frac{-2\lambda^2}{\sum_{k=1}^{t} d_k^2}} \tag{5}$$

Analysis of Lipschitz Bias Estimates. In our problem, we have a set of $m = \binom{n}{2}$ CAT pairs from which the Lipschitz bias is obtained (Definition 2). We use the biases of a sample of r CAT pairs to estimate the average bias of all pairs. What is challenging here is that the biases of the CAT pairs are obviously *not independent*. For example, if both (q_1, q_2) and (q_2, q_3) have small biases, then very likely (q_1, q_3) has a small bias too. We treat the m CAT-pairs as a sequence, defining a sequence of random variables $X_1, \ldots, X_m$ as the

biases calculated from each pair respectively (i.e., $\max(0, \left|c_i - c_j\right| - d(q_i, q_j))$).

Let the sample of r CAT-pairs' biases that are revealed to us as the first ones $X_1, \ldots, X_r$; i.e., we find that $X_i = b_i$ (a constant), for $1 \leq i \leq r$. Rename $X_{r+1}, \ldots, X_m$ to $Y_1, \ldots, Y_{m-r}$, and define $Z_i = \mathbf{E}[B \mid Y_0, \ldots, Y_i]$, for $0 \leq i \leq m-r$, where B is the random variable for the overall Lipschitz bias as in Definition 2. The Z_i sequence is the expected Lipschitz biases when the remaining bias variables $Y_1, \ldots, Y_i$ are revealed one by one. Thus, $Z_0, \ldots, Z_{m-r}$ is a Doob martingale w.r.t. $Y_0, \ldots, Y_{m-r}$, with

$$Z_0 = \frac{\sum_{i=1}^{r} b_i}{r}, \quad Z_{m-r} = B \tag{6}$$

as Z_0 is only based on the sample, and Z_{m-r} is the true Lipschitz bias when all Y_i's are known.

To bound the difference between Z_k and Z_{k-1} as in Inequality 4, we observe that $Z_k = \frac{S_{k-1}+b_{r+k}}{r+k}$, where $S_{k-1} = \sum_{i=1}^{r+k-1} b_i$. Since $0 \leq b_{r+k} \leq \frac{1}{2}$ (as any CAT satisfies $\frac{1}{2} \leq c_i \leq 1$ from Appendix A.3), we have $\frac{S_{k-1}}{r+k} \leq Z_k \leq \frac{S_{k-1}+0.5}{r+k}$, or equivalently, $\frac{S_{k-1}}{r+k} - Z_{k-1} \leq Z_k - Z_{k-1} \leq \frac{S_{k-1}+0.5}{r+k} - Z_{k-1}$. Letting B_k be $\frac{S_{k-1}}{r+k} - Z_{k-1}$, we get $d_k = \frac{0.5}{r+k}$ in Inequality 4. This, together with Equation 6 and Azuma-Hoeffding Inequality 5, gives us

$$Pr(\left|B - \frac{\sum_{i=1}^{r} b_i}{r}\right| \geq \lambda) \leq 2e^{\frac{-8\lambda^2}{\sum_{k=1}^{m-r} \frac{1}{(r+k)^2}}} \tag{7}$$

To simplify the above upper bound and get a closed form, we notice

$$\sum_{k=1}^{m-r} \frac{1}{(r+k)^2} \leq \sum_{k=1}^{m-r} \frac{1}{(r+k)(r+k-1)} = \sum_{k=1}^{m-r} (\frac{1}{r+k-1} - \frac{1}{r+k})$$
$$= \frac{1}{r} - \frac{1}{m}$$

Incorporating this into Inequality 7, we have proven the following:

Theorem 2. *Using the average bias of a sample of query-pairs of size r to estimate the Lipschitz bias of in-total $m = \binom{n}{2}$ query-pairs has the following accuracy guarantees: the probability that the error is greater than or equal to λ is at most $2e^{\frac{-8\lambda^2}{\frac{1}{r} - \frac{1}{m}}}$.*

Theorem 2 gives us strong theoretical guarantees on the accuracy of our estimation of fairness. For example, for $n = 50$ CATs (i.e., $m = 1225$), using a sample of only $r = 36$ CAT-pairs, we can achieve a Lipschitz bias upper bound of $\lambda = 0.1$ with probability at least 0.9.

5 EXPERIMENTAL EVALUATION

We have performed a systematic experimental evaluation using five real-world datasets. Through the experiments, We aim to answer the following research questions (RQ):

- **RQ1:** Regarding *scalability*, what is the throughput of processing high-rate data-update dynamic networks under our continuous training and predicting CATs?
- **RQ2:** How accurate is our model for predicting CATs?
- **RQ3:** Regarding *extensibility*, if we replace our BE model by different DGNN models, how does it function?
- **RQ4:** How are accuracy and throughput affected as we vary some model parameters?
- **RQ5:** How effective is our FilterBE2FE algorithm?
- **RQ6:** How effective is our fairness-aware training scheduling?

5.1 Datasets, Baselines, and CAT Queries

We use five datasets **MIMIC** [19], **GDELT** [25], **ICEWS18** [6], **ICEWS14** [44], and **WIKI** [23]. Their details and our setup are in Appendix A.4. Our default back end DGNN model is TGCN [48] but replacing its GRU sequence model by LSTM (resulting in slightly better accuracy). We compare against *five baselines*: (1) linkGS [49] for continuous link prediction (which is a restricted set of simple CATs) in graph streams, (2) predictive relation queries (RQ) in dynamic graphs [27], (3) our model's BE replaced by TransE [5], (4) our model's BE replaced by EvolveGCN [34], and (5) our model's BE replaced by ContinualGNN [46]. Note that none of the above is a whole baseline, as no previous work can replace our front end model as well as the FilterBE2FE and FairnessAwareTraining algorithms. The point is that we can plug in any DGNN model as the back-end model.

For each of the five datasets, we generate random continuous analytics targets based on the one in Section 4.2. In particular, for MIMIC data, the generated CATs are similar to Q1 in Section 4.2. For the GDELT dataset, the generated CATs select from country nodes (induced subgraph) with group-by attribute country ID. The ICEWS18 dataset is similar. For ICEWS14 and WIKI datasets, the CATs select from name nodes. The number of CATs that are generated for a dataset depends on the specific experiment, as detailed below.

5.2 RQ1: Throughput and Scalability

We perform the continuous training involving both BE and FE models. The FilterBE2FE has a parameter δ, the number of nodes fed to FE. The choice should be dynamic, as a larger network has significantly more leaves in $\mathcal{T}$, while the network size does not affect as much the height or internal-node number of $\mathcal{T}$. Thus, we impose a maximum number of internal nodes—empirically we find that a number between 3 and 5 gives a good tradeoff between training cost and accuracy; hence we use 4 by default.

First, for scalability, we examine the processing throughput under two modes: (1) the models are being trained continuously and the CAT queries are also being evaluated (i.e., predicted) continuously (shown as "training" lines in Figures 5 and 6); (2) only the CATs are evaluated continuously using a trained model from (1) (shown as "eval" lines in Figures 5 and 6). The throughput results are shown in Figure 5 for the MIMIC data and in Figure 6 for the GDELT data (the results for other datasets show a similar trend and are omitted).

We compare among several models: our *default* back end model, the relational query work ("RQ" in the figures) in [27], and replacing the back end of our model by EvolveGCN [34] ("EG" in the figures). Finally, we also compare against the lightweight link-prediction for graph streams work [49] ("linkGS" in the figures). We will further compare the query result accuracy with these baselines in RQ2 (Section 5.3) and RQ3 (Section 5.4). We vary the number of continuous CAT queries between 7 and 105. From Figures 5 and 6, we can see that mode 2 mentioned above ("eval" only) is much faster than mode 1 (continuous training). This is because the training involves repeated backpropagation through time while the CAT

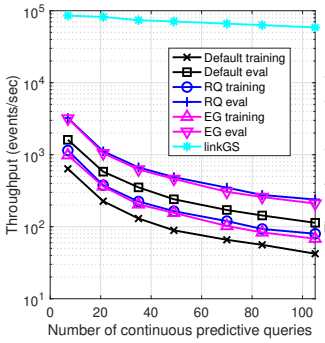

Fig 5 Throughput (MIMIC)

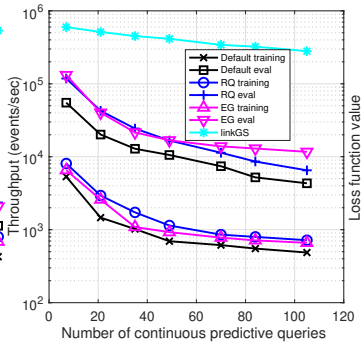

Fig 6 Throughput (GDELT)

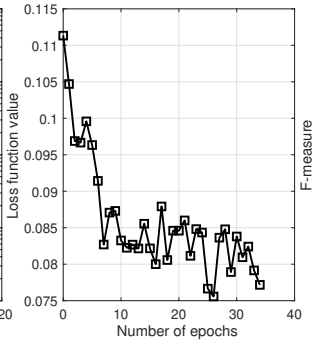

Fig 7 Loss function (ICEWS18)

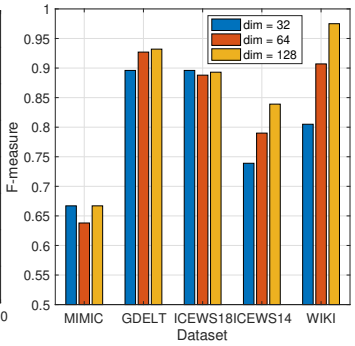

Fig 8 Accuracy vs. dimensionality

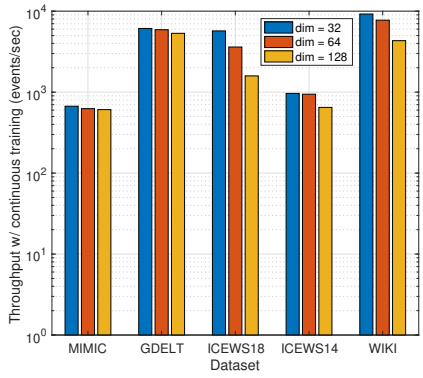

Fig 9 Throughput w/ continuous training

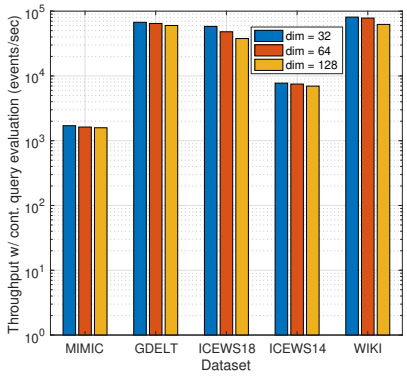

Fig 10 Throughput with eval only

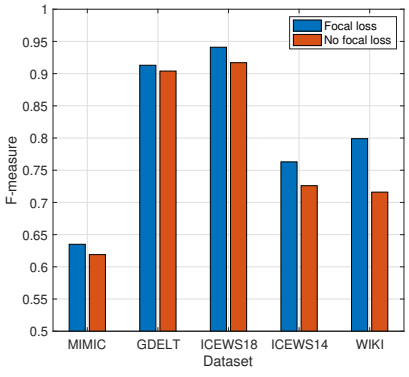

Fig 11 Accuracy w/ and w/o focal loss

evaluation only involves a forward pass of the model. As expected, more continuous CAT queries subscribed in the system entail a higher burden of continuous training and query evaluation, and hence a smaller throughput. Arguably, for most applications, such throughputs are able to keep up with the actual event rates (even with a single machine). The ContinualGNN baseline, however, is one to two orders of magnitude slower than our model in both modes; hence we omit it to avoid cluttering the figures. The RQ and EG baselines have higher throughput than using our default back end model. The linkGS method extends classical link prediction to graph streams without neural network training, and is 1-2 orders of magnitude faster. However, these methods are less accurate as shown in RQ2 and RQ3 next.

5.3 RQ2: Learning Embedding and Accuracy

For RQ2, we examine the effectiveness of our model training, and compare the CAT query result accuracy with two closest methods from previous work, relational query (RQ) and linkGS. Figure 7 shows that the loss function value over epochs for the ICEWS18 data (other datasets are similar). The loss drops precipitously in the first few epochs and then flattens. Observe that there are slight fluctuations of the loss between adjacent epochs because FairnessAwareTraining iteratively picks two pairs of CATs to train based on the biases, and the loss values differ among the CATs.

We then compare the CAT result accuracy with two closest approaches in previous work, namely linkGS [49] and RQ (continuous relational queries) [27]. Neither linkGS nor RQ can handle continuous analytics targets as complex as we do. To enable the comparison, we let all three methods run the same continuous link/relation predictive queries over all five datasets, with the accuracy results shown in Table 1, where we measure the precision, recall, and F-measure (shown as prec., rec., and F-m in Table 1, respectively). We can see that, even though linkGS has a much greater throughput as shown in Figures 5 and 6, our model is significantly more accurate than it (F-m is the overall accuracy). This is because linkGS is based on simple and efficient classical methods (it contains several methods, and we take the best accuracy among them), and is not a representation learning method. RQ has a slightly higher throughput, but our model is much more accurate.

5.4 RQ3: Extensibility

In answering RQ3, we examine the extensibility of our architecture by replacing the back end model by alternative representation learning methods of temporal networks. We use three such baseline models: an efficient incremental variant of knowledge graph embedding method TansE, as well as two recent ones EvolveGCN [34] and ContinualGNN [46]. The accuracy result is shown in Table 2.

From the results in Table 2 we can see that our default back end model is in general more accurate than the other three baseline back-end models, even though TansE and EvolveGCN give slightly higher throughput as shown in Figures 5 and 6 (TransE has a similar throughout to RQ which is a variant of it). TransE does not have an explicit sequence model as the other models and has the worst accuracy. Nevertheless, all alternative back-end models are

Table 1: Accuracy of continuous link/relation predictive queries compared to two methods in previous work.

	MIMIC			GDELT			ICEWS18			ICEWS14			WIKI		
	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.
linkGS	0.302	**0.901**	0.453	0.543	**0.914**	0.681	0.53	**0.878**	0.661	0.507	**0.99**	0.671	0.657	0.794	0.719
RQ	0.441	0.559	0.493	0.574	0.545	0.559	0.613	0.657	0.634	0.533	0.371	0.437	**0.737**	0.424	0.538
Ours	**0.734**	0.816	**0.773**	**0.598**	0.891	**0.716**	**0.658**	0.86	**0.746**	**0.567**	**0.99**	**0.721**	0.66	**0.859**	**0.746**

Table 2: Accuracy comparison with three alternative back end models.

	MIMIC			GDELT			ICEWS18			ICEWS14			WIKI		
	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.
Default	**0.644**	0.608	**0.625**	**0.876**	**0.969**	**0.92**	**0.992**	**0.815**	**0.895**	**0.705**	0.836	**0.765**	**0.845**	**0.772**	**0.807**
TransE	0.617	0.579	0.598	0.683	0.639	0.66	0.832	0.668	0.741	0.171	0.158	0.164	0.329	0.319	0.324
EvolveGCN	0.44	**0.694**	0.539	0.721	0.651	0.684	0.789	0.753	0.771	0.507	0.754	0.606	0.414	0.653	0.507
ContinualGNN	0.579	0.612	0.595	0.767	0.823	0.794	0.941	0.769	0.846	0.637	**0.841**	0.725	0.729	0.506	0.597

functioning to be able to answer CAT queries. This demonstrates the extensibility of our framework in that it can accommodate other temporal network representation learning models.

5.5 RQ4: Varying Model Parameters

We now move on to RQ4. We first look into the effect of changing the dimensionality of embedding vectors. We compare the cases when the embedding dimensionality is 32, 64, and 128, respectively, and examine the impact on accuracy when predicting CAT queries, the throughput when our model is being trained continuously while answering CAT queries, and the throughput when it is only answering CAT queries using a trained model.

Figure 8 shows the result on the average query accuracy (F-measure) for all five datasets. We can see that, for 3 out of the 5 datasets, namely GDELT, ICWS14, and WIKI, it is clear that the accuracy increases as the embedding dimensionality increases from 32 to 64 to 128. This is not the case with the MIMIC and ICEWS18 datasets. The reason may be that 32 dimensions already capture the needed latent features. Figure 9 shows the throughput with the five datasets when the system is under continuous training and CAT query evaluation, while Figure 10 shows the throughput when it is only continuously evaluating CAT queries with a trained model. In both cases, the throughput slightly decreases as the dimensionality increases. This is due to the greater computational overhead with a higher dimensionality.

We next examine the impact of the focal loss [26] (to improve accuracy with imbalanced classes, Appendix A.3) in our system. The results are shown in Figure 11 for CAT result accuracy and in Figure 12 for throughout under continuous training and query evaluation. Figure 11 shows that average CAT result accuracy improves when using focal loss for all five datasets, with ICEWS14 and WIKI being more significant. This may be due to more class imbalance with these two datasets. On the other hand, Figure 12 shows that focal loss in general has minimal impact on throughput. This is because the focal loss only adds a regularization term to the loss function and does not significantly affect the loss convergence.

5.6 RQ5: Effectiveness of FilterBE2FE

In RQ5, we look into the effectiveness of our FilterBE2FE algorithm, which continuously selects the data aggregation nodes based on the community structure to feed to a front end model. For comparison, without the FilterBE2FE optimization, naturally we just feed all the nodes in the back end network explicitly referenced/needed by the CAT queries to the front end model. We compare the average query accuracy in Table 3. It is clear that, overall, FilterBE2FE is very effective and considerably improves the accuracy over all five datasets. As discussed in Section 3, FilterBE2FE adjusts and learns the best data items to feed to the front end model in a data-driven manner, and hence performs better.

We also examine the impact of FilterBE2FE to throughput when the system is under continuous training and CAT evaluation (Figure 13) and CAT query evaluation only using a trained model (Figure 14). Figure 13 shows that there is a small overhead associated with FilterBE2FE, and Figure 14 shows that there is virtually no overhead to CAT query evaluation. This is because FilterBE2FE only makes small incremental changes at each continuous training step, while it does not make any changes at the forward inference pass for CAT evaluation.

5.7 RQ6: Impact of Fairness-Aware Scheduling

Finally, for RQ6, we investigate the effectiveness and overhead of FairnessAwareTraining. For comparison, without the fairness-aware training scheduling for training, a natural alternative is to do a *round-robin* scheduling of each CAT for training. Specifically, we examine the Lipschitz bias, the average CAT result accuracy, and throughput under continuous training with and without the fairness-aware scheduling in Figures 15, 16, and 17, respectively. From Figure 15, we can see that FairnessAwareTraining significantly reduces the Lipschitz bias, often by one to two orders of magnitude in four out of the five datasets, which achieves the main purpose of our fairness-aware scheduling.

Figure 16 shows that, as a side effect, FairnessAwareTraining also improves the average CAT result accuracy, quite significantly in some datasets such as MIMIC, ICEWS14, and WIKI. This is because, by allocating more training time to those "hard" (less accurate and

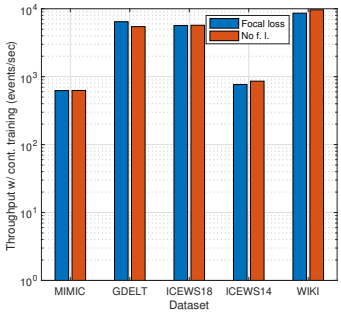

Fig 12 Throughput vs. focal loss

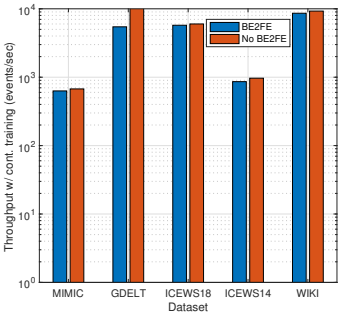

Fig 13 Throughput w/ training

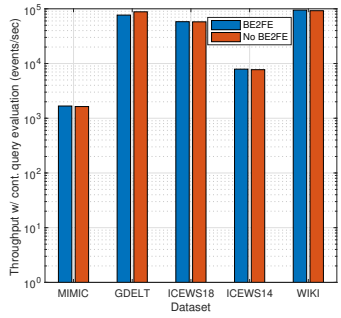

Fig 14 Throughput w/ query eval.

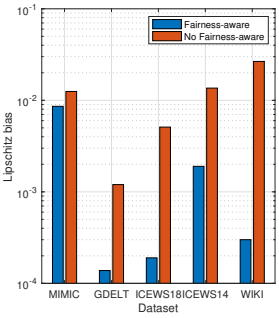

Fig 15 Bias vs. scheduling

Table 3: Accuracy comparison with and without BE2FE optimization.

	MIMIC			GDELT			ICEWS18			ICEWS14			WIKI		
	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.	prec.	rec.	F-m.
BE2FE	**0.812**	**0.631**	**0.71**	**0.896**	**0.956**	**0.925**	0.89	**1**	**0.942**	0.643	**0.999**	**0.782**	**0.714**	**0.798**	**0.754**
No BE2FE	0.586	0.567	0.576	0.863	0.927	0.894	**0.992**	0.832	0.905	**0.727**	0.709	0.718	0.597	0.705	0.647

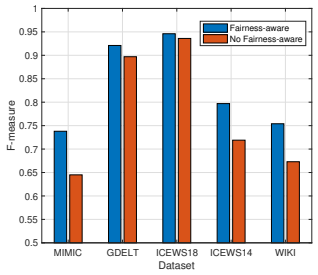

Fig 16 Accuracy comparison

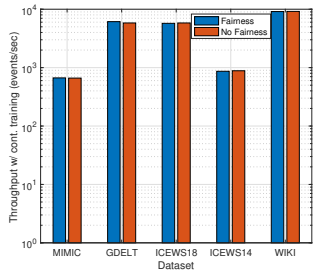

Fig 17 Throughput comparison

requiring more training) CATs, FairnessAwareTraining effectively improves the overall result accuracy. In addition, Figure 17 shows that FairnessAwareTraining has nearly the same throughput for continuous training and CAT evaluation as the round-robin scheduling. This is because there is little overhead in the extra logic of deciding which query to train next.

6 OTHER RELATED WORK

Most closely related work has been discussed in previous sections. We survey other related work here.

Online Learning. Online machine learning in general has a long history. It learns to update models from data streams sequentially [8, 15, 16]. The proposed algorithms include Perceptron [36], Online Gradient Descent [51], and Passive Aggressive [9], which are all for learning linear models. The work on online learning with kernels includes [38]. More recently, online learning has been studied for deep learning [24, 39, 50]. However, none of them studies online continuous learning in dynamic networks where a number of analytics tasks need to be continuously trained and predicted.

Dynamic Graph Neural Networks. There has recently been some work on DGNN. We refer the readers to two excellent surveys [21, 40]. As shown in our experiments in Section 5, our back end has great extensibility and can plug in any of these DGNN models, including [18, 27, 34, 46, 48]. We focus on the problems of how to filter data between the BE and FE models for specific CATs and how to schedule their training in a fair manner.

Multi-Task Learning and Fairness in Machine Learning. Our architecture contains multiple continuous analytics targets, which bears some resemblance to multi-task learning (MTL). MTL has been studied for deep neural networks [10], mostly under computer vision and NLP. However, none of the previous work solves the problems of determining the selection of back end shared model's embeddings for front end task specific models, as well as fairness-aware training of multiple tasks. Finally, *bias* and *fairness* have been an issue of intense study in machine learning lately. There have been various proposals to define fairness for machine learning algorithms [29]; yet we are the first to study the notion of fairness in scheduling multiple prediction targets during online training, especially when multiple users are in the system. Our experiments also indicate that fairness-aware scheduling improves overall prediction accuracy.

7 CONCLUSIONS

We study a novel problem of continuously predicting a number of continuous analytics targets (CATs) in dynamic networks. We focus on two problems. One is to adaptively determine the best set of network nodes whose embeddings from the back end DGNN should be passed to a front end model for predicting a CAT. The other is to devise a fairness-aware training scheduling algorithm and to estimate the fairness with accuracy guarantees. Our extensive experiments using five real-world datasets demonstrate the effectiveness, accuracy, scalability, and extensibility of our approach.

Acknowledgments. This work is supported by NSF grants IIS-2124704, OAC-2106740, and New England Transportation Consortium project 20-2.

REFERENCES

[1] J. Ahmed and M. Ahmed. 2018. Semantic web approach of integrating big data – A review. *International Journal of Computer Sciences and Engineering* (2018).

[2] N. Alon and J. Spencer. 1992. *The Probabilistic Method.* New York: Wiley.

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02)*. 1–16.

[4] Shivnath Babu and Jennifer Widom. 2001. Continuous queries over data streams. *ACM SIGMOD Record* 30, 3 (2001), 109–120.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc.

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

[7] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric Deep Learning: Grids, Groups, Graphs Geodesics and Gauges. *arXiv:2104.13478* (2021).

[8] N Cesa-Bianchi and G Lugosi. 2006. *Prediction, learning, and games*. Cambridge University Press.

[9] K Crammer, O Dekel, J Keshet, S Shalev-Shwartz, and Y Singer. 2006. Online passive-aggressive algorithms. *JMLR* (2006).

[10] Michael Crawshaw. 2020. Multi-Task Learning with Deep Neural Networks: A Survey. *arXiv preprint arXiv:2009.09796* (2020).

[11] Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press.

[12] Cynthia Dwork, Moritz Hardty, Toniann Pitassiz, Omer Reingold, and Richard Zemel. 2012. Fairness Through Awareness. In *The 3rd Innovations in Theoretical Computer Science*.

[13] R. S. Evans. 2016. Electronic Health Records: Then, Now, and in the Future. *IMIA Yearbook of Medical Informatics* Suppl 1 (2016).

[14] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*.

[15] S.C.H. Hoi, J Wang, and P Zhao. 2014. Libol: A library for online learning algorithms. *Journal of Machine* (2014).

[16] S. C. H. Hoi, D. Sahoo, J. Lu, and P. Zhao. 2018. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871* (2018).

[17] INRIX 2022. INRIX IQ. Available at https://inrix.com/products/iq-location-intelligence-solutions/..

[18] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. 2020. Recurrent Event Network: Autoregressive Structure Inference over Temporal Knowledge Graphs. In *EMNLP*.

[19] AEW Johnson, TJ Pollard, L Shen, L Lehman, M Feng, M Ghassemi, B Moody, P Szolovits, LA Celi, and RG Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific Data* (2016).

[20] Justin M. Johnson and Taghi M. Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 27 (2019).

[21] M. Seyed Kazemi. 2022. Dynamic Graph Neural Networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*, Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao (Eds.). Springer Singapore, Singapore, 323–349.

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *IEEE Transactions on Neural Networks* 5, 1 (2016), 61–80.

[23] Julien Leblay and Melisachew Wudage Chekol. 2018. Deriving validity time in knowledge graph. In *The Web Conference*.

[24] J Lee, J Yun, S Hwang, and E Yang. 2017. Lifelong learning with dynamically expandable networks. *arXiv:1708.01547* (2017).

[25] Kalev Leetaru and Philip A Schrodt. 2013. GDELT: Global data on events, location, and tone, 1979–2012. *ISA annual convention* 2 (2013), 1–49.

[26] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. 2017. Focal Loss for Dense Object Detection. In *ICCV*.

[27] Xuanming Liu and Tingjian Ge. 2020. Mining Dynamic Graph Streams for Predictive Queries Under Resource Constraints. In *PAKDD*.

[28] Andrew McGregor. 2014. Graph stream algorithms: A survey. *ACM SIGMOD Record* 43, 1 (2014), 9–20.

[29] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2022. A Survey on Bias and Fairness in Machine Learning. *Comput. Surveys* 54, 6 (2022), 1–35.

[30] S.P. Meyn and R.L. Tweedie. 2012. *Markov Chains and Stochastic Stability*. Springer London.

[31] M. Mitzenmacher and E.Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[32] Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. 2020. Confidence-Aware Learning for Deep Neural Networks. In *ICML*.

[33] Faisal Mushtaq and Matt E. Moore. 2020. Cloud-based interoperability will accelerate clinical outcomes. *Physicians Practice* (2020).

[34] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*.

[35] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71.

[36] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958).

[37] Tara Safavi, Danai Koutra, and Edgar Meij. 2020. Evaluating the Calibration of Knowledge Graph Embeddings for Trustworthy Link Prediction. In *The 2020 Conference on Empirical Methods in Natural Language Processing*.

[38] D Sahoo, S. C. H. Hoi, and P Zhao. 2016. Cost sensitive online multiple kernel classification. *ACML* (2016).

[39] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. 2018. Online Deep Learning: Learning Deep Neural Networks on the Fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2660–2666.

[40] J. Skarding, B. Gabrys, and K. Musial. 2021. Foundations and modelling of dynamic networks using Dynamic Graph Neural Networks: A survey. *IEEE Acces* 9 (2021).

[41] Naisha Sultana1, Gandikota Ramu, and B. Eswara Reddy. 2014. Cloud-based Development of Smart and Connected Data in Healthcare Application. *International Journal of Distributed and Parallel Systems* 5, 6 (2014).

[42] Pedro Tabacof and Luca Costabello. 2020. Probability Calibration for Knowledge Graph Embedding Models. In *The International Conference on Learning Representations (ICLR)*.

[43] V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports* 9, 5233 (2019).

[44] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*.

[45] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018).

[46] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming Graph Neural Networks via Continual Learning. In *CIKM*.

[47] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural Subgraph Matching. In *CoRR abs/2007.03092*.

[48] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems* 21, 9 (2020), 3848–3858.

[49] Peixiang Zhao, Charu Aggarwal, and Gewen He. 2016. Link Prediction in Graph Streams. In *ICDE*.

[50] G Zhou, K Sohn, and H Lee. 2012. Online incremental feature learning with denoising autoencoders. *AISTATS* (2012).

[51] M Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*.

A APPENDIX

A.1 Connection to FE

We now consider the mapping between the chosen δ nodes (let the set be $\mathcal{I}$) and the input positions to a model at FE for a CAT (e.g., MLP). The idea is that when FilterBE2FE slightly changes the membership of $\mathcal{I}$ to $\mathcal{I}'$ (e.g., they differ by only one node), the nodes $\mathcal{I} \cap \mathcal{I}'$ should not change their input positions at the FE model—this would make the training more stable (as the "wiring" of the neurons is mostly unchanged).

To achieve this goal, we randomly hash each node in $\mathcal{T}$ to one of the $\delta' > \delta$ input positions at the FE model. We use a parameter $\delta' > \delta$ so that the chance that a chosen node in $\mathcal{I}$ happens to collide with another node in $\mathcal{I}$ can be made arbitrarily small. In the event that more than one node maps to the same position, we can perform average pooling (or max pooling) between them.

Lemma 1. *The probability that a particular node in $\mathcal{I}$ does not collide with another node in $\mathcal{I}$ at the input of the front end model is $e^{-\frac{\delta-1}{\delta'}}$.*

Proof. The probability that a node v does not collide with any of the $\delta - 1$ nodes in $\mathcal{I} \setminus \{v\}$ is exactly $(1 - \frac{1}{\delta'})^{\delta-1} \approx e^{-\frac{\delta-1}{\delta'}}$. □

From Lemma 1, the probability of no collision increases (approaching 1) as δ' increases. We use *null* (all 0's) for an input position that does not correspond to any node in $\mathcal{I}$.

A.2 Query Template

In this paper, we consider a general template of CATs as

```
SELECT CASE WHEN COUNT(*)>0 THEN 'True' ELSE 'False' END
FROM (SELECT 1
      FROM <node_selector>
      [GROUP BY <attribute>]
      HAVING <condition>)
```

The CAT query returns "True" if at least one group satisfies the condition. The *node selector* selects a set of nodes and the subgraph induced by them. The GROUP BY and HAVING refer to node or edge attributes. This form is expressive enough to characterize a broad range of CATs. In dynamic networks, very often a query is targeted towards each substream that corresponds to an individual object or person. This is the reason for the optional GROUP BY clause above, while the HAVING clause specifies the condition within each group/substream (or a single group if there is no GROUP BY). Our representation-learning framework has potential to process an even broader range of CATs including reachability and subgraph pattern matching, which is beyond the scope of this paper and is left as future work (e.g., neural subgraph matching is studied in [47]).

A.3 Training Details and Loss Function

Logging Data Updates. A CAT is about the state of data in the dynamic graph $\mathbb{G}$ at time Δt from now. Thus, we need to delay the training for Δt because only then do we have the ground truth label to supervise the training. Consequently, for all CATs $q_1, \ldots, q_n$, we need to log the data updates in the last Δt_m, denoting the maximum Δt among all queries. In this way, we will be able to use the needed version of data to train over each query.

Class Imbalance. Sample class imbalance is a common problem in machine learning [20]. It is a potential issue in our context because the *selectivity* of a CAT query can be small, i.e., most of the time its result is *false*, and our training uses a small fraction of *true* samples. The vast number of "easy" negatives would overwhelm the training, causing it to under-perform for the minority positive class. To deal with this problem, we perform the following: **(1)** Log the data contexts of the instances of the minority class (typically the positive class). During training, with a certain frequency, we randomly pick one of the instances to "replay" the minority class backpropagation training. **(2)** Inspired by Lin et al.'s work on a computer vision problem [26], we replace the cross-entropy loss function of MLP QNet by the following focal loss function:

$$\mathcal{L}_f(p_i, y_i) = \begin{cases} -\alpha(1-p_i)^\gamma \log p_i, & (y_i = 1) \\ -(1-\alpha)p_i^\gamma \log(1-p_i), & (y_i = 0) \end{cases} \quad (8)$$

where $\gamma > 0$, and $0 < \alpha < 1$ is a weighting factor balancing the importance of positive/negative examples. We use $\gamma = 2$ by default. Intuitively, for example, for "easy negatives" (i.e., $y_i = 0$ and p_i is small), the p_i^γ factor makes the loss much smaller than the original cross-entropy loss. On the other hand, if this is a misclassification, e.g., $y_i = 1$ and p_i is close to 0, then the loss is close to the original cross-entropy loss. We set $\alpha = 1 - p_+$, where p_+ is the fraction of positive examples (after replays)—hence, the fewer positive examples we use, the greater weight their loss has.

Confidence. It is important for FE to also return an accurate confidence value c_i associated with the returned result of query q_i. One way to assign the confidence c_i is to use the maximum class probability at the output layer of an FE model. In Equation 8, this corresponds to

$$c_i = \max(p_i, 1 - p_i) \quad (9)$$

That is, if $p_i \geq 0.5$, the model will report *true* and $c_i = p_i$; otherwise it will report *false* and $c_i = 1 - p_i$. However, it is well known in the machine learning literature that the reported probability p_i is quite inaccurate and uncalibrated for deep neural networks [14, 37, 42].

Most of the existing approaches of probability calibration rely on post-processing, i.e., learning a model to adjust the maximum class probability after the original model has made the prediction. One problem with this approach, e.g., Isotonic regression or Platt scaling [14], is that the extra training of the post-processing model causes some overhead. One way is to use a regularization approach [32] that further enhances the focal loss function in Equation 8 with a confidence-order loss as follows:

$$\mathcal{L}_c(x_i, x_j) = \max(0, -sgn(f_i - f_j)(c_i - c_j) + |f_i - f_j|) \quad (10)$$

where $sgn(f_i - f_j)$ is the sign of $f_i - f_j$ (i.e., 1, 0, or -1), and f_i is the frequency that the QNet predicts a result that is the same as what it returns in the end during the SGD training of the sample of q_i. Intuitively, for an "easy" sample, during the iterations of SGD, the model already gives mostly consistent prediction answers, while for a "hard" sample, the model may go back and forth and take a longer time to get to the final returned answer. Thus, the relative order on frequency (f_i vs. f_j) should be consistent with the relative order on the returned confidence (c_i vs. c_j); otherwise, Equation 10 will penalize it with a loss. For instance, if $f_i > f_j$ but $c_i < c_j$,

then there is a positive loss $c_j - c_i + f_i - f_j$. Finally, the total loss function is a weighted sum of the focal loss $\mathcal{L}_f$ in Equation 8 and the confidence-order loss $\mathcal{L}_c$ in Equation 10:

$$\mathcal{L} = \sum_{(x_i, y_i) \in \mathcal{S}_w} \mathcal{L}_f(p_i, y_i) + \lambda \sum_{(x_i, x_j) \in \mathcal{SP}_w} \mathcal{L}_c(x_i, x_j) \qquad (11)$$

where λ is a weight constant, $\mathcal{S}_w$ is the set of samples from the current stream window, and $\mathcal{SP}_w$ is a set of sample pairs in the current window.

A.4 Details of Datasets and Setup

We use five real-world datasets as follows.

- **MIMIC**. The MIMIC-III (Medical Information Mart for Intensive Care III) dataset [19] is a large, freely-available database comprising deidentified health data associated with over 40,000 patients. It includes high temporal resolution data such as vital sign measurements, laboratory test results, procedures, medications, caregiver notes, imaging reports, demographics, etc. The total size is 27GB.
- **GDELT**. The GDELT (Global Database of Events, Language, and Tone) is an event-based social media dynamic graph [25]. It monitors print, broadcast, and web news media in over 100 languages from across every country to keep continually updated on breaking developments anywhere on the planet. It contains real-time measurements of 2,300 emotions and themes. The part that we use consists of 2,278,405 events with 240 temporal relationship types.
- **ICEWS18**. The ICEWS18 (Integrated Crisis Early Warning System in 2018) [6] contains social-media event data, which is coded interactions between socio-political actors (i.e., cooperative or hostile actions between individuals, groups, sectors and nation states). Events are automatically identified and extracted from news articles by the BBN ACCENT event coder. The part that we use contains 468,558 dynamic graph events in 256 relationship types.
- **ICEWS14**. The ICEWS14 data is similar to ICEWS18, but the data was collected in 2014 [44]. These events are triples consisting of a source actor, an event type (according to the CAMEO taxonomy of events), and a target actor. The part of the dataset that we use contains 665,304 temporal events in 260 relationship types.
- **WIKI**. Wikidata is a free and open knowledge base that can be read and edited by both humans and machines [23]. It has structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others. The portion of data that we use consists of 669,934 temporal events in 24 relationship types.

All the algorithms described in this paper are implemented in Python. All the experiments are run on a machine with Intel i7-8750H CPU and GeForce RTX 2080 GPU.