DRAGON: Dynamic Recurrent Accelerator for Graph Online Convolution

JOSÉ ROMERO HUNG, CHAO LI, TAOLEI WANG, JINYANG GUO, PENGYU WANG, CHUANMING SHAO, JING WANG, and GUOYONG SHI, Shanghai Jiao Tong University, China
XIANGWEN LIU and HANQING WU, Alibaba Cloud Computing Ltd., China

Despite the extraordinary applicative potentiality that dynamic graph inference may entail, its practical-physical implementation has been a topic seldom explored in literature. Although graph inference through neural networks has received plenty of algorithmic innovation, its transfer to the physical world has not found similar development. This is understandable since the most preeminent Euclidean acceleration techniques from CNN have little implication in the non-Euclidean nature of relational graphs. Instead of coping with the challenges arising from forcing naturally sparse structures into more inflexible stochastic arrangements, in DRAGON, we embrace this characteristic in order to promote acceleration. Inspired by high-performance computing approaches like Parallel Multi-moth Flame Optimization for Link Prediction (PMFO-LP), we propose and implement a novel efficient architecture, capable of producing similar speed-up and performance than baseline but at a fraction of its hardware requirements and power consumption. We leverage the hidden parallelistic capacity of our previously developed static graph convolutional processor ACE-GCN and expanded it with RNN structures, allowing the deployment of a multi-processing network referenced around a common pool of proximity-based centroids. Experimental results demonstrate outstanding acceleration. In comparison with the fastest CPU-based software implementation available in the literature, DRAGON has achieved roughly 191× speed-up. Under the largest configuration and dataset, DRAGON was also able to overtake a more power-hungry PMFO-LP by almost 1.59× in speed, and at around 89.59% in power efficiency. More importantly than raw acceleration, we demonstrate the unique functional qualities of our approach as a flexible and fault-tolerant solution that makes it an interesting alternative for an anthology of applicative scenarios.

CCS Concepts: • **Computer systems organization** → **Real-time system architecture**; *Embedded hardware*; • **Hardware** → *Hardware accelerators*; • **Computing methodologies** → Neural networks;

Additional Key Words and Phrases: Convolutional neural networks, HW accelerator, embedded systems, dynamic graphs

ACM Reference format:
José Romero Hung, Chao Li, Taolei Wang, Jinyang Guo, Pengyu Wang, Chuanming Shao, Jing Wang, Guoyong Shi, Xiangwen Liu, and Hanqing Wu. 2023. DRAGON: Dynamic Recurrent Accelerator for Graph Online Convolution. *ACM Trans. Des. Autom. Electron. Syst.* 28, 1, Article 1 (January 2023), 27 pages.
https://doi.org/10.1145/3524124

This work is supported in part by the National Natural Science Foundation of China (No. 61972247), Shanghai S&T Committee Rising-Star Program (No. 21QA1404400), and Alibaba Innovative Research (AIR) Program.

Authors' addresses: J. Romero Hung, C. Li (corresponding author), T. Wang, J. Guo, P. Wang, C. Shao, J. Wang, and G. Shi, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai 2000240, China; emails: romerohung@sjtu.edu.cn, lichao@cs.sjtu.edu.cn, {sjtuwtl, lazarus, wpybtw, cyunming, jing618, shiguoyong}@sjtu.edu.cn; X. Liu and H. Wu, Alibaba Cloud Computing Ltd., 969 Wenyixi Road, Hangzhou, China; emails: {vicki.liuxw, hanqin.wuhq}@alibaba-inc.com.

1084-4309/2023/01-ART1 $15.00
https://doi.org/10.1145/3524124

1 INTRODUCTION

Graph neural networks (**GNN**) represent the most recent efforts within the AI community to leverage the modeling and computing capabilities of graph representation. Just as in its Euclidean counterpart at CNN, the spectral-based variance of GNN, i.e., **graph convolutional networks** (**GCN**) introduces convolutional aggregators to produce accurate structural synthesis. This facilitates important graph analysis tasks such as link prediction and node classification.

Because many real-world phenomena have been found better described with dynamic graphs, i.e., when the graph elements change in time, the temporal perspective of GNN and its vast applicability has deserved considerable interest [44, 45, 50]. Better yet, a plethora of potential situations would find special utility in handling ultra-fast dynamic graph inference. For instance, the automated financial field and high-frequency trading [5, 27, 35], robot piloting and lidar navigation [1, 36], telecommunications network workload [10, 61], weather tracking and prediction [39, 47], smart grids[21, 48], and many other public services autoregulation in future smart cities.

The aforementioned examples have in common a high dependency on a constant, reliable, and massive data provision of relational information, which is, in turn, generated from some sort of real-time sensorial network. In every case, operations must be registered, responded to, and broadcasted at a very high ratio, generally at the levels of nanoseconds. A fitting hardware platform must be in pair to these speeds, ubiquitous, and in-place processing requirements.

Although some of the static graph accelerators proposed to date [12, 32, 60] could be seemingly adapted for the dynamic case, their direct regularity-exploiting approach would not allow realistic implementations without incurring exorbitant resource requirements. It is necessary to explore different alternatives for efficiently addressing real-time GCN implementation on hardware.

Based on prior structural proximity studies for minimizing communication and computing overload like [2, 24, 30], we have previously developed a hardware acceleration approach for static graph embedding named ACE-GCN [23]. Due to its specific resource-efficient and data-driven characteristics, ACE-GCN is considered an attractive option for a potential expansion into the dynamic embedded context. Howbeit its good results, the unipolar processing foundation of ACE-GCN lacks the concurrence virtues of regular multi-processing implementation. This situation unavoidably leads to considerable centroid stabilization periods that may affect response capacity to sudden dataset changes and prevent it from fully exploiting its accelerative potentialities.

In this article, we propose a **Dynamic Recurrent Accelerator for Graph Online (DRAGON).** With DRAGON, we unify structural, and temporal graph embedding into a single framework and provide resource-efficient acceleration for graph link prediction. Moreover, due to its intended low-power real-time segment, our accelerator includes unique data-driven and fault-tolerant qualities. The key acceleration premise behind DRAGON derives from the introduction of a befitting multi-processing strategy that we have named "**multi-polar centroid contribution**" based on the well-established moth-flame optimization paradigm [40].

Firstly, we transfer the concept of "implicit-processing-by-association" from the static graph embedding case of ACE-GCN [23] and propose a new structural-temporal integrated processing core aimed at the dynamic graph acceleration requirements. Then, following the multi-processing oscillative approach of PMFO-LP [3], we enhance the former and deploy an enclosed network of processing elements that mutually optimize processing paths toward prediction by following a common proximity-based reference and auto-completion platform.

To the best of our knowledge, DRAGON is the first resource-conservative hardware accelerator specifically designed for dynamic graph link prediction. Our experiments demonstrate that DRAGON offers excellent acceleration to task completion on several real-world dynamic datasets in comparison with the state-of-the-art multi-purpose CPU-based execution.

Table 1. Principal Acronyms used in this Work

Acronym	Meaning	Acronym	Meaning
SIO	Sub-graph in observance	SEU	Structural embedding unit
TSIO	Tiled sub-graph in observance	TEU	Temporal embedding unit
FV	Feature vector	IDC	ID counter
ID	Node identificator	IDL	ID counter list
KLNS	k-largest node selector	DNI	Destination identifier
SER	Structural encoded result	DNL	Destination identifier list
TER	Temporal encoded result	EVA	Event assembler
NP	Neural processor	EVL	Event assembler list
SP	Similarity processor	BSC	Bank of structural centroids
GATH	Gatherer	BTC	Bank of temporal centroids
HSE	High-similarity estimator	BSR	Bank of structural results
TC	Type creator	BTR	Bank of temporal results
TM	Type matcher	EMS	Exclusive memory sector
PVE	Prevalence estimator	CCL	Comparator cluster
SC	Structural centroid	TC	Temporal centroid

We extensively explore the effects on performance and efficiency of varying key design parameters. In general, experimental results demonstrate that DRAGON has unimpeded acceleration levels, conservative resource occupation, competitive accuracy, and responsive fault shielding.

This article makes the following key contributions:

— It introduces the mathematical deduction of temporal feature exchangeability for real-world graphs within the dynamic link prediction context.
— It analyzes the enhancement opportunities of the mainstream hardware-based acceleration approach to dynamic link prediction acceleration.
— It designs and implements a novel resource-constrained multi-processing accelerator for dynamic link prediction, based on fast structural/temporal estimations and resource scheduling.
— It presents the quantitative and qualitative experimentation that demonstrate the virtues and constraints of our approach as a data-driven embedded accelerator.

The rest of this article is organized as follows: In Section 2, we introduce the studies preceding us on this topic and analyze the main challenges. In Section 3, we present the theoretical definitions. In Section 4, we describe the acceleration and deployment strategy. In Section 5, we detail the architecture and datapath of DRAGON. Section 6 describes the experimental setup and Section 7 analyses the results of those experiments. Finally, in Section 8, we conclude this article.

For the convenience of the reader, we present Table 1 with several of the most recurrent acronyms utilized in this article.

2 BACKGROUND AND MOTIVATION

2.1 Steady Algorithmic Progress in GCN and Link Prediction

Link prediction on static graphs has been extensively developed through heuristic methods like BFS and random walks [42, 44]. Other successful studies have proposed more stochastic and even mixed heuristic-modeled approaches [23, 53, 54]. But the capability of learning from real-world graphs and the general applicability of these perspectives have been demonstrated limited [49].

Link prediction through neural networks has received even greater attention in the last years [9, 19, 26, 33, 59]. The mainstream practice introduces encoder-decoder configurations, where the encoder provides dimensional reduction and locality embedding while the decoder redeploys the information to its original dimensionality. The result is the most probable configuration of its future node connectivity [25, 57, 63].

The specific technique differs among authors. In [7, 14, 15, 41], the encoder-decoder is implemented with dense layers at the sides while its temporal variation is processed by a chain of LSTMs in the middle. At the encoding stage of [7], a pair of GCNs are separately applied to the gates of LSTM for improved accuracy. In [31], GRUs instead of LSTMs are utilized, then, an interaction proximity operator right before decoding improves general accuracy.

In [28, 34, 46] algorithms with distinguishable GCN-RNN closely integrated units contribute to embedding chains of timesteps from individual sub-graphs decomposition as a way to expedite the interpretation of dynamic graph information. In [4] these specific type of units are referred to as TNA or temporal neighborhood aggregator. From the algorithmic perspective, dynamic graph information research seems to be steadily and progressively advocating this type of integrated static-temporal solution.

2.2 Previous Work on CNN, RNN, and Static GCN Accelerators

The majority of the recent progress in CNN and RNN hardware acceleration has been almost exclusively focused on Euclidean-type datasets such as pixel and acoustic-based. Nevertheless, the non-Euclidean nature of graph datasets fundamentally differs from the aforementioned.

The relational, non-cardinal and non-explicit allocation of nodes in a graph becomes complex to optimize through traditional methods. The direct implementation of techniques as pruning, quantization, and parallelization, reiteratively elaborated in the Euclidean domain [18, 38, 54, 56] are less effective or simply inapplicable in the graph realm. This is ultimately evidenced in the general lack of proper GCN hardware accelerators in comparison with CNN and RNN.

Among the few GCN accelerators for static prediction tasks proposed to date, four are distinguishable: EnGN [32], HyGCN [60], AWB-GCN [12], and our own contribution ACE-GCN [23]. In the first three studies, authors were fully concerned with coping with the issues provoked by the attempt of mapping naturally ultra-sparse datasets into super dense and more controllable processing structures. This rigid perspective is derived from ingenious but highly complex micro-architectures with costly computing requirements, that like in the case of EnGN and HyGCN, force to circumscribe system implementation to mere simulation environments.

Proper physical implementation was realized by UWB-GCN and its latest version AWB-GCN. In it, a more straightforward approach was based on a cascaded crossbar network. This type of structure assured better transversal access to the pre-arranged graph information, thus improving speed, PE management, and resource utilization. However, the solution still required a large quantity of operators working concurrently, challenging the scalability necessary to implement a potentially more complex CGN-RNN ubiquitous deployment.

In ACE-GCN [23], we have successfully demonstrated the mitigation effects that the gradual circumvention of direct graph neural operators through faster proximity-based estimation may bring over the total computing requirements. Although explicit multi-processing was not considered, the particular results of that study implied that even for a single-core system deployment, a considerable amount of computing potential was being underexploited/underutilized during large portions of execution time.

2.3 Seeking a Proper Approach for a Ubiquitous Link Prediction Deployment

Whereas static GCN acceleration is focused on non-time variant inference tasks such as structural node identification, dynamic link prediction requires the inclusion of temporal information defined by entire sets of chained structural events. This already glances at concerns in the exponential demand on computing and storing capacities for any intended physical deployment.

Although in principle any of the aforementioned static GCN accelerators could be adapted for the dynamic case, only ACE-GCN provides a real resource-constrained alternative. With its superiority core-to-core versus more rigid transversal baselines rigorously demonstrated, ACE-GCN's scalability still heavily depends on the speed of its sole similarity detector to accurately intercept ideal **structural centroids** (**SCs**). A straight solution to the bottleneck would simply consider multiplying the number of similarity cores to simultaneously process separated zones of the dataset. Nonetheless, this naÃŕve approach would also defy the data-driven and resource constriction in aim.

A better multi-processing perspective to link prediction over dynamic graphs has been already applied in PMFO-LP [3]. Based on the well established **Multi-moth Flame Optimization** (**MFO**) computing paradigm [40], the high-performance computing solution PMFO-LP [4] is one of the few practical attempts to accelerate link prediction via system implementation [3, 62]. In analogy to the flight pattern of natural moths around light sources, in PMFO-LP, a network of semi-independent computing elements named "moths" are able to describe regular oscillating trajectories toward the inference solution named "flame".

Just as in ACE-GCN, the accelerative principle of PMFO-LP heavily relies on the particular power-law distribution that real-world graph datasets exhibit to properly balance workload. Nevertheless, its SGD-based super-computing type of deployment and specific performance goals made any resource and energy optimization efforts beyond the scope of that study. In order to transfer the same approach to a low-power and data-driven environment, a significant improvement of its optimization methodology should be developed.

2.4 Data-driven Response and Fault-tolerance Qualities of Real-time Systems

An embedded system is said data-driven when its output closely reflects the changes perceived in its input [2, 24]. This element adds the nuisance of a continuous, unexpected, and mutable input, in front of which most inflexible model-based systems become unusable or highly unreliable [55]. Hence, the capacity of a system for a quick and accurate operational recovery in response to sudden changes shall determine the quality of data-driven response [20, 24].

On the other hand, any automated system should exhibit a level of fault-tolerance quality against potential component failure and the consequent risk of process interruption. Partial or faulty information reception should not tether the capacity of a real-time system to complete its tasks. In this sense, a compromise between functional accuracy, speed-up, and storage requirements is usually reached by introducing a controlled delay tolerance. In this work, we refer to such parameter as the upholding coefficient φ.

3 DEFINITIONS

3.1 Dynamic Graph Link Prediction by Stacks of Convolutional-recurrent Layers

Let $G = (V, E)$ to represent a static undirected graph composed of a set of N nodes like $V = \{v_1, \ldots, v_N\}$, a set of edges $E = \{e_{i,j}\}$, and adjacency matrix $\{A_{i,j}\} > 0$. Graph convolutional neural networks or GCN, aims at synthesizing each node plus the aggregation effect of its associated neighborhood into an embedding space s.t. $R^d (d \ll n)$. This is obtained through the reductive feedforward configuration:

$$GCN^{(l)}(X^{(l)}, A) = \sigma(AX^{(l-1)}W^{(l)}), \tag{1}$$

with $A = D(A+I)$ after identity matrix transformation, activation function $\sigma(.)$, a trainable weight matrix $W^{(l)}$, the input feature matrix $X^{(l)}$, and $GCN^{(l)}(.)$.

Differently from static networks, dynamic networks introduce the uncertainty of time and evolution [45] where graphs are subject to periodic addition/deletion of nodes and edges.

Let $G_{(t)} = (V_{(t)}, E_{(t)})$ to represent a dynamic network G^D s.t. $V^D = \cup_t V_{(t)}\ \forall t \in [0, \ldots, \Omega]$, discretized with a sequence of static timesteps s.t. $\{G_{(t)} = G_{(0)}, \ldots, G_{(\Omega)}\}$. When conditioned on the past adjacency component sequence $\{A_{(t)} = A_{(0)}, \ldots, A_{(\Omega)}\}$, link prediction modeling problem considers a $P(.)$ probability of links to appear at a time $G_{(\Omega+1)}$ through the relation $\hat{A}_{(t+1)} = argmaxP(A_{(t+1)}|A_{(1)}, \ldots, A_{(t)})$.

A commonly seen method for capturing graph dynamics while preserving topology information, has proved an efficient and straightforward approach [4, 7, 31, 41]. The same consists in stacking GCN layers like the mentioned:

$$GCN^{(l)}(X^{(l)}, A)_{(t)} = X_{(t)}. \tag{2}$$

Then, followed by any sort of **recurrent neural network** (**RNN**), for instance, the composed by **gated recurrent units** (**GRU**) [4, 8], the temporal evolution of $G_{(t)}$ can be learned from its $X_{(t)}$ component, with a definitive output like:

$$H_{(t)} = GRU(X_{(t)}, H_{(t-1)}). \tag{3}$$

3.2 Structural and Temporal Equivalence in Real-world Graph Datasets

Large real-world graphs tend to describe the power-law distribution and high sparsity. In this context, vertex degree probability is defined by $P(k) = CK^{-\alpha}$ where the probability of a random vertex connecting to a $k-$number of neighbors is C proportional to $k^{-\alpha}$ for some fixed constant α.

Based on their set of features, a proximity coefficient such as Jaccard similarity may quantify the interception ratio between a pair of static sub-graphs $Z_{(a)}$ and $Z_{(b)} \subseteq G$, then an absolute proximity event can be spotted with a parameterized threshold.

From the property of vertex exchangeability of random graphs, we may consider two static sub-graphs sharing structural embedding equivalence if: $X_a^{(l)} \cong X_b^{(l)} \cong X_E^{(l)}$, for $l \neq 0$, with an inference error shaped like:

$$Q_S \to C * \frac{1}{J(a,b)^\alpha}, \tag{4}$$

where C is a fixed constant, $J(a,b)$ is the Jaccard coefficient between $Z_{(a)}$ and $Z_{(b)}$, and α becomes a design parameter.

In this context, if the structural embedding sequences of $Z_{(a)}$ and $Z_{(b)}$ are mutually equivalents throughout time i.e., $\forall t \in [0, \ldots, \Omega]$, it provides that:

$$\{X_{a(1)}, \ldots, X_{a(t)}\} \cong \{X_{b(1)}, \ldots, X_{b(t)}\}. \tag{5}$$

The temporal embedding $H_{a(\Omega+1)}$ can be estimated from $H_{b(\Omega+1)}$ through a relation like:

$$H_{a(\Omega+1)} = GRU(X_{E(\Omega)}, H_{b(\Omega+1)}). \tag{6}$$

Then, a temporal equivalence error Q_T is considered proportional to the structural equivalence error Q_S throughout $\forall t \in [0, \ldots, \Omega]$ when diminished by a constant β s.t.,

$$Q_t \to Q_s * \frac{1}{t^\beta}. \tag{7}$$

In general, this implies that the probability of $H_{a(t)} = H_{b(t)}$ depends on the continuity in time of their structural equivalence. Moreover, the deeper their equivalence in time, the smaller the uncertainty of their exchangeability.

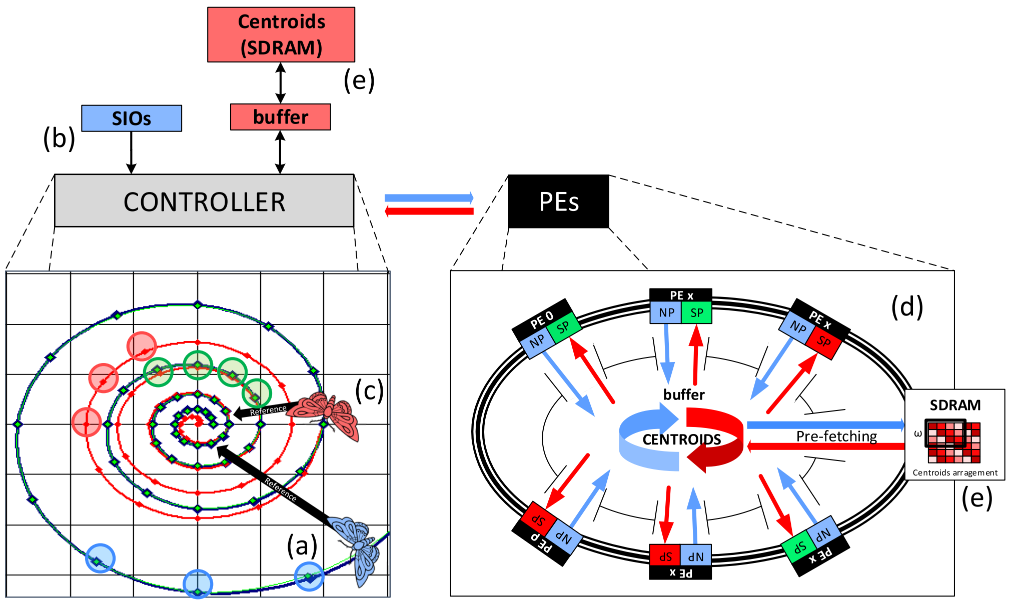

Fig. 1. A graphic description of the "multi-polar centroid contribution" principle developed for proximity-based multi-processing of dynamic link prediction. Based on the well-established multi-moth processing paradigm, in DRAGON we leverage the inherent tendency of the processing path of different moths to gradually transverse trajectories. The blue moth (a) represents a new incoming sub-graph partition or SIOs (b). Its path calculation is represented by the green dots. The red moth (c) represents previously-stored path information. These become centroids for the proximity check that is parallelly carried with path calculation. A positive sequence of interceptions helps the platform to estimate the ultimate inference solution and circumvent the slower and more expensive path processing of the blue moth. At (d), a conceptualization of the ring-shaped PE network for the multi-polar centroid contribution hardware implementation. A chained network of PEs is decoupled from the role of moths as a method to provide leaner hardware distribution. Each PE is composed of an NPs) and an SPs, for direct or estimated neural operation respectively. Path oscillation is capitalized through a commonly accessed bank of centroids, NPs write to the bank while SP reads from it in order to reference the path oscillation toward the solution. In the example, some (red-cased) SPs have not found matching centroids. These centroids will have another chance at the contiguous SPs (recycled) till finding a matching SIO (green-cased). Centroids are massively stored at SDRAM (e) with a generally sparse arrangement and prefetched inside the process through an intermediary buffer named ω window.

4 METHODOLOGY

4.1 The Multi-polar Centroid Contribution Approach

In the PMFO-LP study, an important characteristic that is left unheeded is the strong tendency of moths to progressively overlap their trajectories toward inference solutions. As shown in Figure 1, this trajectory redundancy may be calculated with a lightweight proximity-based operator, registered, and controlled from a common parameter through some centralized scheduling platform.

The utilization of a proximity-based approach also allows the exploitation of the self-descriptivity capacity that high-dimensional datasets may provide. This can be managed with the introduction of concurrent proximity checks at several dimensional-reduction stages without halting the process. This characteristic is also overlooked at PMFO-LP, where graph information needs to be arranged and passed through a single lengthy dimensional-reduction stage at pre-processing, which ultimately affects its fault-tolerance and data-driven practicality.

Based on a platform like the proposed, a partially transversal path reference can be gradually calculated from each separated moth. This will have the effect of mutually reducing moths path oscillation and ultimately, quickening arrival of the solution with high accuracy.

The consequent undirect influence between moths would not only lead to a faster reference optimization, thus inference, but also open up the possibility for resource management otherwise absent in PMFO-LP. In general, we refer to this MFO/proximity-based path-circumventing methodology as the "**multi-polar centroid contribution**" approach and it represents the key acceleration premise behind our design DRAGON.

Differently than in PMFO-LP. in DRAGON, we decouple the role of "moths" from the algorithmic perspective to the physical instantiation of PEs, this allows us to implement a more flexible PE scheduling system. According to the level of trajectory overlapping, an associated moth completes its algorithmic paths by equating its future trajectory to any matching centroid. Meanwhile, the actual physical PE associated with that operation is now made available for processing new additional SIOs, improving parallelism and hardware efficiency.

Through this method, distinct physical PEs can simultaneously detect, contribute, update to, and recourse from a common bank of centroids that becomes a single-parameterized reference and auto-completion system for optimal moth oscillation. In our study, the term "**multi-polar**" relates to the ability of the system to progressively influence/optimize algorithmic moths processing routes by exploiting the **polarized** distribution of centroids through a network of concurrent pseudo-independent PEs.

Distributed contribution to a shareable parameter pool like the proposed may help also to improve inference accuracy due to a more favorable stochastic representation. Moreover, from a micro-architecture perspective, recoursing from a proximity-based approach allows transferring the natural storage complexity of a multi-dimensional source graph to a much simpler indexable arrangement. Thus, the information can be registered in memory with a generally sparse distribution, more easily accessed and processed through direct cascaded partitioning architecture like the one deployed by AWB-GCN [12].

In analogy to machine learning related terms, we call "training" the period that precedes the stabilization point of centroids optimization, while the period that immediately follows is referred to as "trained". Once the bank of centroids reaches an optimal trained point, the centroid optimization mechanism can be disabled and may directly serve future evaluations.

All in all, besides the multi-processing generation parameter ρ the scalability and performance of our accelerator will depend on four additional design parameters:

(1) The length of the sliding window "ω", (2) The capacity for SC/TC at the BTR/BTC defined by "α" (with "$\beta = \omega * \alpha$"), (3) The capacity for proximity events per SC at EVL "ϵ", and (4) The fault-tolerance upholding coefficient "φ".

4.2 Physical Deployment Strategy

We interpret the multi-polar centroid contribution to both structural and temporal phases of graph embedding involved in the link prediction algorithm. This is materialized in the unified meta-heuristic ring-shaped network at the heart of DRAGON, which is capable of jointly hosting convolutional and recurrent neural functions along with their associated circumventing proximity-based operators. Proximity calculations are provided by a modified version of the similarity-detector included within ACE-GCN [23].

For the structural embedding phase acceleration, the reference and auto-completion platform gravitates around feature-based SCs. The choice allows us to take advantage of the high dimensionality of source datasets by introducing earlier structural proximity checks and circumvention right before pre-processing.

For the temporal embedding phase, proximity check is done over the structural historical record of SIOs instead of their feature vectors. In this aspect, **temporal centroids** (**TCs**) are described by chains of **structural embedding results** (**SER**) and encoded under the consequential **temporal embedding results** (**TER**).

The "Implicit-processing-by-association" workload circumventing approach developed at ACE-GCN, is further adapted for the dynamic link prediction case as follows:

Based on relations (10) and (11), it is possible to estimate a temporal embedding $H_{a(t)}$ an SIO Z_a from a pre-stored $H_{b(t)}$ of an SIO Z_b if and only if their mutual structural equivalence is deep enough in time while considering Q_T as an accuracy loss susceptible to storage parameter β.

To evaluate structural equivalence continuity, chains of structural encoded results like (10) are directly stored. A single chain will have a maximum depth of Ω timesteps with a striding window of ω timesteps ($\omega \subseteq \Omega$), where each ω striding over Ω is linked to a specific temporal encoded result represented in $H_{(t)}$.

Concluding from (12), wider ω may have the effect of improved accuracy due to information uncertainty reduction. From the prediction $H_a(\Omega + 1)$ a regular classification layer is implemented to identify the SIO structural type at the predicted stage.

Having the potential structure reduced to a certain centroid significantly eases links deduction which in turn can be obtained with a simpler heuristic method. For this effect, we employ a historical ranking approach with nodes IDs registered by their matching ratio to the particular predicted SC throughout Ω timesteps.

5 SYSTEM ARCHITECTURE OF DRAGON

5.1 Components Overview

The architecture of DRAGON is composed of sets of functionally semi-independent modules that have been linked together to form super-modules. Their functions are controlled by local FSMs (finite state machines) which are in turn driven by successive layers of FSMs.

Ultimately, the entire custom logic is driven by a top FSM that also serves as an interpreter-scheduler with the rest of the external SoC components through the native communication protocol.

The memory format of DRAGON follows similar nomenclature to that in ACE-GCN. Nodes feature vectors are represented under sets of 64-bits flexible fixed-point [17, 58], divided into 16 bits for node **identification** (**ID**) and 47-bits of feature operational information. From the last group, 1 MSB represents the sign, next 4 MSB the integer and, 42 LSB of fractional information.

Originally stored at off-chip SDRAM, SIOs are buffered and scheduled by a data-driven gathering unit to the PE network according to availability. Stored centroid access from SDRAM occurs more directly using a time-division-multiplexed pre-fetched mechanism as the included used in ACE-GCN. Due to the considerable centroids recycling capacity allowed by the high representativity of dataset, SIOs off-chip access is designed with priority over centroid access.

We categorize the entire architecture into four main circuits: Pre-processing, neural, proximity, and heuristic circuits. Details can be seen in Figure 2 and further explained as follows:

Pre-processing circuit: This circuit is the first to come into contact with the source dataset and it is fundamental to the real-time and data-driven functionalities of the accelerator. Its general purpose is to receive and prepare the dataset prior to the actual embedding process. Mostly represented by the **gatherer** (**GATH**) and its sub-module the ***k*-largest-node selector** (**KLNS**).

— **Gatherer (GATH):** The purpose of this module is to constantly track the arrival of graph information and whenever requirements have been met, assemble SIOs and prompt their processing down to the PE network. Based on the original implementation of our novel

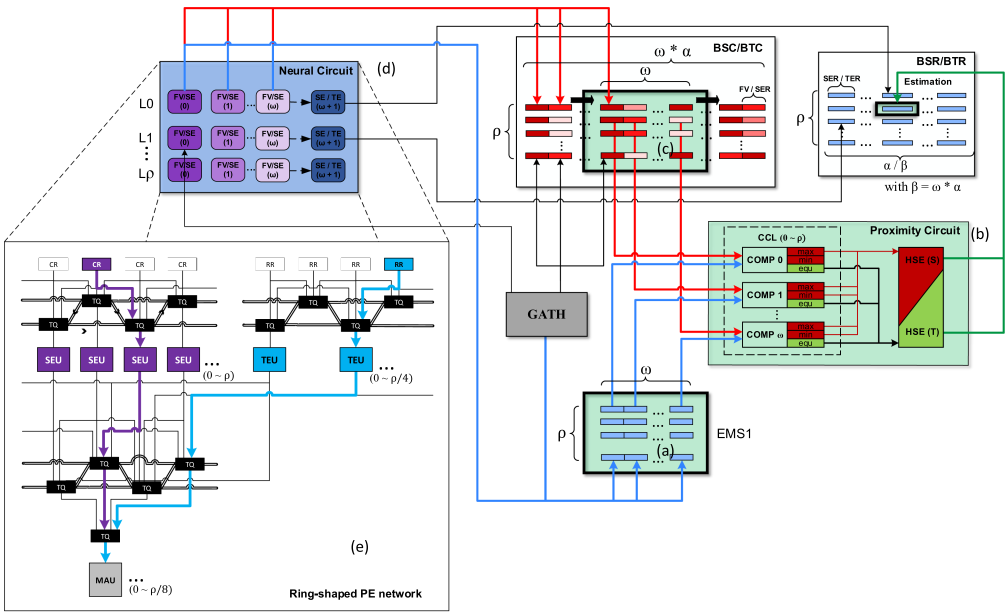

Fig. 2. A representation of the core micro-architecture of DRAGON (after KLNS abstraction—before heuristic stage). (a) EMS1 with size $\omega \times \rho$ allocates the tiled SIO to be compared through the proximity circuit (b) against a counterpart SC/TC (c) stored in a general sparse arrangement within BSC/BTC to ensure stochastic cross-representation. For this, a buffered ω window of size $\omega \times \rho$ unidirectionally strides over BSC/BTC with a total length of $\omega * \alpha$. The first level comparison is done by a set of ρ comparators clusters or CCLs cores, each composed of ω parallel comparison cores that provide specific results to a single high-similarity-estimator or HSE at two types of the operational modes: structural (S) or temporal (T). HSE is then able to identify or predict the status of a particular SIO by invoking the encoded result from BSR/BTR (of size $\rho \times \alpha$ for (S) or $\rho \times \beta$ for (T) (with $\beta = \omega * \alpha$) without incurring in the neural circuit. Centroids are generated by the neural circuit (d). It follows a one-hop distribution smoothing like in AWB-GCN [12]. Structural embedding units (SEUs), temporal embedding units (TEUs), and multiply-accumulate units (MAUs) are unidirectionally cascaded through three levels of task schedulers named TQs. These elements define the NPs and in conjunction with the CCLs of the proximity circuit (included within SPs) form the ring-shaped PE network of DRAGON.

work ACE-GCN [23], it has been modified to track and launch the processing of multiple SIOs simultaneously. The input information of this module is $G_{(t)}$ for a particular timestep t, defined by its graph adjacency matrix and graph feature matrix. Datasets are purposely arranged in a non-stochastic manner as to originate a data-driven fault-prone environment.

— **k-largest node selector (KNLS):** This sub-module is functionally integrated under GATH and provides the necessary pre-processing to the raw dataset before it can be effectively scheduled to any PE available. Internally, it produces a sub-graph abstraction and partition following the novel reductive heuristic model from [11] where "k" is a design parameter influenced by the graph total degree. The output is the resulting $Z_{(t)}$ also named tiled SIO in ACE-GCN. It has a shape: $(k + 1) * u^1$, where u^1 is the first layer feature dimension prior to the neural dimensional reduction.

Neural circuit: Due to its inherent complexity, most of the computational load within DRAGON is executed by the neural circuit. It provides all the necessary neural operations for the proper structural and temporal graph embedding process, including fully connected, convolutional and recurrent.

Internally, the circuit is arranged in a ring-shaped network of **neural processors** (**NPs**) (jointly with the SPs from the proximity circuit) with each neural NP composed of three types of units: SEUs, TEUs, and MAUs.

Each type is generated proportionally to a multi-processing instantiation parameter named ρ accordingly $\rho_{SEU} = \rho$, $\rho_{TEU} = \rho/4$, and $\rho_{MAU} = \rho/8$. These relations have been obtained through comprehensive timing and ablation analysis.

Units are concatenated following a cascaded one-hop smoothing configuration like in the novel work AWB-GCN [12] for static GCN. The method seeks to produce a deadlock-free transit and an optimal resource distribution throughout MAC resources. We have extended this idea and decoupled the role of CNNs (SEUs) units from MACs (MAUs) to include GRUs (TEUs) within the resource distribution. Unlike in AWB-GCN, in our design, this process is controlled from three levels of task queues or TQs instead of one. Each processing family is briefly introduced as follows:

— **Structural embedding units (SEUs):** These units produce the necessary SIO structural embedding in a specific timestep as defined in Equation (2), i.e., the information synthesis of a particular SIO and its one-hop neighborhood. The internal architecture of SEUs is based on the core processor of our novel work at ACE-GCN [23]. We have expanded its multi-processing capabilities through the multi-polar centroid contribution approach as explained in the methodology chapter of this work. For this instance, each SEU can independently and simultaneously contribute to and read from a shared parameter platform centered around the concept of proximity centroids. The exception would be the exclusive memory sectors EMS1 and EMS2 that are unavoidably assigned for the operative requirements of each SEU and cannot be shared.

— **Temporal embedding units (TEUs):** The purpose of TEUs is to calculate the temporal embedding from the structural embedding obtained by each of its SEU counterparts as defined in Equation (3). Internally, it performs the basic GRU operations [8]. Its design follows an RNN circuit explicitly cascaded in three layers. Its non-linear operations are obtained from pre-trained LUTs and stored weight vectors. Similar to its SEUs counterpart, each GRU request is serviced with TEUs, and these in turn with MAUs according to a centralized task queuing system of three levels.

— **Multiply-accumulate units (MAUs):** Neural operations inside SEUs and TEUs are provided by the multiply-accumulate resources from the MAU network according to availability and priority. They are pre-configured following a flexible fixed-point format [17] with ranges provided by pre-analysis [52]. Since recurrent operations are less frequent than convolutional, with this method, idles TEUs cycles are better distributed to contiguous SEUs, guaranteeing a leaner resource utilization and operational flexibility.

— **Temporal classification layer (TCL):** TCL is a fully connected neural network reductive to α classes that produce the definitive SC prediction regarding $t = \Omega + 1$ from the definitive TER $H_{(\Omega)}$ at $t = \Omega$. Since it's a non-frequent operation, it is serviced by the MAUs network through TQs just as SEUs and TEUS but with top priority.

Proximity circuit: This circuit performs the key acceleration strategy behind DRAGON. It provides a faster circumvention mechanism that alleviates the computational load from the more demanding neural circuit. Its most relevant module is the **high-similarity estimator** (**HSE**) assisted by its sub-modules: **prevalence estimator** (**PVE**), **centroid matcher** (**CTM**), **centroid creator** (**CTC**), and **comparator clusters** (**CCLs**). The conjunction between CCLs and HSE constitutes the **similarity processors** (**SPs**) and is arranged as a ring-shaped network.

— **High-similarity estimator (HSE):** HSE works under two modes regarding the type of embedding stage that is seeking to circumvent: structural (S) and temporal (T). It operates over two large SDRAM sector denominated **bank of structural centroids** (**BSC**) and **bank of temporal centroids** (**BTC**) for each mode respectively. BSC and BTR share the same design parameters, with capacity for α centroids with each described by ω vectors multiplied by ρ processors. An encoded centroid is the symbolic representation of a particular embedding result that eases its memory indexation. BSC and BTR associated encoded results (SER and TER) are stored at two memory sectors denominated **bank of structural results** (**BSR**) and **bank of temporal results** (**BTR**), with sizes of α for (S) and $\beta = \omega * \alpha$ for (T) times ρ. The exact meaning of ω depends on the mode. In (S), each ω allocation represents a feature vector FV, together they describe a particular SC with its structural embedding result encoded at BSR as a SER. In (T), each ω represents a proper SER, several SERs form a TC, which is encoded at BTR as a TER. The BSC/BTC-EMS comparison algorithm is carried out through a buffer named "ω window". Just as its EMS counterpart, ω window has a size of $\omega \times \rho$ that allows the expedited comparison of up to ρ centroids in parallel. In (S) mode, window striding is done centroid to centroid, i.e., every ω FVs. In (T) mode, striding is done SER to SER i.e., column to column. In both cases, strides are indexed to a respective SER/TER at BSR/BTR. Details of this function are further explained in Figure 2.

— **Comparator clusters (CCLs):** In analogy to the split between SEUs and TEUs from its multiply-accumulate operators MAUs, a single HSE module is scheduled among multiple CCLs according to availability. Each CCLs is allocated aside their neural counterparts within an SP at the ring-shaped processing network for a total of ρ CCLs generated. Internally, each CCL is composed of ω comparators that provide parallel partial processing to ω elements of the BSC/BTC-EMS algorithm with specific output to HSE.

— **Prevalence estimator (PVE), centroid matcher (CTM), and centroid creator (CTC):** The purpose of these is to serve as a breach between HSE and BSC/BTC memory controller during the centroid comparison process. PVE controls the number of centroids in bank by estimating their historic prevalence, deciding their deletion or inclusion accordingly. CTM registers any successful similarity event taking place while CTC creates new centroids if no similarity event has been generated.

Heuristic circuit: The heuristic circuit includes all the modules involving the final meta-heuristic algorithm to the definitive link prediction. It is composed of three modules EVA, IDC, and DNI. A brief introduction of each is presented as follows:

— **Event ID assembler (EVA):** The purpose of EVA is to register the ID composition of any SIO successfully matching a particular SC, i.e., whenever a structural similarity event has been detected. The operation utilizes an event list named EVL with capacity for α SCs, ϵ events per SC, and k IDs per event. Due to the high consistency in dataset distribution, a single full EVA collection from a single timestep $G_{(t)}$ is reusable for all incoming timesteps.

— **SIO ID counter (IDC):** The purpose of IDC is to keep a register of all the neighboring IDs ever involved with a specific SIO after its KLNS abstraction. In this regard, each ID is ranked according to the number of detections within an SIO during the entire Ω scope analyzed. These operations utilize a set of small, dedicated registers named IDLs with a total of N IDLs, i.e., one for each potential SIO.

— **Destination nodes identifier (DNI):** At the end of the process and after having reduced the uncertainty to a known SC, DNI completes the SIO link prediction by inferring its most probable ID composition at the future timestep $t = \omega + 1$. For this operation, it realizes a heuristic search at EVL of the event in the predicted SC with the highest ID popularity ratio according to IDL. Details on this process are further explained in Figure 3.

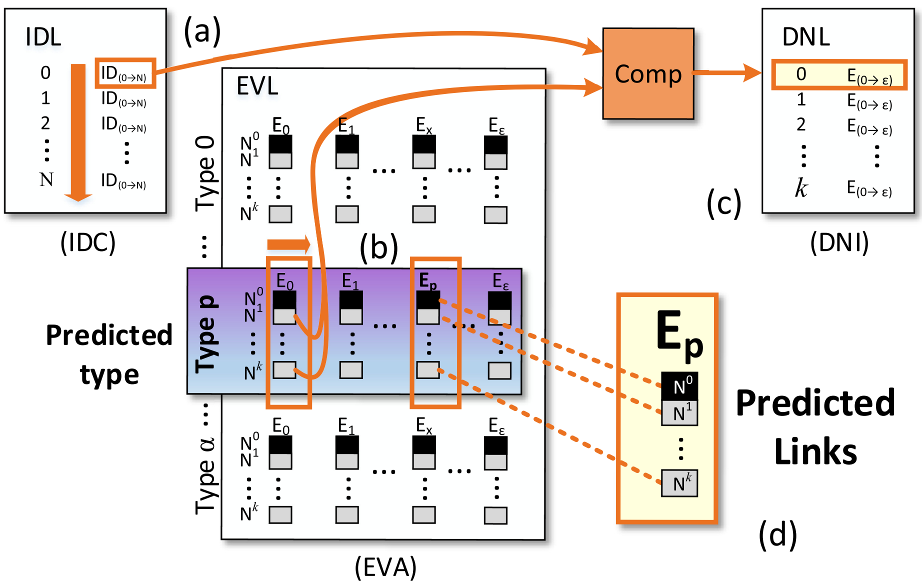

Fig. 3. A representation of the heuristic circuit and its ranking-based link prediction method. (a) DNI module starts by reading IDL belonging to the predicted SC ($t = \Omega + 1$) in a top-down fashion, from the ID with more occurrences to those with fewer or none. (b) DNI makes a fast comparison of the ranked IDs in IDL with the IDs within each of ϵ proximity events within the EVL sector dedicated to the predicted SC. (c) Each positive interception increases a counter for the respective event that is ranked within DNL. (d) Whenever any of the events have reached at least k, such event is the most probable definitive structure of $Z_{(\Omega+1)}$. For processing the next SIO, EVL remains untouched while IDC and DNL are reset and declared available.

The distinct circuits and their components communicate with each other by following a specific datapath as shown in Figure 4. Details of the datapath are further described:

5.2 Datapath Description

(1) The process starts with the arrival of the first temporal timestep $G_{(t)}$ at time $t = 0$, i.e., $G_{(0)}$ from the full set G to GATH. As data arrives fragmented and unorganized, GATH continuously tracks $G_{(t)}$ for enough information to declare a subgraph-in-observance or SIO. This is obtained through the sub-module KLNS which produces the sub-graph partition/abstraction denominated $Z_{(t)}$ or tiled SIO. Meanwhile, during the partition operation, KLNS transmits the selected IDs to the IDC. This module keeps all the IDs ever involved with a particular SIO, ranked by the number of appearances throughout the full length of time analyzed ($t = 0, \ldots, \Omega$). The detected $Z_{(t)}$ is distributed to one of the ρ available CCL at the ring-shaped PE network. Step (1) continues in parallel to the rest of the process until detecting and launching each potential SIO (N) within $G_{(t)}$.

(2) At the proximity circuit, with HSE in (S) mode, $Z_{(t)}$ at EMS1 is evaluated against BSC through the ω buffer window, at strides of ω (for a maximum of α strides), if a high-similarity event (structural) is NOT detected process continues to (3.a), otherwise, it continues to (3.b).

(3) (a) $Z_{(t)}$ is scheduled to any of the ρ available SEUs at the neural circuit, this produces its structural embedding $X_{(t)}$. PVE sub-module analyzes storage availability and directs CTC to create a new SC within BSC. This transfers $X_{(t)}$ jointly with its encoded SER to BSR and writes the FV composition of $Z_{(t)}$ into BSC from the next α position available. The process continues to (4). (b) HSE reports the matching SC to EVA at the heuristic circuit, then EVA

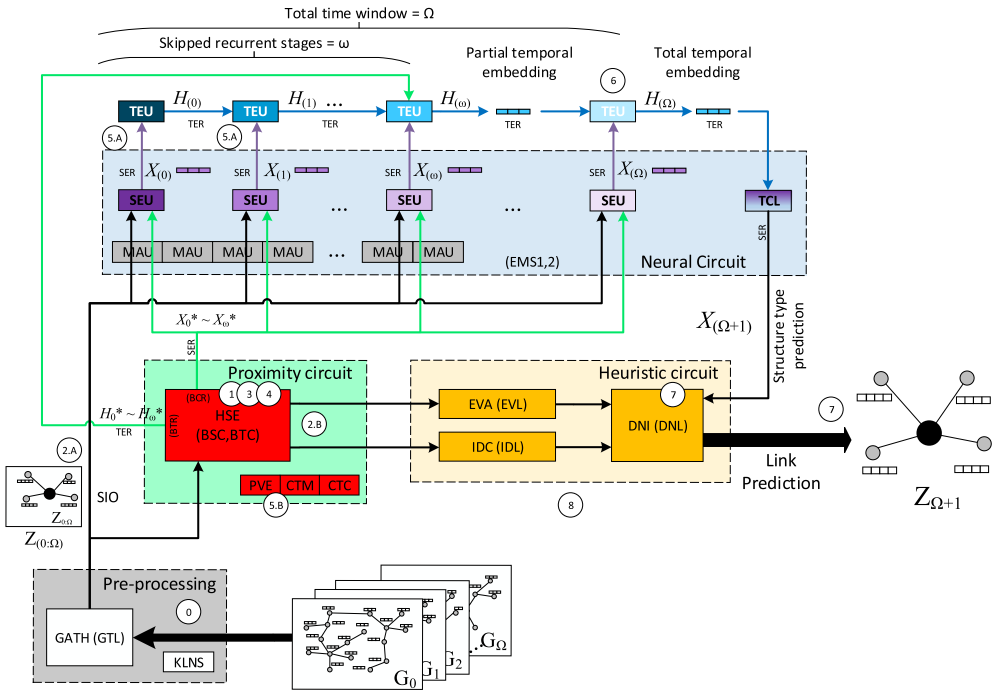

Fig. 4. A representation of the datapath running in DRAGON. Numbered circles indicate the different steps of the process. Raw unorganized graph source is transferred to the pre-processing circuit (grey box) that produces SIO partitions $Z_{(0,\dots,\Omega)}$. In (S) mode, the proximity circuit (green box) searches for a proper SC in BCR that allows a SER fast estimation so as to avoid slower SEUs (small purple boxes) intervention, likewise for all future timesteps of Z (fading purple boxes). Proximity circuit in (T) mode checks for a concatenation of at least ω timesteps of $Z_{(0,\dots,\Omega)}$ matching a TC in BTR. If found, the related TER replaces the next TEU operation. If it has reached the last timestep, the process takes the definitive TER to TCL. This produces the predicted SC for $Z_{(\Omega+1)}$. The actual ID configuration of $Z_{(\Omega+1)}$ is calculated through the heuristic circuit (yellow box). Through a search-and-rank dynamic, it launches the most probable ID neighboring the reference ID at the center of $Z_{(\Omega+1)}$.

passes the ID composition of $Z_{(t)}$ to the EVL segment dedicated to that specific SC. PVE and CTM update the prevalence of the matching SC, directly invoke the encoded SER from BSR and temporarily store it at EMS2 (on behalf of the specific $Z_{(t)}$). The process continues to (4).

(4) The CCL is declared available. If there has been processed and accumulated in EMS2 at least ω instances of the same $Z_{(t)}$ (with $t \in [0,\dots,\Omega]$) the process continues to (5), otherwise it returns to (1) to track for updates on the same or another $Z_{(t)}$.

(5) HSE in (T) mode compares the encoded SER at EMS2 against BTC through the ω buffer window, in strides of 1 (for a maximum of $\omega * \alpha$ strides) seeking for any equivalent chain of at least ω encoded SERs. If a high-similarity (temporal) event is NOT detected the process continues to (6.a), otherwise it continues to (6.b).

(6) (a) The ω concatenated SER vectors of $Z_{(t)}$ are jointly scheduled to a single available TEU at the neural circuit. The selected TEU will sequentially process the vectors to produce $H_{(t)}$ encoded as a TER which is representing a single stride of the window ω. The process continues to (7). (b) PVE and CTM update the prevalence of the matching TC, directly invokes the encoded TER associated with the matching chain from BTR, and feed it to any available TEU as $H_{(t)}$ input. The process continues to (7).

Table 2. Synthesis and Benchmarking Equipment Utilized in this Study

DRAGON	MFO	DynGEM	node2vec / dynnode2vec
Intel Stratix 10 SX (2800)	IMAN1 (CELL processor)	32 cores CPU	7 cores CPU
244 Mb SRAM	3.2 Ghz	2.6 Ghz	3.6 Ghz
16 GB DDR SDRAM	256 MB per CELL	128 GB RAM	64 GB RAM
		GPU K40C	GPU: N/A

PMFO-LP hardware information is deducted from CELL nominal information [6]. DynGEM, node2vec, and dynnode2vec common equipment information is directly provided by dynnode2vec study [37].

(7) For $H_{(t)}$, If $t < \Omega$, CTC writes TER at BTR in a position according to the last timestep analyzed, then the process returns to (1) to wait for updates of the same or additional $Z_{(t)}$. If $t = \Omega$ then the temporal sequence has reached the last timestep, in this case, $H_{(t)}$ is passed directly to TCL to produce a new predicted SER for timestep $t = \Omega + 1$, then the process continues to (8).
(8) At the heuristic circuit, DNI reads the SIO predicted SER and localizes the EVL sector related to the "winner" SC. DNI compares the IDs inside each event at the winner SC with IDL and ranks them by the level of concurrence. For instance, if no event was found to contain this ID, DNI checks the next ID iteratively until confirming at least k IDs. The link prediction of current $Z_{(t)}$ at $t = \Omega + 1$ will be equal to the IDs that form the specific winner event from the winner SC at EVL, with the winner event signalized by the very top of DNL. The process continues to (9).
(9) After the specific SIO link prediction is completed, EMS1, EMS2, IDL, and DNL are reset, while EVL is kept unchanged being valid for all successive SIOs at this point. The process continues to (2). END.

6 EXPERIMENTAL SETUP

DRAGON has been designed as a customized logic within a generic SoC environment. It is not attached to any specific IP, except for those related to the native bus communication and I/O resources. The specific synthesis device is an Intel programable acceleration card D5005 with a Stratix V GX 5SGXEA7K2F40C2 FPGA chip. More details of the implementation device of DRAGON and the rest of baselines are shown in Table 2. Optimal values for neural weights, bias, and flexible fixed-point distribution have been pre-calculated by software means [13, 52].

To demonstrate the applicative and functional characteristics of our accelerator, we analyze DRAGON from three main perspectives:

Algorithmic characteristics: As DRAGON runs over a hardware-based acceleration premise, we firstly compare our approach against mainstream software implementation. In this sense, we present the execution time and AUC accuracy of DRAGON for AS and Hep-th datasets (detailed in Table 3) [29], under eight different magnitudes at $\rho = \{1, 2, 4, 8, 16, 32, 64, 128\}$.

The results are contrasted with those reported in three state-of-the-art algorithms, deployed in software and regular multi-purpose CPU-based systems, these are DynGEM [15], node2vec [16], and dynnode2vec [37].

Next, taking dynnode2vec results as baseline and Hep-th as dataset, we present the effects on speed-up and accuracy, of modifying the key design parameters in DRAGON: ω, α, ϵ, and φ and analyze the slope caused in the graphic.

Physical characteristics: To analyze the specific hardware qualities of DRAGON, we take PMFO-LP [3] as the baseline. Basic acceleration principles of PMFO-LP are shared by our design, however, PMFO-LP is deployed in a super-computer named IMAN1, motored by a network of CELL

Table 3. Dataset Utilized in Present Study

Dataset	V	E	Degree
USAir	332	2,126	6.4
NetScience	1,589	2,742	1.72
Political Blogs	1,222	5,366	4.39
Yeast	2,375	6,136	2.58
Router	5,022	6,258	1.24
King James	1,773	9,131	5.15
Hep-th	34k	421k	12.38
AS	6k	13k	2.16

Number of vertices (V), number of edges (E) and degree.

processors and plenty of computing power and memory capacity. Inherently, PMFO-LP does not concern with resource consumption as much as with raw acceleration.

In this context, we present the speed-up of DRAGON under "training" and "trained" modes against the results reported in PMFO-LP, throughout six different size datasets (as detailed in Table 3): USAir, NetScience, Political blogs, King James, Yeast, Router [43, 51].

Since PMFO-LP does not report on power efficiency, we have calculated a relationship between the nominal power consumption of CELL [6] (estimated on 80 Watts at 4 GHz), the number of CELL units involved, and the reported graph/execution time. Please notice that the estimation does not include other of their components potential expenditures.

We contrast the calculated value to the power requirements of DRAGON which is obtained through Quartus power analyzer built-in tool. Next, based on the aforementioned results, we report the scalability quality of DRAGON versus PMFO-LP. Finally, it is reported the on-chip memory and area demand for the maximum size configuration at $\rho = 128$.

Functional characteristics: To introduce the data-driven benefits of DRAGON, we have constructed a bipartite dataset that intermingles between a pure AS and a modified version of Hep-th pruned to the size of AS.

Although both datasets exhibit relatively similar degrees, they correspond to different originating phenomena, thus containing different structures. The intention is to mimic the conditions of a realistic data-driven environment where a sudden change in the structures populating the sensorial network may occur.

The ability of the system to recover from data-driven changes should be in line with its neural resource's promptness to come back to zero utilization. In this regard, we analyze DRAGON under two modalities: (a) Full implementation and (b) Implementation excluding the mechanism for partial update/delete of SCs, in other words, removing the ability of the system to deal with the arrival of new structures after precedent structures have been detected.

This way, data-driven quality should be measurable by the ratio of direct neural operations throughout time completion. Since DRAGON contains networks of PEs, it is also necessary to study the general workload distribution. For this, we calculate and present, the median standard deviation based on the totality of NPs workload throughout completion time, with the number of SEUs varying by $\rho = \{8, 32, 128\}$, accordingly.

7 PERFORMANCE ANALYSIS

7.1 Algorithmic Characteristics

In Figure 5(a) and its respective report in Table 4, it can be observed that in general, the smaller configurations of our hardware-based accelerator were directly able to surpass two of the software-

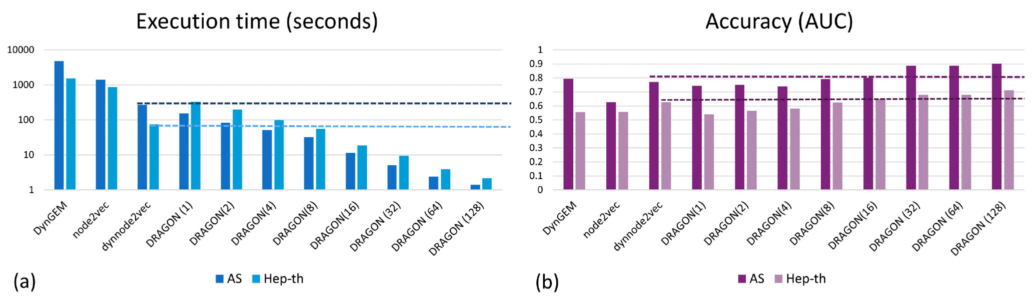

Fig. 5. (a) From left to right, link prediction execution time in seconds (logarithmic scale) for datasets AS (dark blue bars) and Hep-th (light blue bars). The comparative includes three state-of-the-art algorithms results as reported in articles, versus DRAGON under three different size configurations $\rho = \{1, 2, 4, 8, 16, 32, 64, 128\}$. Dotted lines mark the best results from baseline and are utilized as a reference. In all cases, data is delivered organized and steady, i.e., not considering the data-driven scenario. (b) From left to right, link prediction accuracy is quantified by **area under curve** (**AUC**) for datasets AS and Hep-th. Each reading belongs to its respective measure from (a).

Table 4. Numerical Report on (a) Link Prediction Execution Times (seconds) and (b) the Related Accuracy Levels (AUC) for Datasets AS and Hep-th

(a)

	AS	Hep-th
DynGEM	4,812	1524
node2vec	1,393.2	865.2
dynnode2vec	269.4	75
DRAGON (1)	152.58	329.45
DRAGON (2)	83.16	196.65
DRAGON (4)	51.32	99.79
DRAGON (8)	32.48	55.15
DRAGON (16)	11.45	18.90
DRAGON (32)	5.12	9.53
DRAGON (64)	2.39	3.96
DRAGON (128)	1.42	2.15

(b)

	AS	Hep-th
DynGEM	0.7949	0.555
node2vec	0.6258	0.5577
dynnode2vec	0.771	0.6269
DRAGON (1)	0.7436	0.5398
DRAGON (2)	0.7514	0.5645
DRAGON (4)	0.7403	0.5816
DRAGON (8)	0.7918	0.6236
DRAGON (16)	0.8022	0.6501
DRAGON (32)	0.8407	0.6734
DRAGON (64)	0.8875	0.6810
DRAGON (128)	0.9019	0.7128

A total of eleven models were compared, including three CPU-based baselines (as reported) and hardware-based DRAGON at eight different size configurations.

based baselines i.e., DynGEM and node2vec for a good margin. This already demonstrates a successfully functional implementation of the prediction task.

Specifically against the fastest baseline dynnode2vec, a single core of DRAGON only provides a mild acceleration in the case of AS (1.76× dynnode2vec) or is markedly slower in the case of Hep-th (−4.39× dynnode2vec). As a hardware-based accelerator, it is expected to widely surpass its baselines running at intrinsically less efficient multi-purpose CPUs.

Looking in detail at the evolution of size configurations, we can observe that the performance of DRAGON is greatly improved by the number of PEs in the network. In the case of AS, the execution time is drastically reduced down to the largest configuration tested at $\rho = \{32, 64, 128\}$, with speed up reaching around 5×, 112×, and 191× dynnode2vec, respectively.

This outstanding scalability is even more noticeable for the case of Hep-th, where our accelerator was already able to surpass dynnode2vec at a size of $\rho = 8$, producing a definitive speed-up of 34.88× dynnode2vec at $\rho = 128$.

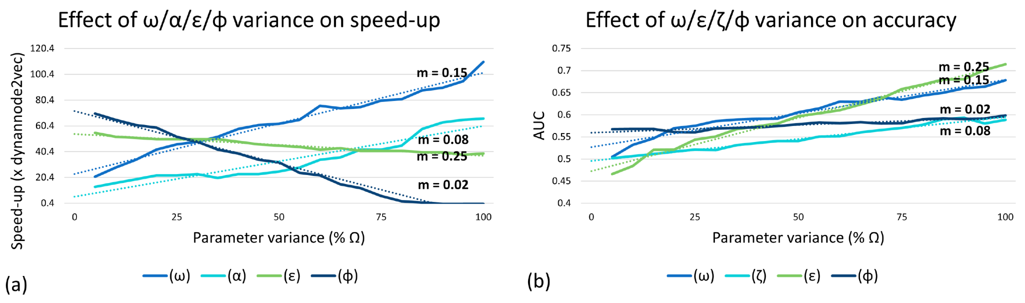

Fig. 6. (a) Effects of key parameters variance on speed-up regarding reported baseline results. At the abscissa, parameter's variance is shown in terms of percentage from the total temporal length (%Ω). The parameters studied are the size of sliding window ω, total storage capacity for centroids α (102×), total storage capacity for events per centroid ϵ and the upholding coefficient φ (14× cycles). At the ordinate, the speed-up are obtained from report (× dynnode2vec). Parameter's behavior is analyzed from the slope m generated by each respective curve. (b) Effect of key parameters variance on accuracy (AUC).

Another important observation is that, since the considered number of timesteps is different for AS than for Hep-th, baseline algorithms report longer execution times for the first dataset than for the second. In contrast, our accelerator exhibits longer processing times for Hep-th than for AS. This indicates that DRAGON is influenced more by the degree of the graph than by the number of temporal stages. Dataset sparsity and the number of PEs also play a fundamental role in the accuracy of DRAGON.

The approximation technique within NPs initially induces a small degree of inaccuracy to DRAGON. As can be seen from Figure 5(b), this effect is less evident for the AS dataset, but it is distinguishable for Hep-th and its comparatively lower sparsity (12.38 degree points). Also, smaller configurations of DRAGON tend to perform with lower accuracy than those reported by dynnode2vec. However, at larger configurations, the metric tends to progressively improve.

In this order, at ρ = 16 we can already observe an advantage over dynnode2vec, with AUC = 0.65, improving until ρ = 128 with AUC = 0.71. Indeed, by enlarging the PE network, DRAGON is provided with additional parallel CCLs for the similarity estimation circuit. Then, each CCL has independent capacity to discriminate and contribute to/from the common bank of centroids, based on different arriving nodes.

This method divides the risk of misclassification and eventually drives to better accuracy results. Next, we analyze the effects of directly modifying the main hyperparameters in our design which are presented in terms of % ω. In Figures 6(a) and 6(b) and their respective reports in Table 5, it can be observed that ϵ and φ both have negative slopes for speed-up.

In the case of ϵ, increasing the capacity for events per SC seems to minimally affect speed-up while strongly enhancing the accuracy of prediction. In the case of ϕ, the drop in speed is quite evident to the point of impairing the acceleration. Understandably, introducing an upholding delay to the system caused an exponential drop in speed. Additionally, in comparison with other parameters, accuracy does not improve by much.

On the other hand, both ω and α produced strongly positive slopes. For both cases, extending the number of recurrent stages to be considered at a single cycle or storing more SC/TC considerably improves speed-up. The reason for the first is quite evident since more on-chip resources are being added to a more immediate comparison process.

For the second, the explanation might be related to a richer set of centroids available for comparison, consequently producing earlier matching of ideal centroids. In terms of accuracy, the provision of a larger chain of stages reduces the introduction of uncertainty in the embedding esti-

Table 5. Numerical Report on the Effect of Key Parameters Variance in Terms of %Ω for (a) Speed-up (× Dynnode2vec) and (b) the Related Accuracy Levels (AUC)

(a)

Ω	ω (%Ω)	α (%Ω 102×)	ϵ (%Ω)	ϕ (%Ω 14×)
5	21.24	13.74	55.42	70.89
10	28.05	16.58	52.19	65.40
15	34.83	19.40	51.45	61.73
20	42.71	22.68	50.67	59.48
25	46.07	22.04	50.80	52.19
30	48.19	23.15	50.15	48.24
35	52.44	20.49	50.69	42.17
40	58.17	23.07	48.37	39.50
45	61.01	23.69	46.16	34.48
50	62.53	25.16	45.77	32.15
55	65.16	28.37	44.12	24.97
60	76.37	34.65	42.25	22.23
65	74.12	36.51	43.28	15.31
70	75.00	42.80	41.44	12.66
75	80.54	42.92	41.36	6.01
80	81.38	45.08	41.50	2.33
85	88.13	58.55	40.16	1.84
90	90.11	63.40	40.10	0.16
95	95.52	65.16	38.68	0.35
100	110+	66.96	39.17	0.08

(b)

Ω	ω (%Ω)	α (%Ω 102×)	ϵ (%Ω)	ϕ (%Ω 14×)
5	0.5047	0.5016	0.4660	0.5672
10	0.5322	0.5068	0.4842	0.5677
15	0.5464	0.5114	0.5208	0.5682
20	0.5710	0.5163	0.5214	0.5607
25	0.5751	0.5212	0.5440	0.5611
30	0.5857	0.5201	0.5506	0.5691
35	0.5892	0.5311	0.5672	0.5702
40	0.5908	0.5359	0.5738	0.5717
45	0.5917	0.5408	0.5804	0.5746
50	0.6056	0.5407	0.5970	0.5785
55	0.6144	0.5506	0.6036	0.5829
60	0.6298	0.5515	0.6102	0.5810
65	0.6292	0.5604	0.6234	0.5834
70	0.6392	0.5653	0.6366	0.5806
75	0.6341	0.5702	0.6581	0.5808
80	0.6433	0.5771	0.6685	0.5903
85	0.6500	0.5902	0.6799	0.5922
90	0.6594	0.5933	0.6807	0.5904
95	0.6637	0.5801	0.7013	0.5916
100	0.6782	0.5887	0.7146	0.5981

mations, considerably improving accuracy. The improvement when enlarging EVL with capacity for more events is also present, although is not as definitive as the previous parameter.

7.2 Physical Characteristics

In Figure 7(a) and its respective report in Table 6, we present the acceleration of DRAGON including centroid detection delay (training), relative to the results reported by PMFO-LP throughout different size configurations.

It is noticeable that for smaller ρ configurations and smaller datasets, our accelerator tends to perform much slower than PMFO-LP. In a single-core comparison, the lowest marker is in USAir at around −3.65× under PMFO-LP. This relation quickly improves as we provide proportionally larger datasets, with only around −0.56× under PMFO-LP for Router dataset.

Once again and as seen in 8.1 apart: algorithmic analysis, the number of PEs seems to be a strong determinant in performance. At $\rho = 8$, our accelerator is already able to catch up with PMFO-LP in terms of execution time, with 0.024× and 0.071× above PMFO-LP for Yeast and Router datasets, respectively. At $\rho = 64$ all dataset relations are positive and at $\rho = 128$ the best results are displayed, with the lowest score of 0.49× above PMFO-LP for USAir, and the highest of 1.12× above PMFO-LP for Router dataset.

Conversely, in Figure 7(b) its respective report in Table 6, we present the performance results once SCs have been fully detected and duly stored (trained). We can observe that in this case, the scalability relation becomes more balanced for both extremes.

In the worst case, with the smaller dataset and $\rho = 1$, DRAGON was able to execute the task at around −1.44× slower than PMFO-LP, this is more than three times the acceleration of the "training" modality. Under this modality, our accelerator was able to catch up earlier to PMFO-LP results for as little as $\rho = 4$. By $\rho = 16$, all the datasets have positive relations and the best results are exhibited at $\rho = 128$ for the Router dataset, at around 1.59× above the much more resourceful PMFO-LP.

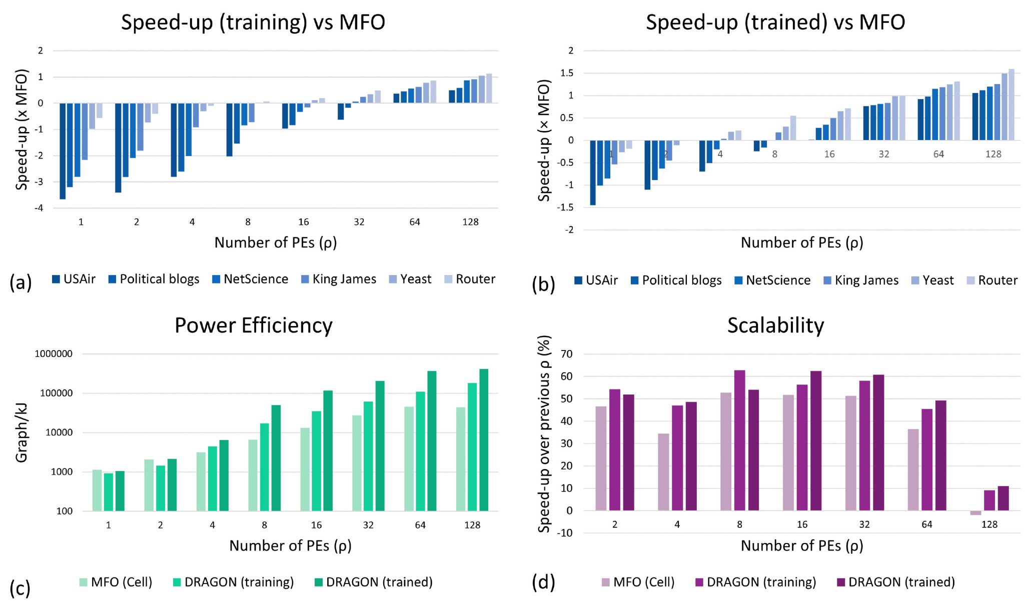

Fig. 7. Maximum achieved speed-up (a) including centroids training delay and (b) after completed centroids training. Magnitudes are relative to PMFO-LP (as reported) times for six different sizes of real-world datasets [43, 51]. (c) Power efficiency is represented in Graph per Kilojoule (logarithmic scale) for PMFO-LP and DRAGON (trained and training) at Router dataset [51]. PMFO-LP power demand is calculated from its core Cell microprocessor, according to its nominal characteristics [6], not including other power expenditures. (d) Scalability of PMFO-LP (as reported) and DRAGON (training and trained perspectives) at Router dataset. Scalability measures the speed-up over the previous smaller network configuration, represented in % above previous ρ. For all cases, design parameters set to: ω = 20%, α = 75% (102×), ϵ = 60%, and φ = 35 (14× cycles)%. Network sizes tested as $\rho = \{1, 2, 4, 8, 16, 32, 64, 128\}$.

These results demonstrate that, even though a single core of DRAGON cannot individually compete in speed against a single CELL processor from PMFO-LP, the scalability of our solution was able to ultimately surpass PMFO-LP when the right amount of hardware resources was granted.

Moreover, for a potential in-place online implementation, over a fairly stable sparse dataset, a centroid pre-trained system is perfectly practical. In this specific scenario, DRAGON was able to perform considerably better than its PMFO-LP counterpart in more than half of the cases. Notice that all these results are produced by only a fraction of the power required from PMFO-LP, and with constrained hardware resources.

In that regard, the definitive hardware benefits of our implementation are further reflected in Figure 7(c) and (d). In the first, it is reported the power efficiency calculated for PMFO-LP and obtained from DRAGON in trained and training modes. It is observable that for all three cases, increasing the number of parallel PEs steadily improves power efficiency, this indicates excellent energy-wise scalability.

In Figure 7(c), the estimated efficiency of PMFO-LP varies from 1,137 graph/kJ for ρ = 1, to 44,076 graph/kJ for ρ = 128. For DRAGON in training modality, we have managed to obtain efficiency from 6,319 graph/ kJ and 183,775 graph/kJ, at the previously described sizes. Meanwhile, DRAGON in trained modality becomes the best energy-wise solution, with 20,549 and 419,633 graph/ kJ for the described sizes.

Table 6. Execution Time (in milliseconds) as Reported by PMFO-LP at [3] Followed by DRAGON Under Training and Trained Modalities, for Six Real-World Temporal Datasets and Eight (ρ) Size Configurations

PEs (ρ)	Baseline	USAir	Political	NetScience	King James	Yeast	Router
1	PMFO-LP	10,130	13,000	23,842	25,692	55,192	81,790
	DRAGON (training)	47,176.828	54,600.273	90,616.503	81,040.326	109,098.578	127,602.950
	DRAGON (trained)	24,789.528	26,130.377	44,260.503	39,419.286	69,818.818	96,761.577
2	PMFO-LP	6,581.769	8,299.815	13,803.05	14,255.91	30,425.58	43,714.59
	DRAGON (training)	29,002.894	31,604.118	42,608.924	39,986.017	52,615.290	61,243.572
	DRAGON (trained)	13,864.876	15,683.853	22,421.909	20,686.465	33,691.522	44,205.854
4	PMFO-LP	4,055.407	5,040.127	8,684.661	9,302.966	19,852.52	28,623.92
	DRAGON (training)	15,416.800	18,157.359	26,158.971	17,804.118	25,841.552	31,504.631
	DRAGON (trained)	6,890.237	7,623.495	10,431.414	8,962.243	16,639.923	23,416.802
8	PMFO-LP	2,091.895	2,617.063	4,310.301	4,469.184	9,503.573	13,511.19
	DRAGON (training)	6,327.361	6,658.179	7,974.690	7,682.840	9,735.051	14,477.064
	DRAGON (trained)	2,593.972	3,039.975	4,268.245	3,793.990	7,258.303	8,727.905
16	PMFO-LP	960.335	1,193.372	2,134.429	2,213.149	4,721.341	6,526.128
	DRAGON (training)	1,886.205	2,189.410	2,829.593	2,560.907	5,294.780	5,445.228
	DRAGON (trained)	945.319	932.385	1,583.178	1,477.007	2,856.229	3,803.537
32	PMFO-LP	500.922	637.561	1,016.491	1,074.386	2,258.746	3,170.278
	DRAGON (training)	815.445	747.119	958.709	858.693	1,682.341	2,135.236
	DRAGON (trained)	284.269	356.194	560.227	586.188	1,139.056	1,592.064
64	PMFO-LP	292.392	367.435	623.965	658.324	1,384.313	2,014.626
	DRAGON (training)	213.661	252.510	397.389	406.387	776.355	1,083.808
	DRAGON (trained)	152.369	185.628	289.928	300.749	614.896	869.536
128	PMFO-LP	337.373	429.288	643.858	708.352	1,424.229	2,053.147
	DRAGON (training)	225.142	270.822	344.279	370.876	693.699	964.443
	DRAGON (trained)	163.855	202.550	292.379	313.576	571.682	790.615

Looking in detail at the speed-wise scalability report of Figure 7(d), we find that for each network size, PMFO-LP outperforms each of its previously tested configurations with relative stability. However, a sudden drop at its maximum $\rho = 128$, exposes scalability issues of their implementation. By contrast, DRAGON has scaled much better in all of its tested configurations including the largest one (although with a similar pace than PMFO-LP). In general, these results confirm that our parallel implementation is in general more scalable and stable than PMFO-LP.

In Figure 8(a), we present the definitive on-chip memory of DRAGON for the largest configuration tested at $\rho = 128$. The first includes a breakdown per memory sectors stored at DRAM and a circular distribution representation in % from the total on-chip memory utilization. It can be seen that the on-chip memory shared sectors, which are mostly destined for the ω window and buffering, are the largest division within DRAGON, approximately 63% over the totality.

Meanwhile, the actual structural and temporal operational demand including SEUs, TEUs, and MAUs, together both do not require more than 25% of the totality. This is expected since the key operational principle of ω window is inherently memory-intensive even when compared with the requirements of neural operations. Despite this difference, the utilization of ω window still represents a better investment in terms of total resources.

In Figure 8(b), we present the logic demand breakdown of DRAGON, according to the functional circuit served. Each functional group is represented in terms of ALMs. As around 90% of the chip is occupied by the NPs, the largest resource division is dedicated to the MAUs with around 60% of

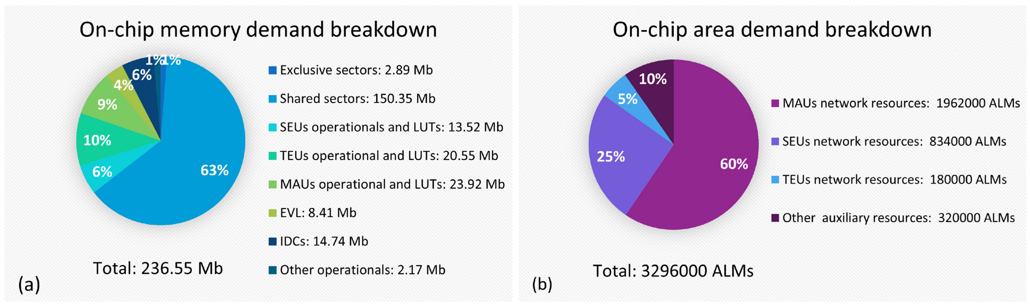

Fig. 8. (a) On-chip memory demand breakdown, per memory sectors in % regarding totality (circle), plus each sector resulting size in Mb (list). (b) On-chip area demand breakdown, per functional group in % regarding totality (circle), plus each group resulting number of ALMs (list). For all cases, at the largest size configuration $\rho = 128$ with design parameters set to: $\omega = 20\%$, $\alpha = 75\%$ (102×), $\epsilon = 60\%$, and $\varphi = 35$ (14× cycles)%.

the chip utilized while the second largest is related to the SEUs with 25%. Notice how only around 10% of the chip is required for the SPs and other auxiliary logic. This demonstrates the reduced burden that our proximity-based logic brings to the chip area.

We have also managed to abstract a large portion of MAC resources out of each structural processor (differently than originally conceived in ACE-GCN) into a distributed MAU service for multiple SEUs and TEUs. Finally, the estimated bandwidth requirements were adequately kept between 243 Gbps and 366 Gbps approximately (depending on the configuration and dataset), producing stable long-term operation at 267 MHz.

7.3 Functional Characteristics

In Figure 9(a), we can observe the response quality of DRAGON with the centroids updating capacity and without it (causing a full reset of centroids when meeting criteria). After stabilizing the centroids training stage at around 35% to task completion, the accelerator is suddenly fed with a structurally bipartite dataset. When including the centroid updating mechanism, we can observe that neural circuit utilization shortly rises at around 10% before dropping again to zero utilization, taking no longer than 10% of the total time to produce this recovery.

Meanwhile, when detaching the updating element that forces the recalculation of centroids, the neural utilization drastically tops +70% of utilization before starting the slow trajectory back to zero utilization, which occurs at around 80% of the processing time analyzed.

The results demonstrate that when DRAGON is detached from its capacity of centroid reutilization, an important difference in terms of computing complexity occurs. This confirms also that our accelerator is appropriately shielded against sudden dataset structural modifications, successfully inheriting the data-driven qualities from its static predecessor.

Finally, in Figure 9(b), we present the results of our accelerator concerning its multi-processing workload quality of distribution. The median standard deviation individually calculated for three cases $\rho = 8,32,128$ from the totality of SEUs involved, shows that in general, workload distribution is better for the smaller configurations. In the case of $\rho = 128$, even though the workload may seem somewhat unbalanced in the beginning, with around 75.64 points of deviation and populated with steep spikes, in general, it exhibits descending tendency.

A minimal is reached at 21.19 points at around 30% to completion time, after which deviation fluctuates between this and 43.84 maximum points for the rest of the processing time. According to these results, we may conclude that the workload distribution of DRAGON is satisfactory.

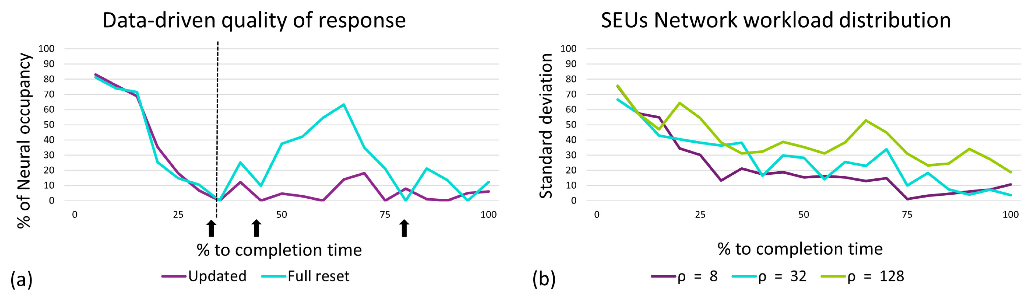

Fig. 9. (a) Data-driven quality of response against sudden dataset structural changes. Two configurations are studied: (dark blue line) with on-the-fly BSC/BTC updating capacity and (light blue line) without BSC updating capacity and BSR/BTR reset. Abscissa is % to task completion, ordinate is % of neural resources occupancy by direct embedding calculation, in opposite to embedding by estimation. The dotted vertical line indicates the completion time % from which a bipartite dataset starts to get introduced to the system. Small arrows along the abscissa indicate the zero-point utilization of neural resources. For both cases, tested at $\rho = 128$. (b) NPs network workload, quality of distribution. Abscissa is % to task finalization. Ordinate represents the median standard deviation based on individual workloads from all the NPs available at network sizes $\rho = \{8, 32, 128\}$. Design parameters set to $\rho = 20$ %, $\alpha = 75\%$ (102×), $\epsilon = 60\%$, and $\varphi = 35(14\times$ cycles) %.

8 CONCLUSIONS

In this work, we present DRAGON, a novel resource and energy-efficient hardware accelerator developed for high-speed dynamic link prediction. It is a powerful, scalable, and reliable FPGA-based solution aimed at real-time embedded sector with competitive accuracy and outstanding resource-efficiency results. In essence, we optimize a more resource-demanding PMFO-LP high-performance approach to graph link prediction by shortening the computation path of each PE in the network through the introduction of a smart temporal-structural reference and auto-completion system, which is, in turn, adapted from our previously developed static proximity-based accelerator into the current dynamic requirements.

In comparison with state-of-the-art multi-purpose CPU-based implementations, our customized FPGA-IP was able to surpass the faster of those considered (dynnode2vec) by roughly 191× for the largest size configuration tested $\rho = 128$ and with an improvement of 14.42% in accuracy. When directly compared against the high-performance physical implementation of PMFO-LP, DRAGON demonstrated to escalate well toward the inference when assigned enough resources. At $\rho = 16$, our implementation already surpasses PMFO-LP in every dataset, and at $\rho = 128$ it was able to exceed acceleration by 1.12× and 1.59× above PMFO-LP for the training and trained modes, respectively. At the maximum ρ, this drives an estimated power gain over PMFO-LP of around 71.49% and 89.59% for the training and trained case, respectively.

The superb speed-up acceleration, improved accuracy ratios, and strong scalability advantage of our implementation are explained from four perspectives: (1) Each PE within DRAGON is less computationally complex and more data straightforward than the pseudo-multi-purpose CELL microprocessor within PMFO-LP. (2) Further improved PMFO-LP approach to link prediction by exploiting the overlapping trajectories tendencies between moths. (3) Opportunities for fast convolutional estimations prior to the pre-processing stage. (4) In general, a more elaborated neural network and multi-processing configuration.

More importantly, due to its unique operational principles, DRAGON exhibits effective fault-tolerant shielding and data-driven functionalities nonexistent in PMFO-LP or any other link prediction implementation so far, adding our accelerator even more applicability into the high-speed embedded computing field.

REFERENCES

[1] Farzin Amzajerdian, Diego Pierrottet, Larry Petway, Glenn Hines, and Vincent Roback. 2011. Lidar systems for precision navigation and safe landing on planetary bodies. In *Proceedings of the International Symposium on Photoelectronic Detection and Imaging 2011: Laser Sensing and Imaging; and Biological and Medical Applications of Photonics Sensing and Imaging*, International Society for Optics and Photonics, Vol. 8192, SPIE, Beijing, 27–33.

[2] Hadi Banaee and A. Loutfi. 2015. Data-driven rule mining and representation of temporal patterns in physiological sensor data. *IEEE Journal of Biomedical and Health Informatics* 19, 5 (2015), 1557–1566.

[3] Reham Shawqi Barham, Ahmad Abdel-Aziz Sharieh, and Azzam Sleit. 2019. Multi-moth flame optimization for solving the link prediction problem in complex networks. *Evolutionary Intelligence* 12, 4 (2019), 563–591.

[4] Stephen Bonner, Amir Atapour-Abarghouei, Philip T. G. Jackson, John Brennan, Ibad Kureshi, G. Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. 2019. Temporal neighbourhood aggregation: Predicting future links in temporal graphs via recurrent variational graph convolutions. In *Proceedings of the 2019 IEEE International Conference on Big Data (IEEE BigData)*, Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay A. K, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye (Eds.). IEEE, Los Angeles, CA, 5336–5345.

[5] Jonathan Brogaard, Terrence Hendershott, and Ryan Riordan. 2014. High frequency trading and price discovery. *The Review of Financial Studies* 27, 8 (2014), 2267–2306.

[6] A. Buttari, P. Luszczek, J. Kurzak, J. J. Dongarra, and G. Bosilca. 2007. SCOP3: A rough guide to scientific computing on the playstation 3, Version 1.0. Innovative Computing Laboratory, Technical Report UT-CS-07-595, Version 1.0. Computer Science Department, University of Tennessee 595, 7 (2007), 1–77.

[7] Jinyin Chen, Jian Zhang, Xuanheng Xu, Chenbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. 2019. E-LSTM-D: A deep learning framework for dynamic network link prediction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 51, 6 (2019), 3699–3712.

[8] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). ACL, Doha, 1724–1734.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the Advances in Neural Information Processing Systems 29*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). Neural Information Processing Systems Foundation, Inc. (NIPS), Barcelona, 3837–3845.

[10] Luoyang Fang, Xiang Cheng, Haonan Wang, and Liuqing Yang. 2018. Mobile demand forecasting via deep graph-sequence spatiotemporal modeling in cellular networks. *IEEE Internet of Things Journal* 5, 4 (2018), 3091–3101.

[11] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD 2018, Yike Guo and Faisal Farooq (Eds.). ACM, London, 1416–1424.

[12] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steven K. Reinhardt, and Martin C. Herbordt. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*, IEEE, Athens, 922–936.

[13] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10) (JMLR Proceedings, Vol. 9)*, Yee Whye Teh and D. Mike Titterington (Eds.). JMLR.org, Chia Laguna Resort, Sardinia, 249–256.

[14] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187 (2020), 104816.

[15] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep embedding method for dynamic graphs. *CoRR* abs/1805.11273 (2018), 1–8. arXiv:1805.11273. http://arxiv.org/abs/1805.11273

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, San Francisco, CA, 855–864.

[17] Philipp Gysel, Jon J. Pimentel, Mohammad Motamedi, and Soheil Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 29, 11 (2018), 5784–5789.

[18] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark Horowitz, and Bill Dally. 2016. Deep compression and EIE: Efficient inference engine on compressed deep neural network. In *Proceeding of the IEEE Hot Chips 28 Symposium (HCS'16)*. IEEE, Cupertino, CA, 1–6.

[19] Shonosuke Harada, H. Akita, Masashi Tsubaki, Yukino Baba, Ichigaku Takigawa, Yoshihiro Yamanishi, and H. Kashima. 2018. Dual convolutional neural network for graph of graphs link prediction. *BMC Bioinformatics* 21, 3 (2020), 94–115.

[20] David J. Hill and Barbara S. Minsker. 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environ. Model. Softw.* 25, 9 (2010), 1014–1022.

[21] Md Jakir Hossain and Mahshid Rahnamay-Naeini. 2021. State estimation in smart grids using temporal graph convolution networks. In *Proceeding of the 2021 North American Power Symposium (NAPS)*. IEEE, A&M University Campus, TX, 1–5.

[22] Shixun Huang, Zhifeng Bao, Guoliang Li, Yanghao Zhou, and J. Shane Culpepper. 2020. Temporal network representation learning via historical neighborhoods aggregation. In *Proceeding of the 36th IEEE International Conference on Data Engineering (ICDE'20)*. IEEE, Dallas, TX, 1117–1128.

[23] José Romero Hung, Chao Li, Pengyu Wang, Chuanming Shao, Jinyang Guo, Jing Wang, and Guoyong Shi. 2021. ACE-GCN: A fast data-driven FPGA accelerator for GCN embedding. *ACM Trans. Reconfigurable Technol. Syst.* 14, 4 (2021), 21:1–21:23.

[24] Sabrine Khriji, Yahia Benbelgacem, Rym Chéour, Dhouha El Houssaini, and Olfa Kanoun. 2022. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *J. Supercomput.* 78, 3 (2022), 3374–3401.

[25] Thomas Kipf and M. Welling. 2016. Variational graph auto-encoders. arXiv:1611.07308. ArXiv abs/1611.07308 (2016), 1–3.

[26] Thomas Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. ArXiv abs/1609.02907 (2017), 1–3.

[27] Christian Leber, Benjamin Geib, and Heiner Litz. 2011. High frequency trading acceleration using FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'11)*, IEEE Computer Society, Chania, Crete, 317–322.

[28] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. 2019. GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks. In *Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications*, IEEE, Paris, 388–396.

[29] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett (Eds.). ACM, Chicago, IL, 177–187.

[30] Shibao Li, Junwei Huang, Zhigang Zhang, Jianhang Liu, Tingpei Huang, and Haihua Chen. 2018. Similarity-based future common neighbors model for link prediction in complex networks. *Scientific Reports* 8, 1 (2018), 1–11.

[31] Taisong Li, Jiawei Zhang, Philip S. Yu, Yu Zhang, and Yonghong Yan. 2018. Deep dynamic network embedding for link prediction. *IEEE Access* 6 (2018), 29219–29230.

[32] Shengwen Liang, YingWang, Cheng Liu, Lei He, LI Huawei, Dawen Xu, and Xiaowei Li. 2020. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Trans. Comput.* 70, 9 (2020), 1511–1525.

[33] Wanyu Lin, Shengxiang Ji, and Baochun Li. 2020. Adversarial Attacks on Link Prediction Algorithms Based on Graph Neural Networks. In *ASIA CCS'20: The 15th ACM Asia Conference on Computer and Communications Security*, Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese (Eds.). ACM, Taipei, Taiwan, 370–380.

[34] Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. 2022. K-Core based temporal graph convolutional network for dynamic graphs. *IEEE Trans. Knowl. Data Eng.* 34, 8 (2022), 3841–3853.

[35] John W. Lockwood, Adwait Gupte, Nishit Mehta, Michaela Blott, Tom English, and Kees A. Vissers. 2012. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *Proceeding of the IEEE 20th Annual Symposium on High-Performance Interconnects (HOTI'12)*. IEEE Computer Society, Santa Clara, CA, 9–16.

[36] Yecheng Lyu, Lin Bai, and Xinming Huang. 2018. ChipNet: Real-time LiDAR processing for drivable region segmentation on an FPGA. *IEEE Transactions on Circuits and Systems I: Regular Papers* 66, 5 (2018), 1769–1779.

[37] Sedigheh Mahdavi, Shima Khoshraftar, and Aijun An. 2018. Dynnode2vec: Scalable dynamic network embedding. In *Proceedings of the IEEE International Conference on Big Data (IEEE BigData'18)*, Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz (Eds.). IEEE, Seattle, WA, 3762–3765.

[38] Thomas Mealey and Tarek M. Taha. 2018. Accelerating inference in long short-term memory neural networks. In *Proceedings of the NAECON 2018—IEEE National Aerospace and Electronics Conference*, IEEE, Dayton, OH, 382–390.

[39] Tassilo Meindl, Walter Moniaci, Davide Gallesio, and Eros Pasero. 2005. Embedded hardware architecture for statistical rain forecast. *Res. Microelectron. Electron* 1 (2005), 133–136.

[40] Seyedali Mirjalili. 2015. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-based Systems* 89 (2015), 228–249.

[41] Apurva Narayan and Peter HO'N Roe. 2018. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine* 51, 2 (2018), 433–438.

[42] Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Proceeding of the Companion of the TheWeb Conference 2018 on TheWeb Conference 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, Lyon, 969–976.

[43] Link prediction group. Retrieved May 30, 2020 from http://linkprediction.org/index.php/link/resource/data.

[44] Céline Robardet. 2009. Constraint-based pattern mining in dynamic graphs. In *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM'09) (Miami, Florida, USA, 6-9 December 2009)*, Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu (Eds.). IEEE Computer Society, Miami, FL, 950–955.

[45] D. D. Šiljak. 2008. Dynamic graphs. Nonlinear Analysis: *Hybrid Systems* 2, 2 (2008), 544–567.

[46] Uriel Singer, Ido Guy, and Kira Radinsky. 2019. Node Embedding over Temporal Graphs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*. AAAI Press, Macao, 4605–4612.

[47] Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal. 2020. NERO: A near high-bandwidth memory stencil accelerator for weather prediction modeling. In *Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL'20)*, Nele Mentens, Leonel Sousa, Pedro Trancoso, Miquel Pericàs, and Ioannis Sourdis (Eds.). IEEE, Gothenburg, 9–17.

[48] Jaswinder Singh, SK Aggarwal, and Amrinder Kaur. 2017. Distribution transformer monitoring for smart grid using FPGA: A case study implementation in Indian conditions. *International Journal of Systems, Control, and Communications* 8, 4 (2017), 269–290.

[49] Aakash Sinha, Rémy Cazabet, and Rémi Vaudaine. 2018. Systematic biases in link prediction: comparing heuristic and graph embedding based methods. In *Proceeding of the Complex Networks and Their Applications VII - Volume 1 Proceedings The 7th International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2018 (Studies in Computational Intelligence, Vol. 812)*, Luca Maria Aiello, Chantal Cherifi, Hocine Cherifi, Renaud Lambiotte, Pietro Lió, and Luis M. Rocha (Eds.). Springer, Cambridge, 81–93.

[50] Daniel Stutzbach, Reza Rejaie, Nick G. Duffield, Subhabrata Sen, and Walter Willinger. 2006. Sampling techniques for large, dynamic graphs. In *Proceedings of the IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Joint Conference of the IEEE Computer and Communications Societies. IEEE, Barcelona, Catalunya, Spain, 1–6.

[51] Stanford University. Retrieved December 08, 2020 from https://snap.stanford.edu/data/index.html.

[52] Diederik Adriaan Vink, A. Rajagopal, Stylianos I. Venieris, and Christos-Savvas Bouganis. 2020. Caffe barista: Brewing caffe with FPGAs in the training loop. In *Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL'20)*, Nele Mentens, Leonel Sousa, Pedro Trancoso, Miquel Pericàs, and Ioannis Sourdis (Eds.). IEEE, Gothenburg, 317–322.

[53] Srinivas Virinchi and Pabitra Mitra. 2013. Link prediction using power law clique distribution and common edges distribution. In *Proceedings of the Pattern Recognition and Machine Intelligence - 5th International Conference, (PReMI'13). Proceedings (Lecture Notes in Computer Science, Vol. 8251)*, Pradipta Maji, Ashish Ghosh, M. Narasimha Murty, Kuntal Ghosh, and Sankar K. Pal (Eds.). Springer, Kolkata, 739–744.

[54] Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. 2007. Local probabilistic models for link prediction. In *Proceedings of the PReMI 7th IEEE International Conference on Data Mining (ICDM'07)*. IEEE Computer Society, Omaha, Nebraska, 322–331.

[55] NannanWang, Mingrui Zhu, Jie Li, Bin Song, and Zan Li. 2017. Data-driven vs. model-driven: Fast face sketch synthesis. *Neurocomputing* 257 (2017), 214–221.

[56] Shaorun Wang, Peng Lin, Ruihan Hu, Hao Wang, Jin He, Qijun Huang, and Sheng Chang. 2019. Acceleration of LSTM with structured pruning method on FPGA. *IEEE Access* 7 (2019), 62930–62937.

[57] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2020), 4–24.

[58] Yuxuan Xie, Bing Liu, Lei Feng, Xipeng Li, and Danyin Zou. 2019. A FPGA-oriented quantization scheme for mobilenet-SSD. In *Proceeding of the Advances in Intelligent Information Hiding and Multimedia Signal Processing. Number 2 in 156.* Springer, Jilin, 95–103.

[59] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? In *Proceeding of the 7th International Conference on Learning Representations (ICLR'19)*. OpenReview.net, New Orleans, LA, 1–17.

[60] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. HyGCN: A GCN accelerator with hybrid architecture. In *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, San Diego, CA, 15–29.

[61] Chaoyun Zhang, M. Fiore, and Paul Patras. 2019. Multi-service mobile traffic forecasting via convolutional long short-term memories. In *Proceedings of the 5th IEEE International Symposium on Measurements & Networking (M&N'19)*. IEEE, Catania, 1–6.

[62] Han Zhang, Gang Wu, and Qing Ling. 2019. Distributed stochastic gradient descent for link prediction in signed social networks. *EURASIP Journal on Advances in Signal Processing* 2019, 1 (2019), 1–11.
[63] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2022. Deep learning on graphs: A survey. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2022), 249–270.

Received 15 October 2021; revised 19 January 2022; accepted 4 March 2022